GPU-Accelerated Shape Simplification for Mechanical-Based Applications

Jon Hjelmervik
SINTEF ICT
Postbox 124, Blindern
0314 OSLO
jami@sintef.no

Jean-Claude Léon
ENSHMG-INPG Laboratory G-SCOP
Domaine Universitaire, BP 53
38041 GRENOBLE Cedex 9
Jean-Claude.Leon@hmg.inpg.fr

Abstract

In this paper we present a GPU-based method for removing shape details of 3D models. 3D models used in Finite Element Analysis (FEA) are often either constructed for the purpose of manufacturing, or a result of 3D scanning. The models therefore contain shape details that are neither important for FEA nor compatible with the mechanical hypotheses. Vertex removal is a popular method for removing geometrical details where vertices are removed one by one, provided certain constraints are satisfied. The constraints can either be based purely on geometrical properties, or also on mechanical ones.

The computations required in this process can be time consuming, especially if mechanical constraints are involved. The main idea behind our method is to perform the computations for all the vertices in parallel using graphics hardware, and then use the CPU to maintain the data structure representing the triangulation.

As a result, simplification functions can stay interactive while incorporating complementary mechanically-based criteria in addition to the geometric ones involved in shape transformation.

1. Introduction

3D models used as input of FEA are often constructed for the purposes of manufacturing, and therefore contain numerous details that are part of the component “as-manufactured”. To meet the objectives of the component behavior simulation and reduce the time spent in the FEA process, it is required that these details are removed. Expressing mechanical hypotheses defining the simplification of an analysis domain is mandatory for current simulations in the context of FE analyses. The corresponding shape adaptation for FE models is achieved by the elimination of shape sub-domains when their presence has no effect or, possibly, a weak effect on the mechanical behavior while imposing an undesirable local FE mesh density. Such sub-domains are referred to as details since they incorporate simultaneously mechanical as well as geometric meanings. Examples of these details include not only fillets, rounds which remove sharp edges for manufacturing purposes, but also shape features such as holes, small protrusions, etc.

Although modern CAD systems tend to integrate FEA tools in a design environment, generating FE models from design models remains tedious. In general, there is little or no feedback from CAD systems regarding the impact of modifications with respect to mechanical criteria. This is a major issue for integration of FEA into product design, and for development of objective criteria and/or constraints to help engineers prepare FE models. Several approaches have been proposed to ease the preparation of the FE models through detail removal operators [1], [18], [21]. Most of these approaches are based on NURBS model transformations, which have not led to strongly automated processes.

Another approach is to create an initial FE mesh prior to removing the unwanted details [16], [21]. In this approach, triangulation subsets forming details are removed by performing well-known mesh transitions, e.g. collapsing the faces of a tetrahedron to remove the element. However, these transformations are directly performed on a FE mesh and are not able to track shape changes. Thus, it is not possible to apply objective criteria to evaluate the mechanical influence of the induced shape transformations. In this setting, objective criteria are quantitative and based on input parameters of the shape, without contribution from the engineer's know-how.

Yet another approach separates the shape transformation stage from the FE mesh generation stage [7], [13], [14]. Modeling the initial shape as a polyhedron represented by a triangulation allows vertex removal or edge collapse operators to be used to simplify the model. This allows the generation of arbitrary shape changes. The method proposed here is based on this approach.

Vertex removal operators can be time consuming. In addition, if local mechanical criteria are included in the shape transformation, the consumption of each elementary operation is further increased. Thus, the detail removal processes may be too time consuming to be suitable for interactive applications. In this paper we present a method that uses graphics hardware to improve the performance of the vertex removal, even when criteria are associated with the shape transformation process. The main idea behind our method is to perform computations for all the vertices in parallel using graphics hardware, and then determine which vertices to remove.

After the introduction of programmable Graphics Processor Units (GPUs), only a few years ago, there has been an increasing interest for using GPUs for computations. The popularity is not only due to the performance of current GPUs, but also to the expected increase in performance difference between CPUs and GPUs. In this paper we will therefore study whether the GPU can be used to reduce the computational time for mesh simplification and which mechanically-based criteria can be associated with the local shape-change operators.

The paper is structured as follows. Section 2 summarizes the main characteristics of the proposed approach, section 3 reviews the works related to the current proposal. Then, section 4 describes the structure of the proposed algorithm, highlighting the overall architecture and the main steps of the shape transformation process. Section 5 gives a short introduction to GPUs. Furthermore it describes the data structures used in our method and the compromise between the desired efficiency improvement and the restrictions due to the GPU. Section 6 introduces numerical results, before concluding remarks are given in Section 7.

