Simplification of FEM-models on Cell BE

Jon Mikkelsen Hjelmervik[1,2] and Jean-Claude Léon[2]

[1] Sintef ICT, Pb. 124 Blindern, N-0134 Oslo
jon.m.hjelmervik@sintef.no
[2] Grenoble Universités, Grenoble Institute of Technology, G-SCOP Laboratory, 46 av. Félix Viallet, F-38000
jean-claude.leon@inpg.fr

Abstract. Preparing a CAD model for Finite Element (FE) analysis can be a time-consuming task, where shape and mesh simplifications play an important role. It is important that the simplified model has the same mechanical properties as the original one, and that the deviation from the original stays within a given tolerance.
Most FE mesh simplification algorithms are either fully or partially sequential, and are therefore not suitable for architectures with high levels of parallelism. Furthermore, the use of processors such as GPUs of IBMs Cell BE requires algorithms to be adapted to benefit from their computational advantages. Here, we present an algorithm written for parallel processors, and its implementation for the Cell BE.

1 Introduction

Simplification of triangulations is often used in computer graphics, where it is used to reduce the complexity of 3D models before rendering. Often, the user specifies the target number of triangles desired. The simplification software aims at creating a triangulation with the given number of triangles that qualitatively resembles the original model.

FE Analysis (FEA) allows engineers to analyze properties like strength and heat conductivity in a simulated environment. The analysis can be performed on simple components, such as beams, or on complex models, and large assemblies such as models of entire airplanes. The components usually originate from 3D scanning or CAD systems. Even though CAD systems are integrating FEA tools into their tool chains, the process from a CAD model to a fully usable FE mesh remains a manual and time consuming process. In the automotive industry it can take up to four months to create a mesh for a car. In this text we use the terms mesh and FE mesh for shape representations adapted for simulation purposes.

CAD models use smooth shapes such as NURBS, sphere segments and torus segments to represent the surface of objects. Solid objects are represented by their boundary surfaces. Transforming their shape is often time consuming and tedious. FE meshes on the other hand are usually piecewise planar. Volume meshes, such as tetrahedral meshes, are often used to represent solid objects in

[0] Draft

FE analysis and mesh generators are powerful when the input model is locally compatible with the desired FE size. Model preparation brings the CAD model to a shape that is well suited for the meshing process. Later, the prepared model is used as a starting point for meshing. To improve and automate the model preparation, its starting point can be shifted from CAD models to triangulations, enabling also a larger range of shape changes. The domain (2D or 3D) being described by a triangulation, it is then subjected to a FE mesh generation process whose output is called the FE mesh. The target size for each finite element is dependent of the required accuracy. In addition, there may be a number of other important requirements for a mesh, including maximal/minimal angles, maximal valence and relative size of neighbor elements. These requirements are not meaningful for designers so even if a CAD model is exported into the right format, it will not necessarily be acceptable as a FE mesh. Hence, the need for a shape transformation process prior to a FE mesh generation.

During the last five years the trend in commodity hardware have gone from single-core processors to homogeneous multi-core or heterogeneous many-core processors. This evolution is driven by difficulties in developing faster cores as well as the evolution of highly parallel processors, e.g., Graphics Processing Units (GPUs).

The development in hardware has made research in new algorithms necessary, as most algorithms are unsuited for the new massively parallel architectures. During the past years the interest for such research has increased tremendously, and has grown into a new discipline in computer science called "General-Purpose Computing on GPUs" (GPGPU). Research in GPGPU has attracted the interest from researchers all over the world. Due to the increase in flexibility of GPUs more and more algorithms can be adapted to take advantage of them. The research has led to development of new algorithms as well as new uses of algorithms not commonly used. Bitonic sort by Baxter [2], which has been successfully implemented on GPUs by Purcell et al. [21], is one of many examples. For a thorough overview of the field please refer to [20].

Another trend in processor design is heterogeneous processors such as the *Cell Broadband Engine Architectures* (Cell BE) from Sony, IBM and Toshiba. The IBM Cell BE is targeting both supercomputers and the home entertainment market. In addition to one or more traditional processor core(s), called Power Processing Elements (PPEs), the Cell BE include a number of *thin* cores called Synergistic Processing Elements (SPEs). The current Cell BE consists of one PPE and eight SPEs. IBM are planning to release a updated version featuring 32 thin cores and two fat cores, yielding a total of one teraflop per second. In contrast to GPUs, the thin cores of the Cell BE operate independently, and can even execute different programs. This places the Cell BE between current GPUs and multi-core CPUs when it comes both to flexibility and performance.