2. Framework of the proposed approach

To generate arbitrary shape changes, modeling the initial object as a facetted representation, i.e. a polyhedron, lets decimation algorithms produce a simplified shape, i.e. a simplified polyhedron, using either vertex removal or edge collapse operators. Then, this simplified shape serves as basis for the FE mesh generation process. This allows the shape transformation to extract sub-domains expressing mechanical hypotheses where mechanical criteria can be applied. During this shape simplification process, no constraint is assigned to the triangles of the polyhedron in order to produce a simplified shape solely based on the simplification criteria. The equilaterality criterion being a FE solver constraint, it is not taken into account at that stage.

This is the framework of the proposed approach and this helps to better characterize the user's hypotheses and their corresponding sub domains of the object.

Then, the simplified shape serves as basis for the FE mesh generation process, taking into account the equilaterality, size constraints and other mesh generation constraints. This approach helps to better characterize the user's hypotheses with respect to the FE solving process. In the work described here, only the shape simplification phase is addressed. We show how the shape simplification can be powered up using GPU architecture. This performance increase combined with mechanical criteria provides an extended user’s control during the shape simplification process.

During this phase, the main control criterion is based on the deviation of the simplified shape from the input one using a concept of variable envelope around the input shape. This concept drives the decimation criterion described in section 4.4. Indeed, the variable envelope is described in a discrete manner using spheres called error zones. These error zones are also related to the FE size that can be generated locally. Based on this relation the user, based on his/her know-how, can tune the envelope to the FE size that will be used later to generate the FE mesh. Therefore, the envelope acts as a mean to identify the areas acting as details from FE point of view.

3. Related Works

There has been a substantial amount of research invested in developing efficient vertex removal algorithms [15]. Initially, Schroeder and Yamrom [19] proposed an algorithm for decimating manifold surfaces. Each vertex removal consists of deleting all triangles connected to a vertex and triangulating the boundary loop left by the removed triangles. Only vertex-removals that satisfy given decimation criteria are performed.

Vertex removal algorithms come in many flavors [12], [8], [10], [4], which differ in how the order vertices are removed in, how the boundary loop is remeshed and the decimation criteria. Lee et al. [12] presented an algorithm called MAPS for simultaneously decimating and parameterizing a triangulation. Similarly to the method described above, 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.

Hoppe [10] proposed an algorithm for creating multi resolution models that can preserve predefined discontinuity curves. His method is based on computing an energy metric for each edge and then repeatedly performing the edge collapse with lowest energy. Garland and Heckbert [8] proposed an efficient method for performing mesh simplification. Their method uses quadric matrices to represent the simplification error.

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 mesh does not self-intersect.

Foucault et al. [7], proposed to base the decimation criteria on mechanical as well as geometrical properties. This way, the decimation algorithm better preserves the mechanical properties of the model as prescribed by the user through a priori criteria. Thus, providing a decimated model more suited for FEA. In this paper, we address how to improve the performance of such algorithms using graphics hardware as a first step before extending it to even stronger computer demanding criteria.

Botsch et al. [3] proposed to improve the performance of testing the decimation criteria by using GPUs. In their work, the decimation criterion is expressed as the distance from the decimated surface to the original one and this distance should not exceed a given tolerance. This criterion is checked by sampling a piecewise linear approximation to the signed distance field, 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. The signed distance field is stored in a 3D texture in graphics memory. Then, the triangles that are to be checked are rendered using this texture. Their work is a first contribution to an efficiency increase in decimation operators through the use of GPUs.

Adapting applications to take advantage of GPUs has become a popular research field, often referred to as “General-Purpose Computation using Graphics Hardware” (GPGPU). Among the applications that have been ported to GPU-based implementations, numerical solvers for PDEs are the most popular. This has led to the development of many different algorithms for solving linear systems, including the GPU-based implementation of the conjugate gradients method by Krüger and Westermann [11] an Bolz et al. [2]. GPUs have also successfully been used to solve PDEs using explicit schemes, e.g. Hagen et al. [9] For an overview over applications that have been successfully adapted for GPU-based implementation, we refer to Owens et al. [17].

The progress in the field of GPGPU led our interest towards using the computational strength of GPUs in the preparation process of FE models. The goal of our work is to allow shape simplification using computer demanding criteria in interactive applications. Apart from Foucault et al., all the current simplification algorithms are focusing on one shape-based criterion only, while we focus on allowing several criteria to participate in the process.

4. Algorithm Description

We propose a hybrid GPU-CPU algorithm where the data structure holding the triangulation is maintained in system memory, while the majority of the calculations are performed by the GPU. This allows us to use any of the typical data structures for triangulations while taking advantage of the computational power of the GPU. We found that the most computationally intensive tasks are related to validation of the decimation criteria, and remeshing the boundary edge loop of the removed faces. This is especially true when mechanically-based criteria are considered in addition to shape-based ones. Therefore, we move these computations to the GPU to reduce the computation time.

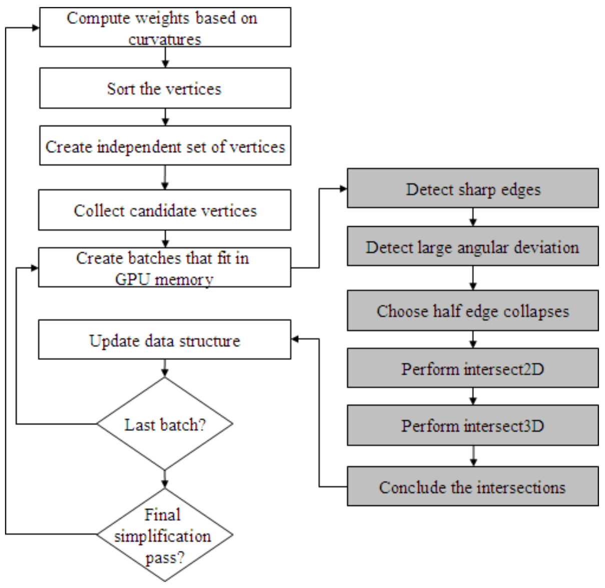
Compute weights based on curvatures
Sort the vertices
Create independent set of vertices
Collect candidate vertices
Create batches that fit in GPU memory
Update data structure
Last batch?
Final simplification pass?
Detect sharp edges
Detect large angular deviation
Choose half edge collapses
Perform intersect2D
Perform intersect3D
Conclude the intersections

Figure 1. Flow chart of the application. The gray operations are performed by the GPU and the white ones on the CPU.

4.1 Overview of the Algorithm

The flow chart given in Figure 1 describes the main steps in our algorithm. Similarly to MAPS, the algorithm performs the simplification in several passes, where an independent set of vertices are considered for removal per simplification pass. At each simplification pass, the vertices that potentially can be removed are sorted based on discrete curvature values. Then, a greedy approach is used to create an independent set of the sorted vertices. Here, an independent set means that the set does not contain any of the 1-ring neighbors of any vertex in the same set. The vertices in the independent set are then candidates for removal, and will be removed if they meet certain given criteria. The use of an independent set ensures that the removal of any vertex in the set does not influence the removal of other vertices in the same set.

Geometrical information about the candidate vertices and their 1-ring neighbors are collected and transferred to the GPU. Since this information may not fit in a texture, we split the candidate vertices into batches. Each batch is treated at the GPU, where the remeshing is computed and the criteria are evaluated. We will describe the GPU implementation in sections 4.3 and 4.4. Based on the computations performed at the GPU, the CPU removes a subset of the candidate vertices from the data structure of the triangulation.

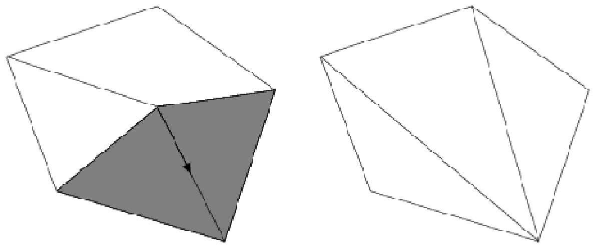
Figure 2. Principle of the half edge collapse.

4.2 Remeshing

The GPUs have little support for complex data structures, therefore our algorithm is based on half-edge collapse. This is a common remeshing scheme where one edge connected to the vertex is collapsed, i.e. the vertex is moved into the other vertex connected to the edge, as illustrated in Figure 2. The two triangles marked in the illustration collapses into two lines, and are removed from the triangulation. Such a choice is also compatible with FEA-based applications. Indeed, for most of the simplification passes the restriction in choice of remeshing imposed by the half-edge collapse does not affect the quality of the simplified model. However, with the range of applications considered here, the vertex removal process terminates either under distance-based criteria or mechanically-based ones rather than a targeted number of faces. This principle means that locally very coarse meshes can be produced while staying compatible with the mechanical requirements, e.g. local size of the FE required in the FE mesh generated after shape simplification process.

During these last stages of iteration, accessing the widest possible range of remeshing schemes is critical to widen the diversity of shapes that can be reached at the end of this process. The use of more general remeshing schemes at the end of the process is important and has a significant effect on the overall shape transformation process. General remeshing schemes can avoid too early termination while satisfying the distance constraints between the initial and simplified shapes as well as other mechanical-based criteria. Due to hardware limitations of the GPU, only half-edge collapse is performed at the GPU. Since general remeshing schemes only apply to a small subset of vertex removals, the time consumption is relatively small compared to the entire simplification process. Similarly, the half-edge collapse scheme addresses configurations equivalent to two-manifold ones. In the present contribution, it is considered that the non-manifold and surface boundary configurations are still handled by the CPU since they occur far less frequently than two-manifold ones.