In this paper we present an algorithm and its implementation on multi-core CPUs and the Cell BE. The goal of the algorithm is to yield high scalability beyond the eight cores we tested it for on the Cell BE. We had the 32 core Cell BE and GPUs in mind when developing the algorithm.

2 Related Works

Computer models of product components often contains numerous details that are part of the component shape "as-manufactured". To meet the objectives and hypotheses of component behavior simulation and reduce the time spent in the FEA process, it is advantageous that these details are removed. Venkataraman et al. [24] presented an algorithm for detecting and removing chains of blends, where the radius is "small" compared to the FE size. Similar strategies can be used also for details such as small holes and fillets, bosses, ... Meshing algorithms normally access surfaces through APIs, and do not fully benefit from working directly on the NURBS representation, which is complex to modify or transform. This led Owen et al. [19] to replace a CAD model by an adapted triangulation before meshing.

Shape simplification is important in computer graphics as well as in preparation of models for FEA. The wide range of applications has led to many different approaches to provide transformation operators, each useful for its applications. Gotsman et al. [10] give a good overview of the algorithms, and we will here describe some of the key features of selected algorithms and data structures.

Simplification algorithms are often categorized based on their restriction to height fields or parametric surfaces, and if they change the topology of the surface. The main focus here is methods operating on manifold surface-triangulations, which is often encountered in CAD models.

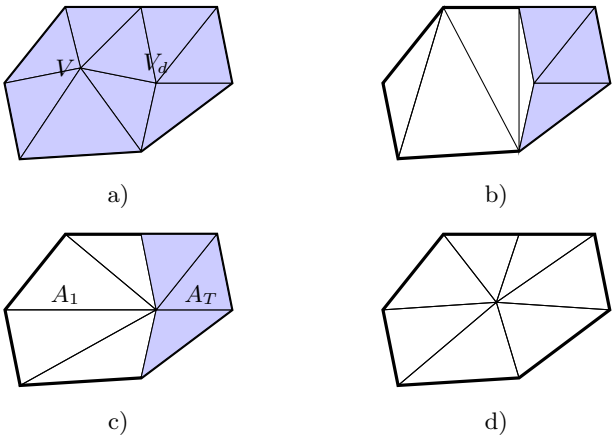

Fig. 1: Vertex removal operations, unchanged areas are marked in blue. a) original triangulation, b) arbitrary re-triangulation, c) half-edge collapse and d) edge collapse.

Most simplification algorithms are based on removing one vertex at a time, possibly grouping removals into passes. Before a vertex is removed, a number of decimation criteria are tested. The most common decimation criteria either

restrict the geometric deviation between the input and simplified models, or verifies that the topology of the surface is maintained. Figure 1 illustrates commonly used schemes for removing a single vertex V. Figure 1b) depicts a general vertex removal where the neighborhood surrounding the vertex V is remeshed freely. One example of this scheme is Schroeder et al. [23], where the remeshing is determined mainly by feature edges and aspect ratios. Hoppe [13] used edge collapse, where one edge is collapsed, placing its two vertices at the same position, leaving two triangles degenerated. This operation is illustrated in Fig. 1d).

Kobbelt et al.[16] introduced half-edge collapse, illustrated in Fig. 1c), where an edge is collapsed by moving *one* of the vertices V onto the other V_d. The main differences between edge collapse and half-edge collapse is that the latter has a smaller affected area A_1 and keeps the original vertex locations. As with edge collapse, half-edge collapse can be extended to support non-manifold surfaces.

Edge collapse is generalized to pair contraction, allowing non-neighboring vertices to be contracted. Isolated components can be connected, which may improve the visual quality of the simplified version. The method provides no guarantee regarding the topology of the resulting triangulation. Garland and Heckbert [9] used pair contraction in what has become one of the most commonly used simplification algorithms, "Surface Simplification Using Quadric Error Metrics" also known as QEM. In their work, vertices connected by an edge and vertices with a Euclidean distance less than a given tolerance are considered pairs. A decimation cost is assigned to each pair, based on a quadric error metric. The metric is compactly represented, using 10 scalars per vertex.

Rossignac and Borrel [22] proposed to cluster nearby vertices into a *representative vertex*, and remove all degenerated edges and triangles from the simplified triangulation. The clusters are usually chosen by assigning them a uniform grid of cells, and all vertices inside the same cell to a cluster. The representative vertex of a cluster can either be one of the existing vertices or chosen as a (weighted) average of the vertices inside the cluster. Lindstrom [18] showed that QEM also could be used in a vertex clustering setting to perform out-of-core shape simplification.

Lee et al.[17] presented an algorithm called MAPS for simultaneously decimating and parameterizing a triangulation. MAPS removes vertices iteratively. In each iteration, an independent set of vertices are removed and each newly removed vertex is localized, i.e. parameterized, in the new triangulation. In order to prioritize the removal of vertices over flat regions, MAPS prefers to remove vertices with low curvature.