4.3 Choose Half-edge Collapse

When a candidate vertex is considered for removal using half-edge collapse, one has the freedom to choose which edge to collapse. Our choice is based on properties relevant in a mechanical setting. Sharp edges are important properties of mechanical models and we therefore aim at keeping such edges in the approximated version. Therefore, we use the dot product between the normals of the two triangles at each side of an edge as the master value. Only the four edges with smallest master value are considered. Not only will this guide the remeshing process to the sharp edges, it also saves considerable amount of computation time when remeshing the neighborhood around a candidate vertex of high valence.

A half-edge collapse can result in a topologically illegal configuration, such as two faces being mapped on top of each other. CPU-based implementations can check for such configurations, and change the remeshing scheme of the candidate vertex. This is not feasible in our GPU-implementation, because such tests require topological information outside the 1-ring neighborhood of the candidate vertex. We therefore use geometric properties within the 1-ring neighborhood to detect if it is likely that a double face occurs. The angle between the new and old triangle connected to each boundary edge is used to detect illegal configurations. If this angle is larger than a predefined tolerance for any of the edges, the remeshing is likely to be topologically illegal or geometrically unwanted, and is therefore marked as unwanted. We found that an angular tolerance of 80 degrees detects almost all illegal situations, without preventing important vertex removals. This test does not replace a topology test performed at a later stage.

During a half-edge collapse, one can easily compute geometrical properties like the local volume variation. The volume variation of the model is an important property when performing FEA. It is one of the a priori objective criteria that we have addressed. It is an objective criterion because it can be quantified before the FEA takes place and it conveys mechanical meaning because it is directly linked to the mass of the object. The mass is an important mechanical property when dynamic behavior simulation is addressed. Similarly, volume variation is linked to variations in the centre of gravity and hence it is effectively at the basis of several a priori mechanically-based criteria. Not only global volume variations, but also local ones are important to characterize and to offer to the mechanical engineer. This allows him/her to prescribe volume variations over a sub-domain of the object according to his/her mechanical hypotheses. Local evaluation of volume variation can be used also to display its distribution over the simplified shape, thus providing qualitative information regarding the distribution of the volume variation. The local volume change is therefore used to choose among the edges selected through the master property (large angle between the connected faces).

4.4 Decimation Criteria

A vertex is only removed if the half-edge collapse fulfils the decimation criteria described here. The two-sided Hausdorff distance yields an accurate estimate of the geometrical simplification error. However, it is computationally intensive, and it is difficult to include this criterion in interactive applications where large models are used. A less computational intensive, but accurate criterion, is the one-sided Hausdorff distance from the original vertices to the decimated mesh. This is a common choice for simplification algorithms. It corresponds to assigning an error zone to each original vertex and testing if the simplified version intersects all error zones. Each error zone can be represented as a sphere with radius equal to the user-specified geometrical tolerance. However, error zones can also take mechanical criteria into account, by appropriately adjusting the radius of each sphere. To this end, the radii can be set equal to the FE map of sizes desired, enabling a priori information about the FEA to be used in the simplification process. In this case, the FE map of sizes is based on the user's know-how about the mechanical behavior of the structure and his/her ability to locate stress gradients. Further, the sphere radii can be set automatically based on a posteriori FEA criteria. In this case, the sizes of the error zones reflect the sizes of the finite elements required to match the analysis accuracy specified by the user. The sphere sizes can be defined using a strain energy error estimator based on a previous analysis to provide a new model for better FEA results [6], [5]. Here, the FE map of sizes is considered to be set up a priori only.

Each error zone is associated with one vertex and does not become active until the associated vertex is removed. The decimation criterion is satisfied if each active error zone intersects at least one triangle. The error zone is then assigned to one of the intersecting triangles. Note that the number of active error zones does not exceed the number of original vertices. This guarantees that the number of intersection tests to be performed at each simplification pass does not increase during the simplification process.

The use of error zones to incorporate a priori mechanical criteria allows the decimation process to take into account some of the aspects of FEA. However, it does not give the user objective information of how well the mechanical properties of the simplified version correspond to the original model. To incorporate this kind of information, we keep track of the global variation of volume. Since the local volume variation is already computed, it is only a matter of adding these values, and compare with the user defined global volume tolerance.

The decimation criteria performed by the GPU are restricted to be dependent on information in the 1-ring neighborhood of the candidate vertex. Any global criterion, such as variation of volume, must therefore be evaluated at the CPU using the volume calculations from the GPU. Further, topological criteria must be performed to ensure that the topology of the object is maintained. If a half-edge collapse is found to be topologically illegal it is omitted. Otherwise, if the candidate vertex has passed all tests described here, it is removed from the triangulation.

5. GPU Implementation

We will now describe the main features of our GPU implementation and the data structures used. Before describing the implementation, we provide a short description of the GPU and the programming model we used.

Since the GPU is designed to render triangulations, one might believe that the 3D capabilities can be used when implementing the decimation algorithm. However, the GPU is only designed to create 2D images of 3D models, and is not able to create 3D models. We therefore use the standard GPGPU approach, where the stream processing programming model is used.

Due to current limitations regarding the instruction set and number of registers, our GPU-implementation is divided into several steps. Each step is implemented in one or more rendering passes.

5.1. Graphical Pipeline

The GPU is designed to perform a sequence of tasks called the *graphical pipeline*. The graphical pipeline has four main stages:

- Vertex processing, where each vertex of the 3D model is transformed into the appropriate coordinate system.
- Rasterization, which determines which pixels are "hit" by a given triangle. From this point, each pixel is treated individually.
- Fragment processing, where the color of each pixel is calculated.
- Framebuffer operations, updates the framebuffer with the color, depth etc. from the incoming pixels.

A detailed description can be found in [20].

Recently, the vertex processor and the fragment processor have become fully programmable. Both the vertex processor and the fragment processor are data-parallel processors. The high performance of the processors is mainly because they contain a large number of *pipelines* that treat vertices/fragments simultaneously. Pixels treated simultaneously are treated in a Single Instruction Multiple Data (SIMD) manner. In this setting this means that exactly the same sequence of instructions are performed for all the pixels. This is important to keep in mind, especially when the fragment program includes dynamic flow control.

The fragment processor has access to data through one, two, or three dimensional arrays called *textures*. The framebuffer can be read as a texture at later rendering passes.

5.2. Programming Model

In the stream processing programming model input and output data streams and a computational kernel are specified. The kernel is then executed for all the elements in the data stream. Since the same kernel (fragment program) is executed for all the data elements (pixels), the stream processing programming model is well suited for GPGPU applications.

Example 1: Adding two matrices A and B, and storing the result in matrix C.

On the CPU, this is implemented as a double for-loop, which traverses all elements in the matrices and performs the computations sequentially, viz.

```
// instruction stream
for(i=0; i<numRows; i++)
 for(j=0; j<numCols; j++)
  C[i][j] = A[i][j] + B[i][j];
```

In stream processing, we first specify the two matrices we want to add and the matrix we want to store the result in. The next step is to load the computational kernel, before finally the computations are executed, viz.

```
// data stream
setInputArrays(A, B);
setOutputArrays(C);
loadKernel(  matrix_sum_kernel  );
execute();
```

Recently, the two major GPU vendors have released programming libraries that allows the GPU to be used as a computational resource, without going through the graphical system. We have chosen not to use these libraries at the moment, because they target specific hardware. However, we expect to port the simplification algorithm to take advantage of such APIs when they are more mature. There are currently several APIs being developed which allow the programmer to program against a stream processing programming model and all communication with the GPU are hidden from the programmer. Such libraries are likely to reduce the development time for GPGPU algorithms, and thereby further increase the interest for such implementations.

5.3. Data Structures

We use a standard C++ data structure for maintaining the triangulation in system memory. This data structure is used to retrieve the necessary information of all candidate vertices, which is then sent to the graphics memory as textures.

The two main textures are called **positionTex**, and **neighborsTex**, which contain the positions of the candidate vertices and their neighbors. Kernels used to determine the remeshing scheme and evaluate decimation criteria loop over the neighboring vertices. To ensure that vertices treated simultaneously spend the same number of iterations in these loops, the candidate vertices are grouped according to their valence. All vertices in the same row of **positionTex** are of the same valence. This simplifies the implementation and improves performance. Most the computations are performed by rendering into a two dimensional framebuffer of the same dimensions as **positionTex**. Each pixel in the framebuffer represents one vertex removal.

Example 2:

In this example the connectivity of the triangulation is represented as an adjacency list. For simplicity, only a subset of the vertices is presented here, and only the indices are given here.

Table 1: Adjacency list

Index	Valence	Discrete curvature	Neighbors
1	3	0.1	16, 7, 8
2	6	0.11	10, 3, 19, 18, 17, 9
3	5	0.12	10, 11, 4, 19, 2
4	7	0.13	11, 12, 16, 5, 20, 19, 3
5	6	0.15	13, 16, 6, 22, 20, 4
6	6	0.3	16, 7, 15, 23, 22, 5

Note that in Table 1 the vertices are already sorted according to their discrete curvature. The independent set will then consist of the vertices: 1, 2, 4 and 6, which become the candidate vertices. The candidate vertices are then grouped according to their valence, before the textures are created. The first row contains the vertex 1, which is the only candidate vertex of valence 3. The second row contains the vertices of valence 6, namely vertices 2 and 6, and the last row only contains vertex 4. Note that int thetables below the indices of the vertices are given instead of the positions.

Table 2: indices to the vertices in positionTex.

4	
2	6
1	

In the **neighborTex,** the positions of the vertices adjacent to the candidate vertices are stored. Each row contains the neighbors to the candidate vertices stored at the corresponding row in the position texture. Table 2 and Table 3 show the contents of **positionTex** and **neighborTex** respecivly.