Cohen et al. [4] proposed to create a simplification envelope around the triangulation and tests that the simplified version is inside the envelope. The envelope is constructed in such a way that it guaranties that the topology of the model remains unchanged and that the triangulation does not self-intersect.

Foucault et al. [7] proposed to base decimation criteria on properties with mechanical meaning. The general idea is to base the simplification process on properties that are known to have direct influence on the FEA process. Indeed, volume variation is closely related to mass variation and acts as a relevant criterion when

dynamics takes part to the FEA process. Monitoring the position of a center of mass can be also a meaningful criterion. Other examples of mechanically-based criteria include transformation of boundary conditions to modify a pressure distribution while preserving the same resulting force, ... However, the diversity of criteria is bound to mechanical parameters used as input of the FEA process, i.e. quantities related to the solution fields (stresses, strains, ...) of the FEA cannot be addressed by a strictly sequential FEA process (model preparation, mesh generation, solving). It should be noticed also that such criteria can become time consuming compared to shape simplification operators, hence they justify parallel treatments to process large models or to keep the simplification operations within an interactive timescale.

Shape simplification can be performed in parallel on shared memory parallel architectures, providing the vertices being removed at the same time are not too close to each other. A common strategy is to create a set of vertices that can be removed simultaneously without one vertex removal being influenced by any of the others. Such independent sets can be created in parallel. Dadoun and Kirkpatric [5] and Karp and Wigderson [15] presented algorithms that find independent set in polylogarithmic time (assuming a very high number of processors).

The main challenge for parallel algorithms is to find "good" independent sets in parallel. In a sequential setting, it is common to use greedy algorithms that use a cost function to guide the selection of vertices, and a "good" independent set is one where the decimation cost is low for all the vertices. Franc and Scala [8] proposed an algorithm for finding such a set without sorting the vertices, however their method for finding the independent set is sequential.

Botsch et al. [3], proposed to improve performance of evaluation of decimation criteria by using the GPU. In their work, the decimation criterion is expressed as threshold on the distance from the decimated surface to the original. The criterion is checked by sampling a piecewise linear approximation to the signed distance field attached to the original surface, and comparing the sampled value to the tolerance. In their work, the approximation to the signed distance field is computed using a CPU-based implementation of fast marching methods. This is transferred to a 3D texture in graphics memory. Then, the triangles that are to be checked are rendered using this texture. This is a first contribution to an efficiency increase in a decimation operator through the use of heterogeneous processor architectures.

Hjelmervik and Léon [12] presented an hybrid GPU-CPU simplification algorithm. In their work, the GPU performs the most computationally intensive tasks, while the CPU maintains the data structure. This means that data must be sent between the system memory and graphics memory, which imposes overhead. The overhead due to memory transfers makes this algorithm only feasible when decimation criteria with high arithmetic intensities are involved.

DeCoro and Tatarchuk [6] presented the first algorithm to perform the entire simplification on the GPU. They implemented vertex clustering using the geom-

etry shader to discard degenerated triangles. Since the GPUs support *stream out* of vertex data, the simplified version can be reused for further simplification.

Simplification algorithms often sort the vertices based on their decimation cost. Sorting is a fundamental building block for a wide variety of applications. Parallel sorting algorithms suitable for heterogeneous architectures have therefore attracted the attention from several research groups. GPUTeraSort by Govindaraju et al. [11] is successfully implemented on multi-core processors and GPUs, and AA-sort by Inoue et al. [14] has shown high speedups for multi-core processors and the Cell BE processor.

3 Description of the Algorithm

Our main goal is to develop a flexible algorithm for shape simplification that is suitable for many-core heterogeneous architectures. To take advantage of the high level of parallelism, it is important that a high number of vertices are removed simultaneously and that all steps in the algorithm can be performed in a parallel fashion. Existing thin cores do not have the capability to allocate or deallocate system memory, prohibiting implementation of algorithms that use dynamic memory.

The demand for performance improvement seems most crucial either when considering simplification of large models where computationally demanding decimation criteria are in play and/or when the simplification must be performed at an interactive rate. However, many applications use QEM or other computationally inexpensive decimation criteria. Performance may also be important for these applications, either to improve the interactivity or to simplify assemblies consisting of a large number of less detailed objects. Thus, low overhead is a key feature of our algorithm. Furthermore, the algorithm should be extendable to transfer properties attached to the faces (or edges or vertices) of the triangulation to enable the implementation of mechanical criteria like the transformation of pressure fields.