Table 3: indices to the vertices in neighborTex

11	12	16	5	20	19	3					
10	3	19	18	17	9	16	7	15	23	22	5
17	7	8									

5.4 Ordering the Edges

A kernel computing the master values described in Section 4.3 is executed for all candidate vertices with valence at least equal to four. The four edges with lowest master value are candidates for half-edge collapse and their indices are returned from the kernel. Since we need to compute the volume variation for each of the potential edges, we perform the following computations in a framebuffer where each pixel corresponds to a potential edge. To simplify the implementation, we first rotate the list of neighbors, such that the destination vertex in the half-edge collapse is placed first of the neighboring vertices. With these preparations the local volume variation associated with each potential edge can efficiently be computed in a fragment program. The only remaining operation before choosing which edges to collapse is to compute the angles between the old and new triangles, to detect flipped triangles. When the edges to collapse are chosen, we finally rotate **neighborTex**, according to the chosen edges.

Example 2 continued:
To illustrate the rotation, Table 4 show **neighborTex** after the reordering. Given the half-edge collapses where vertex 1 is collapsed into vertex 7, 2 into 3, 6 into 16 and 4 into 11. The first element of the first row is the destination vertex for the half-edge collapse for vertex 1. Therefore, the neighbor vertices to vertex 1 are rotated one element to the left, viz.

Table 4: neighborTex after reordering.

11	12	16	5	20	19	3					
3	19	18	17	9	10	16	7	15	23	22	5
7	8	16									

5.5. Evaluation of Decimation Criteria

We focused on the decimation criterion based on error zones, because this allows the use of both a priori and a posteriori criteria. During the first simplification pass, i.e. before any vertex is removed, there is only one error zone per candidate vertex. This error zone is then attached to one of the new triangles. During later simplification passes error zones originating from already removed vertices must be validated.

To this end, the intersection tests are performed by two kernels called **intersect2D** and **intersect3D**. First, **intersect2D** performs a simple 2D intersection test. Then, **intersect3D** performs the intersection test for the triangle-sphere pairs that passes the 2D test. The **intersect2D** kernel is executed for each triangle and loops over the spheres assigned to the candidate vertex. The return value is an integer value, where each bit represents an intersection test and is set to one if the box test is successful. Since most GPUs today do not support integral data types or bitwise operations, this is implemented using floating point arithmetic. Then, **intersect3D** uses the bit pattern when looping over the spheres that potentially intersect the triangle. A large difference in the number of associated error zones will lead to poor performance, due to the SIMD architecture. Therefore, candidate vertices with a high number of associated error zones are repeated in **positionTex**, with different error zones.

Finally, a kernel called **concludeKernel** checks the results from the intersection tests, verifying that each sphere intersects at least one triangle. Otherwise, the candidate vertex is marked as not removable. The output from this kernel is the index to the destination vertex of the edge collapse, and the volume of the approximation error associated with this half-edge collapse. If the candidate vertex is marked as not removable, a negative index is returned.

5.6. CPU Treatment

The output from **concludeKernel** and **intersect3D** are read back to system memory for the final processing. The decimation criteria performed by the GPU are restricted to be dependent on information in the 1-ring neighborhood of the candidate vertex. Any global criterion, such as total variation of volume, must therefore be evaluated at this stage. Further, tests are performed to ensure that the topology of the model is maintained. Finally, the triangulation is updated and the assigned error zones are reassigned to new faces according to the results from **intersect3D**.

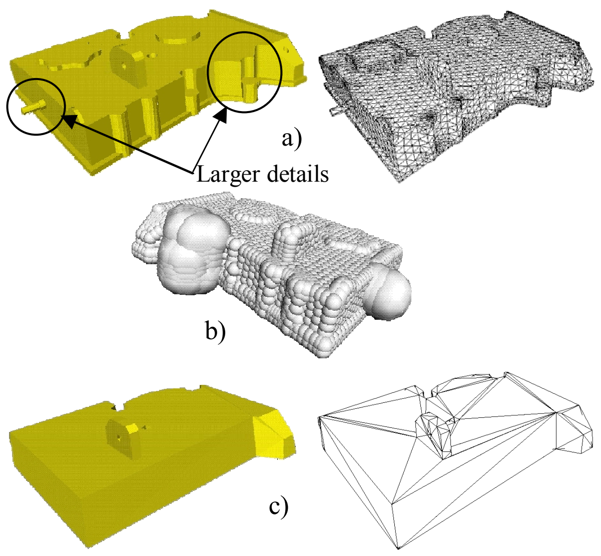

Figure 3: Example highlighting the effect of variable deviation distance on the shape simplification process. a) the initial model shaded and tessellated, b) the map of sizes used for the simplification process, c) the resulting model shaded and tessellated where larger details have been removed.

6. Results

In figure 3, an example is given to highlight the influence of the error zones on the shape simplification process. The variable size of error zones interactively specified by the user helps identify specific sub-domains as details that will be removed. Such a variable envelope adds further flexibility to meet the user's requirements.

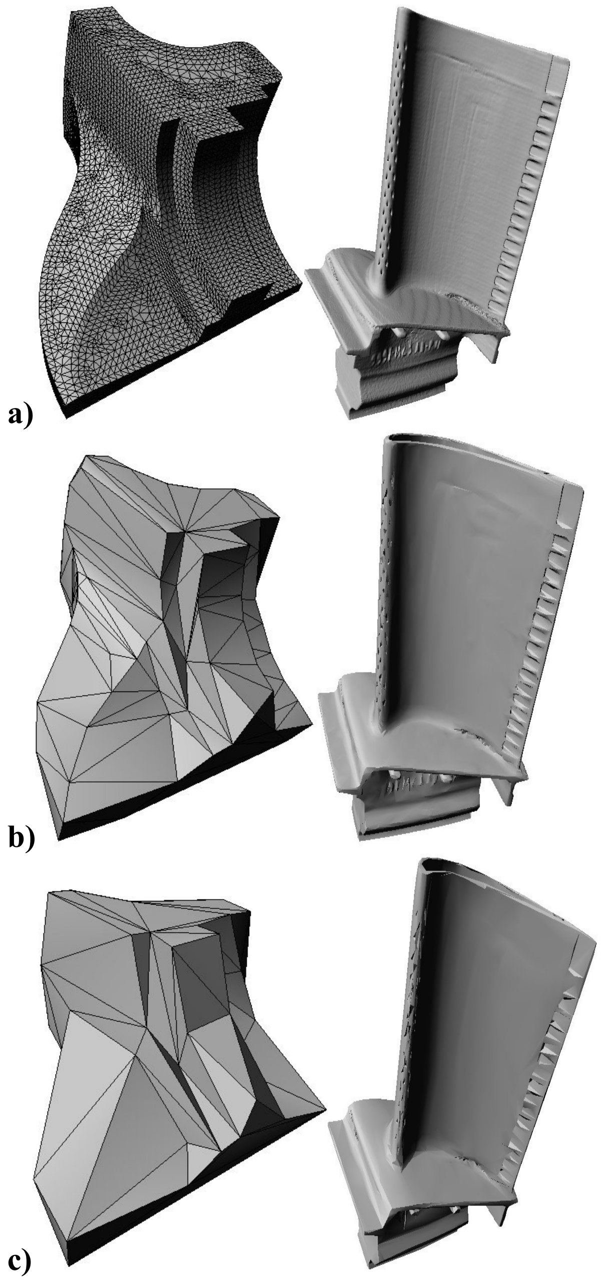

Figure 4: Original models (a), simplified models with active global volume criterion (b) and without active global volume criterion (c). Fandisk models are shown to the left, and blade models to the right.

The examples presented here are decimated with our GPU-accelerated method. Figure 4 shows how changing the tolerance for global volume variation influences the simplified model.

The performance advantages of using the GPU are illustrated in Table 5, which shows that our GPU version is 8 times faster than our CPU version for large models. The performance improvement with our GPU version is reduced for smaller models, but the algorithm outperforms the CPU version even for a model with only 12000 triangles. The models were processed using an AMD Athlon 4400+ CPU and a NVIDIA 7800GT GPU.

Table 5: Runtimes for simplification of mechanical models. The error zones are given relative to the length of the diagonal of the bounding box, and the volume is relative to the bounding box.

Model	Initial #tris	Final #tris	Error zone	Volume tolerance	CPU time (s)	GPU time (s)
Blade	1765388	10818	1.3%	1.7e-6%	71.4	9.7
Blade	1765388	25830	1.3%	1.7e-5%	87.6	10.8
Fandisk	12946	148	1.3%	1.5e1%	0.583	0.21
Fandisk	12946	128	1.3%	1.5e-3%	0.57	0.21

7. Concluding Remarks

In this paper we have presented a hybrid GPU-CPU approach for shape simplification of objects dedicated to mechanical applications. The computational power of the GPUs makes it feasible to include both geometrical and mechanical criteria. These criteria are too computationally expensive for pure CPU implementations to preserve the user's interactivity. Since the candidate vertices are split into batches, the independent set of vertices must be created by the CPU. This prevents us from computing the independent sets on the GPU without extra data transfer between the graphics and system memory.

The current implementation is based on accessing the GPU through OpenGL. We expect a performance improvement if it is ported to an API that allows the GPU to be used without going through the graphical system. Such APIs can increase the performance of the application and reduce the development time, making approaches like the one presented here more attractive in commercial systems.

The proposed approach has highlighted the efficiency of the GPU architecture where significant speed-up factors can be reached. Similarly, the a priori criteria implemented here is a first step toward the development of more computationally demanding criterion. The next step of the project will be to include stiffness based criteria involving computation and propagation of stiffness matrices.