A simplified view of popular simplification algorithms is illustrated in Fig. 2a. Typically, an independent set of vertices, IS_v is created using a greedy, sequential algorithm, which inserts vertices one by one into IS_v. Such a set is used to ensure that no neighboring vertices are removed simultaneously. Indeed, neighboring vertices could be removed in the same pass as long as it does not occur at the same time. Since neighboring vertices may be candidates for removal in different threads, it becomes necessary to implement vertex removal in a thread safe manner. This is the approach implemented and figure 2b illustrates our algorithm. This is an alternative to the implementation a parallel algorithm for creating IS_v. With our approach, it means that multiple threads may perform vertex removals on the same triangulation, while maintaining the integrity of the data structure. If competing threads perform conflicting vertex removals, this situation must be detected and resolved. In our implementation, threads

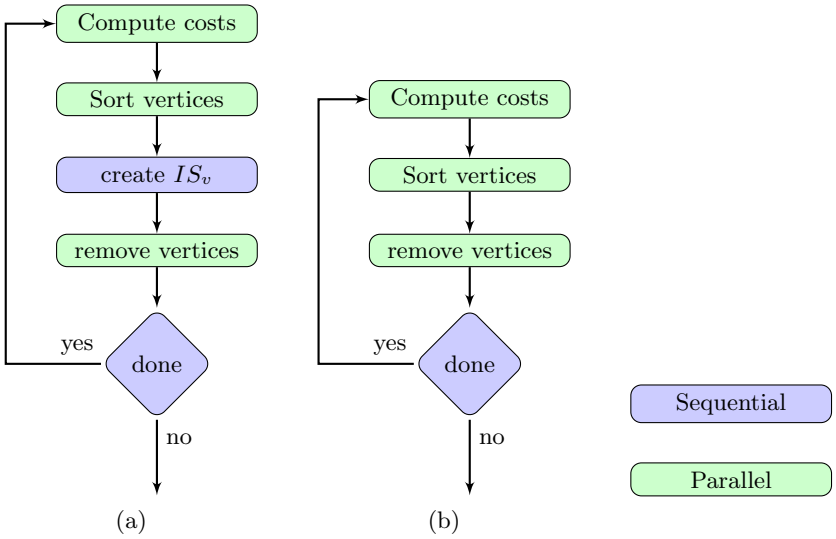

Fig. 2: Flowchart of simplification algorithms. Figure a) illustrates a traditional simplification algorithm. By removing the vertices as they are inserted into IS_v we get the flowchart illustrated in b).

detecting a conflict will ignore the current vertex removal and continue with another candidate vertex.

Serial simplification algorithms can iteratively remove the vertex with lowest decimation cost. The decimation cost can be updated as the neighborhood is modified, ensuring that the priority queue is up to date. In a parallel setting, the use of a priority queue would be a bottleneck, since any update would require exclusive access. Algorithms based on IS_v are guided by the decimation cost, but even the vertex with highest decimation cost can enter IS_v in the first pass. Requiring the candidate vertices to have a decimation cost less than a given threshold can improve this situation. In our algorithm, each thread performs the vertex removals in the order given by the decimation cost. However, the vertices are not reordered within one simplification pass. Depending on the objective of the decimation process, a well suited decimation criterion can prevent unwanted vertex removals, making the application less dependent of the decimation order and on the concept of threshold. Again, this shows what key concepts helped get around the use of IS_v.

Vertices not removed in the first simplification pass may be removed at a later pass. We use a *removable flag* assigned to each vertex, indicating the status of the vertex removal. Vertices failing the simplification criteria are marked as "not removable". This is not a permanent status, since further simplifications can change their neighborhood, hence their cost and status. However, a vertex failing the simplifications criteria once is less likely to be removed than vertices

not yet considered. As a compromise, we reconsider all vertices marked as "not removable" every three pass. To restrict the SPEs to only consider the removable vertices as candidate vertices, we assign removed and not removable vertices cost of -2 and -1 respectively. After the vertices are sorted in ascending order, the removable vertices are located at the top of the array.

A wide variety of data structures can be used to described and store triangulations. Due to its simplicity and efficiency, vertex incidence lists are often used for simplifications. In its simplest form, each vertex contains a list of pointers (or indices) to its neighboring vertices. A vertex removal operation performed on this data structure modifies not only the removed vertex V, but also to the vertices in its one-ring neighborhood because the adjacency relations must be updated. Thus, no other vertex in its two-ring neighborhood can be removed at the same time. Thread safe vertex removal operations using such a data structure has an increased risk of failing, especially when the number of vertices gets small, which is often the case when a fair proportion of vertices are removed.