The use of mechanical properties during the simplification process reduces the time the engineers spend preparing models for FEA, and give them objective information about the impact of their modifications or hypotheses. This is also a contribution to new tools helping engineers to monitor quantitatively their structure simulations.

Acknowledgments

This work has been performed in the framework of the AIM@SHAPE (Advanced and Innovative Models And Tools for the development of Semantic-based systems for Handling, Acquiring, and Processing knowledge Embedded in multidimensional digital objects) European network of excellence, contract n°506766 (http://www.aimatshape.net/). The current work has been performed in the context of a partnership between SINTEF, the University of Oslo in collaboration with INPG.

8. References

[1] C.G. Armstrong, D.J. Monaghan, M.A. Price, H. Ou, and J. Lamont. "Integrating CAE concepts with CAD geometry", In *Engineering Computational Technology edited by B.H.V. Topping, Saxe-Coburg Publications*, 2002, pp 75-104.

[2] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder, "Sparse matrix solvers on the GPU: Conjugate gradients and multigrid" *Computer Graphics SIGGRAPH 03 Proceedings, 2003.*

[3] M. Botsch, D. Bommes, C. Vogel, and L. Kobbelt. "Gpu-based tolerance volumes for mesh processing". *Proceedings of the Computer Graphics and Applications, 12th Pacific Conference on (PG'04)*,2004, pp 237–243.

[4] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright, "Simplification envelopes", *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques (SIGGRAPH 96)*, 1996, pp 119-128

[5] R. Ferrandes, P. M. Marin, J.-C. Léon, F. Giannini, "Evaluation of simplification details for an adaptive shape modelling of components", *Int. Conf. ECT, September 5-7th, Gran Canaries, Spain, September*, 2006.

[6] L. Fine, L. Rémondini, J-C. Léon, "Automated Generation of FEA models through idealization operators", *Int. J. For Num. Meth. In Eng.*, Vol. 49, n°1-2, 2000, pp 83-108.

[7] G. Foucault, P. Marin, and J.C. Léon, "Mechanical Criteria for the Preparation of Finite Element Models", *Int. Meshing Roundtable, Williamsburg (USA), 20-22 September*, 2004, pp 413–426.

[8] M. Garland, and P. S. Heckbert. "Surface simplification using quadric error metrics", *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 1997, pp. 209–216.

[9] T. Hagen, J. Hjelmervik, K.-. Lie, J. Natvig, and M. Henriksen, "Visual simulation of shallow-water waves." *Simulation Modelling Practice and Theory vol. 13*, 2005, pp 716-726

[10] H. Hoppe, "Progressive meshes" *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (SIGGRAPH 96), 1996, pp 99-108.

[11] J. Krüger, and R. Westermann. "Linear algebra operators for GPU implementation of numerical algorithms", *ACM Trans. Graph.*, 22(3), 2003, pp., 908–916.

[12] A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. "Maps: Multiresolution adaptive

parameterization of surfaces", *Computer Graphics Proceedings (SIGGRAPH 98)*, 1998, pp. 95–104.

[13] J.-C. Léon, and L. Fine, "A new approach to the Preparation of models for F.E. analyses", *Int. Journal of Comp. Appl., Vol. 23, n°1, 2, 3*, 2005, pp 166-184.

[14] D. Lesage, J-C. Léon, P. Véron, "Discrete curvature approximations and segmentation of polyhedral surfaces", *Int. J. of Shape Modelling*, Vol. 11, n°2, 2005, pp 217- 252.

[15] D.P. Luebke, M. Reddy, J.D. Cohen, A. Varshney, B. Watson, and R. Huebner, "Level of detail for 3D graphics", Morgan Kaufmann Publishers, 2003.

[16] Mobley, V., Carroll, M. P., Canann, S. A., "An object oriented approach to geometry defeaturing for Finite Element Meshing", *Proceedings of 7th International Meshing Roundtable, Sandia National Laboratories*, 1998.

[17] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell. "A survey of general purpose computation on graphics hardware", *Computer Graphics Forum*, Vol. 26, 2007, pp 80–113.

[18] M. Rezayat. "Midsurface abstraction from 3d solid models : General theory and application", CAD, 28(11), 1996, pp 905-915.

[19] W. J. Schroeder and B. Yamrom. "A compact cell structure for scientific visualization", *SIGGRAPH '94 Course Notes CD-ROM, Course 4: Advanced Techniques for Scientific Visualization*, 1994, pp 53–59.

[20] D. Shreiner, M. Woo, J. Neider, and T. Davis, "OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2 (5th Edition) (OpenGL)", 2005

[21] D. R. White, R. W. Leland, S. Saigal, and S. J. Owen, "The meshing complexity of a solid: an introduction", *Proc., 10th Int. Meshing Roundtable, Sandia National Lab.*, 2001, pp 373-384.