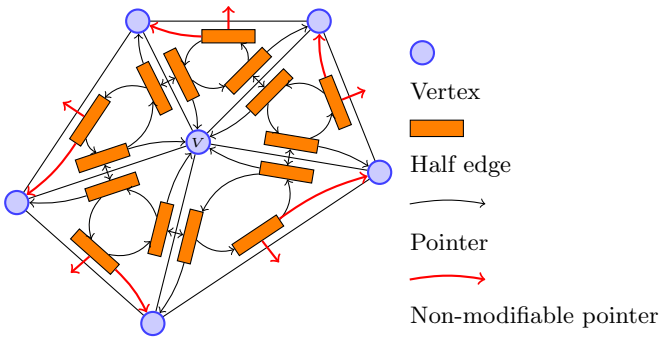

Fig. 3: Illustration of the half-edge data structure. Pointers belonging to boundary edges and therefore not modifiable are marked in red.

Another popular data structure for triangulations is based on half-edges. Figure 3 illustrates the data elements involved in a vertex removal. The edges surrounding the one-ring neighborhood of a vertex V stay unchanged during its removal. Therefore, we only read and modify the half-edges on the "inside" of this area during a vertex removal. These half-edges are only relevant for vertex removals of any vertex inside this one-ring neighborhood. Therefore, the half-edge data structure does not increase the radius of influence beyond where the triangles change.

Vertex removal operators can result in a locally topologically illegal configuration, such as two faces being mapped on top of each other. Before removing V it must be verified that no edge is added between already connected vertices. To perform this test, the area of interest is expanded from the one-ring neighborhood of V to the two-ring. For the (half-)edge collapse the area of interest

Listing 1: Thread safe vertex removal

```
Lock the potential vertex V
Check for conflicts
  Verify that the half-edges ‘‘return’’ to the center vertex V
  Verify that none of the neighboring vertices are locked
Check geometrical (and mechanical) criteria
Check topological criteria
Update data structure
  Update pointer to half-edges
  Update the pointers to the vertex V
unlock the potential vertex V
```

is expanded from the one-ring neighborhood of the V to also include the one-ring neighborhood of the destination vertex V_d. In Fig. 1c) the original area of interest is labeled A_1 and the added area labeled A_T. Due to the small area of interest and simple implementation, half-edge collapse is used in this work.

A thread safe vertex removal operation must detect if either the vertex itself or one of its neighbors is being removed by another thread. Thread safe locking mechanisms are available for most parallel architectures, and we added a lock for each vertex to the data structure to implement such a mechanism. As shown in Listing 1, we lock the potential vertex V as the first step of a vertex removal.. During the update of the data structure, we update the pointers to V as the last step, to ensure that competing vertices removal threads see that V is locked. During a vertex removal with the half-edge collapse operator, the data structure is locally inconsistent within the one-ring neighborhood of V. The inconsistencies take form either as three consecutive half-edges not forming a loop, i.e. a triangle is modified or collapses, or half-edges pointing at the incorrect vertex, i.e. the pointer to the half-edges are updated but not the pointer to the vertex. These inconsistencies can easily be detected, by verifying that there is no vertex locked and that three consecutive vertices form a triangle. If neither inconsistency or locked vertices are found, the vertex removal process can safely continue.

4 Cell BE Implementation

From a programming point of view, memory management related issues constitute the main differences between traditional homogeneous architectures and heterogeneous processors such as GPUs and the Cell BE. So far, we have described vertex removal operations using shared memory architectures with coherent caches. Traditional homogeneous multi-processor and multi-core systems fit this description. However, heterogeneous architectures such as GPUs and the Cell BE are not equipped with coherent caches. Therefore, algorithms implemented on such architectures should not rely on memory transfers being performed in the order they are issued. In this section we present our Cell BE implementation of thread safe vertex removal.

Instead of a hardware controlled cache, each SPE has 256KB of local memory called local store, used to store both code and data. A program running on a SPE

Listing 2: Simplified view of thread safe vertex removal on Cell BE

```
Lock the potential vertex V
While looping around the vertex V to collect neighbors
  Verify that the half-edges ‘‘return’’ to the center vertex V
  Verify that none of the neighboring vertices are locked
Check geometrical (and mechanical) criteria
Check topological criteria
Update data structure
Update the status of V
```

does not access system memory directly, but uses Direct Memory Access (DMA) instructions to copy data to and from system memory. DMA transfers are asynchronous, allowing the SPE to continue execution while data is being transfered. One way of managing the data transfers is by creating a software managed cache, using DMA transfers to and from local store. Such a cache normally contains functions to read/write data, initiate reading of a memory location (touching), and notifying to the cache that the content may be outdated (dirtying). In addition to DMA instructions, each SPE has an atomic unit. The main purpose of this unit is to allow atomic operations such as locking mechanisms. The atomic unit can also be used to create a software-based coherent cache. However, one should try to avoid the use of the atomic unit, as it easily can become a bottleneck in the application. Instead, we use a software-based cache for reading half-edges and vertices, and perform the corresponding write operations without using the software-based cache. Our modified algorithm uses the atomic unit for maintaining the status of a vertex. A vertex can have the following states: locked, removed, not removable, or removable, covering the needs for both the current decimation pass and the process between the passes.

An implication of lacking coherent cache is that we have no guarantee that memory transfers are performed in the order they are issued. The Cell BE has an option to obey the order of the issued commands, but this postpones each memory operation, causing unnecessary stalls. The basic data element in a SPE is a quadword, and data transfers of naturally aligned quadwords are performed as atomic operations. Based on the concept of atomic operations, we therefore store vertices and half-edges in quadwords to guarantee that we never read a partly updated vertex or half-edge. An half-edge is represented as three indices (four if face attributes are explicitly represented), while a vertex is represented as three floating point numbers and one index. Vertices and half-edges can therefore be stored in a quadword each, with no or little overhead. The updated algorithm is given in Listing 2.

Only pointers pointing towards deleted elements are modified, thus all partly updated triangles will have references to deleted elements (see figure 3). Therefore, partly updated triangles will either include a reference to a vertex marked for removal, a half-edge marked for removal or the half-edges will not form a loop. If any of these cases are detected, the vertex removal process ignores V, otherwise no conflict is present and the vertex removal continues. Since our cache

is not coherent, data may remain in a SPEs cache after its value is updated. Our data structure consistency tests will detect the cases where the data is partly updated and can either ignore the candidate vertex, or dirty the cache and re-read the inconsistent data.

In our test cases, the overall cache-hit ratio is 80–90%. The high cache-hit ratio is due to the fact that geometrically near data elements are likely to be located near each other in memory. In our test cases each triangle is initially defined by three subsequent half-edges. However, for the first vertex and half-edge read for a given candidate vertex V the cache-hit ratio is very low. This is because the candidate vertices are not treated in their natural order, but rather in a sorted order. Thus, the cache-hit ratio is almost unaffected by dirtying the cache at the beginning of each vertex removal operation.

The cost of a cache-miss is reduced if the processor is busy while the memory transfer takes place. This can be achieved by touching the cache before the data is needed. This strategy is only feasible if the program can continue execution before the data is transferred. In our algorithm this is not the case. Bader et al. [1] presented an alternative latency-hiding technique for the Cell BE. They use software-managed threads to let the SPE continue working after a memory request is issued. The SPEs do not have hardware support for quickly switching between threads. Therefore, the program itself is responsible for switching between the software-managed threads. We applied this strategy in our application, manually switching threads after a cache miss. To facilitate this behavior, we implemented two different cache touching functions in our software-based cache. One function only starts the required data transfer, while the other function also reports if the data element is already in cache. This allows us to only swap threads when an actual cache-miss is encountered. However, with the high cache-hit ratio in our algorithm this only gave us 2% speedup.

5 Results

To verify correctness, we implemented a multi-core version, without dynamic load balancing. Since we have not put any effort into optimizing the implementation, we do not include runtimes from this implementation. The main purpose here, is to highlight how the speedup scales up when the number of cell is increasing. For each parallelizable stage of our algorithm, we uniformly divide the vertices among the threads. The computation time seems to be sufficiently uniform to justify this choice. The same strategy is used for the Cell BE implementation, where the vertices are partitioned, and each SPE is responsible for a subset of the vertices.

Figure 4 shows the two test models before and after simplification. The vertex-cost is set to its discrete Gaussian curvature. The 90% vertices with lowest costs are considered as potential for removal. Potential vertices are removed if the maximal angular deviation from the previous model is less than 25 degrees

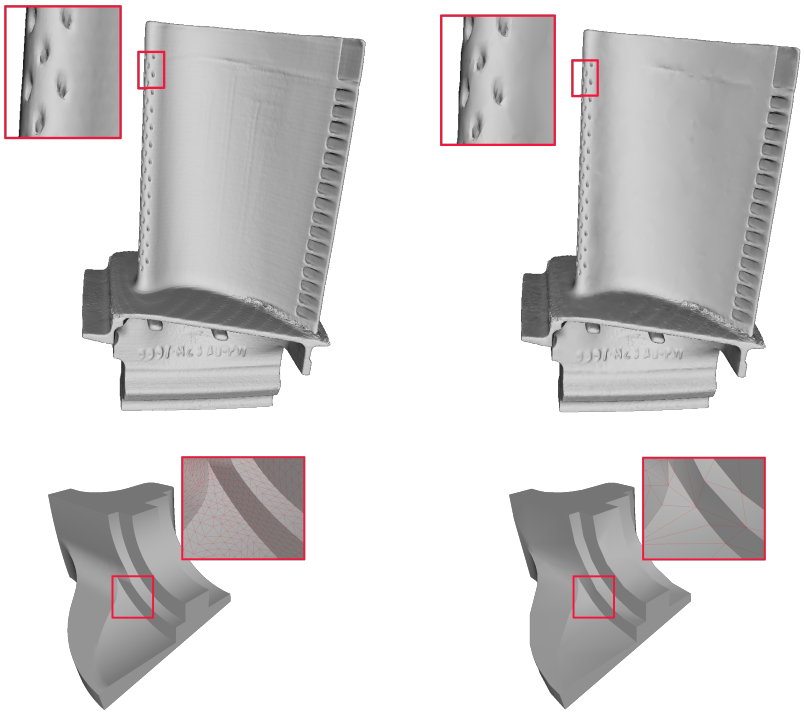

Fig. 4: Turbine blade (top) and fandisk model (bottom) before and after simplification.

and all interior angles in the resulting triangles are larger than 10 degrees. This yields good visual quality of the resulting triangulations. However, they do not provide any guaranteed error estimates. These criteria were chosen because of their balance between computations and memory operations. In addition, we include benchmark results where only topology tests are performed. Obviously, the resulting models are not usable, since the geometry is ignored. However, it shows that our algorithm performs well also for less computational intensive decimation criteria.

In the current Cell BE implementation the sorting is performed by the PPE. Parallel sorting algorithms are becoming a standard component for parallel architectures, and is a part of software development kits for GPUs and the standard C++ library distributed with the `gcc` compiler now contains parallel sorting. However, sorting is not yet a part of the API we used for the Cell BE. Implementing parallel sorting algorithms such as AA-sort by Inoue et al. [14] is outside the scope of our work. We therefore performed the benchmarks using a single-threaded sorting algorithm. For the turbine blade model in Fig. 4, sorting and other sequential parts of the code represents less than one percent of the

total runtime when using one SPE. The lack of parallel sorting implementation does not notably affect this benchmark, but must be remedied if the number of cores are to increase greatly.

Figure 5 illustrates the speedups achieved using a varying number of SPEs. The corresponding runtimes using one SPE is listed in Table 1. For the blade model, the speedup is almost linear for both benchmarks. The much smaller fandisk model does not achieve the same speedup when only topology is tested. This is due to the overhead caused by starting and stopping SPE threads.

Table 1: Runtimes of simplification, using one SPE on IBM QS21.

model	#vertices	runtime geometric	runtime topology
blade	882954	53.7s	9.0s
fandisk	6475	0.37s	0.05s

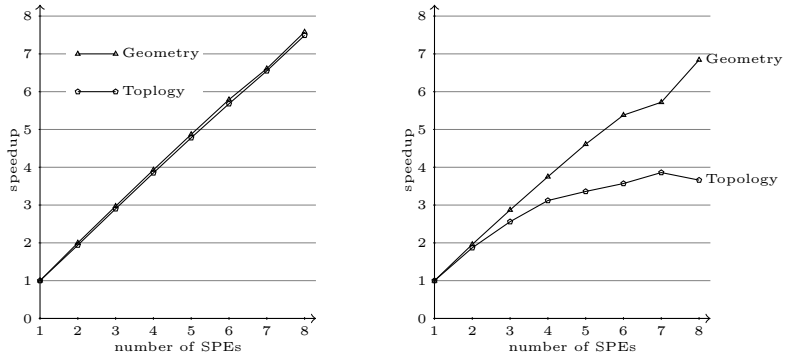

Fig. 5: Speedup of simplification of the turbine blade model (left) and the fandisk model (right). Benchmark of geometry-based simplification criteria and purely topological ones are presented.

6 Concluding Remarks

We have developed a flexible framework for shape simplification, suitable for heterogeneous many-core systems. The vertex removal is performed in a thread safe manner, allowing each thread to operate independently and eliminating the need of an independent set of vertices. The flexibility of the framework allows the inclusion of any decimation criterion based on information located inside the

one-ring neighborhood. Furthermore, properties associated to the faces can be propagated throughout the simplification process, thus allowing simplification criteria with guaranteed error estimates such as error spheres used by Véron and Léon [25].

Our benchmarks have shown that our framework performs well both for arithmetically intensive and memory intensive decimation criteria. Furthermore, it has sufficiently low overhead to be used in applications treating a large number of smaller models. The use of software-managed threads did not give us the expected performance increase. This may change if the software related to software-managed threads is optimized.

The strategy presented here can be used also for other operations performed on triangulations. Operations such as edge-flip and the Bowyer Watson algorithm for vertex insertion are candidates for this framework.

GPUs supporting the CUDA API perform data transfers of naturally aligned quadwords as atomic operations. Locking mechanisms are now also available on commodity GPUs. This opens up the possibility to develop a GPU-based implementation of our framework. Whether or not such an implementation is feasible is an open question, as the execution units of current GPUs operate in a synchronous manner. We will investigate this in a future article.

References

[1] D. A. Bader, V. Agarwal, and K. Madduri. On the design and analysis of irregular algorithms on the Cell processor: A case study of list ranking. In *Proc. of the 21st International Parallel and Distributed Processing Symposium*, pages 1–10. IEEE, 2007.

[2] K. E. Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computing Conference*, pages 307–314, 1968.

[3] M. Botsch, D. Bommes, C. Vogel, and L. Kobbelt. GPU-based tolerance volumes for mesh processing. In *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference on (PG'04)*, pages 237–243, Washington, DC, USA, 2004. IEEE Computer Society.

[4] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright. Simplification envelopes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 119–128, New York, NY, USA, 1996. ACM.

[5] N. Dadoun and D. G. Kirkpatrick. Parallel algorithms for fractional and maximal independent sets in planar graphs. *Discrete Appl. Math.*, 27(1-2):69–83, 1990.

[6] C. DeCoro and N. Tatarchuk. Real-time mesh simplification using the GPU. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 161–166, New York, NY, USA, 2007. ACM.

[7] G. Foucault, P. Marin, and J.-C. Léon. Mechanical criteria for the preparation of finite element models. In *Int. Meshing Roundtable, Williamsburg (USA), 20-22 September*, pages 413–426, 2004.

[8] M. Franc and V. Skala. Parallel triangular mesh decimation without sorting. In *SCCG '01: Proceedings of the 17th Spring conference on Computer graphics*, page 22, Washington, DC, USA, 2001. IEEE Computer Society.

[9] M. Garland and P. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH*, pages 264–270, 1997.

[10] C. Gotsman, S. Gumhold, and L. Kobbelt. Simplification and compression of 3d meshes. In *Tutorials on Multiresolution in Geometric Modelling*, pages 319–361. Springer, 2002.

[11] N. Govindaraju, J. Gray, R. Kumar, and D. Manocha. GPUTeraSort: high performance graphics co-processor sorting for large database management. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 325–336, New York, NY, USA, 2006. ACM.

[12] J. Hjelmervik and J.-C. Léon. GPU-accelerated shape simplification for mechanical-based applications. In *Shape Modeling International*, pages 91–102. IEEE Computer Society, 2007.

[13] H. Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 189–198, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[14] H. Inoue, T. Moriyama, H. Komatsu, and T. Nakatani. AA-sort: A new parallel sorting algorithm for multi-core SIMD processors. In *PACT '07: Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, pages 189–198, Washington, DC, USA, 2007. IEEE Computer Society.

[15] R. M. Karp and A. Wigderson. A fast parallel algorithm for the maximal independent set problem. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 266–272, New York, NY, USA, 1984. ACM.

[16] L. Kobbelt, S. Campagna, and H. peter Seidel. A general framework for mesh decimation. In *in Proceedings of Graphics Interface*, pages 43–50, 1998.

[17] A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. MAPS: multiresolution adaptive parameterization of surfaces. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 95–104, New York, NY, USA, 1998. ACM.

[18] P. Lindstrom. Out-of-core simplification of large polygonal models. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 259–262, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[19] S. J. Owen, D. R. White, and T. J. Tautges. Facetbased surfaces for 3-d mesh generation. In *Proc. 11 th Int. Meshing Roundtable*, pages 297–311, 2002.

[20] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Phillips. GPU computing. *Proceedings of the IEEE*, 96(5):879–899, May 2008.

[21] T. J. Purcell, C. Donner, M. Cammarano, H. W. Jensen, and P. Hanrahan. Photon mapping on programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 41–50. Eurographics Association, 2003.

[22] J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering complex scenes. In B. Falcidieno and T. L. Kunii, editors, *Geometric Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, 1993.

[23] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 65–70, New York, NY, USA, 1992. ACM.

[24] S. Venkataraman, M. Sohoni, and G. Elber. Blend recognition algorithm and applications. In *SMA '01: Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 99–108, New York, NY, USA, 2001. ACM.

[25] P. Véron and J.-C. Léon. Shape preserving polyhedral simplification with bounded error. *Computers & Graphics*, 22(5):565–585, 1998.