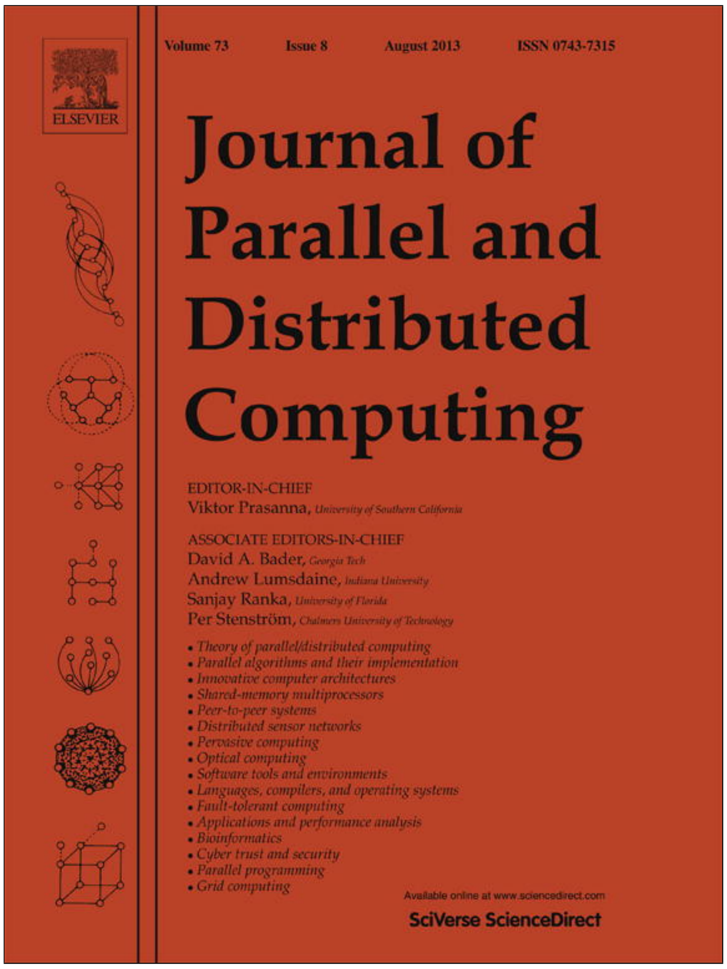

Volume 73
Issue 8
August 2013
ISSN 0743-7315
ELSEVIER
Journal of Parallel and Distributed Computing
EDITOR-IN-CHIEF
Viktor Prasanna, University of Southern California
ASSOCIATE EDITORS-IN-CHIEF
David A. Bader, Georgia Tech
Andrew Lumsdaine, Indiana University
Sanjay Ranka, University of Florida
Per Stenström, Chalmers University of Technology
• Theory of parallel/distributed computing
• Parallel algorithms and their implementation
• Innovative computer architectures
• Shared-memory multiprocessors
• Peer-to-peer systems
• Distributed sensor networks
• Pervasive computing
• Optical computing
• Software tools and environments
• Languages, compilers, and operating systems
• Fault-tolerant computing
• Applications and performance analysis
• Bioinformatics
• Cyber trust and security
• Parallel programming
• Grid computing
Available online at www.sciencedirect.com
SciVerse ScienceDirect

J. Parallel Distrib. Comput. 73 (2013) 1127–1134

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

A CPU–GPU framework for optimizing the quality of large meshes

J.P. D'Amato [b,c,*], M. Vénere [a,c]

[a] CNEA - National Atomic Energy Commission, Argentina
[b] CONICET - National Scientific and Technical Research Council, Argentina
[c] UNICEN - National University of the Center of Buenos Aires Province, Argentina

HIGHLIGHTS

- The development of a framework to optimize tetrahedral meshes in parallel.
- A proposal of a family of local topology operators that improve the mesh quality.
- An implementation of a non-Laplacian smoothing method on the GPU.
- A scheduling algorithm for processing and updating mesh structures in parallel.

ARTICLE INFO

Article history:
Received 24 May 2012
Received in revised form
1 March 2013
Accepted 18 March 2013
Available online 26 March 2013

Keywords:
Parallelism
Re-meshing
Quality
GPU

ABSTRACT

The automatic generation of 3D finite element meshes (FEM) is still a bottleneck for the simulation of large fluid dynamic problems. Although today there are several algorithms that can generate good meshes without user intervention, in cases where the geometry changes during the calculation and thousands of meshes must be constructed, the computational cost of this process can exceed the cost of the FEM. There has been a lot of work in FEM parallelization and the algorithms work well in different parallel architectures, but at present there has not been much success in the parallelization of mesh generation methods. This paper will present a massive parallelization scheme for re-meshing with tetrahedral elements using the local modification algorithm. This method is frequently used to improve the quality of elements once the mesh has been generated, but we will show it can also be applied as a regeneration process, starting with the distorted and invalid mesh of the previous step. The parallelization is carried out using OpenCL and OpenMP in order to test the method in a multiple CPU architecture and also in Graphics Processing Units (GPUs). Finally we present the speedup and quality results obtained in meshes with hundreds of thousands of elements and different parallel APIs.

1. Introduction

FEM is the best method to solve large fluid dynamic problems. This method requires the construction of a mesh to describe the geometry of the domain, which is a real bottleneck in the simulation process (one of the main specialists of the area recently opened his conference saying "No mesh, no computations"). The problem gets worse when the domain changes during the simulation process. When the nodes are moved, elements will usually distort to the point that the mesh is no longer valid, and then regeneration is mandatory.

The most accepted strategy is the re-meshing of the whole domain, as proposed by [21,5], among others. This is because the generation using the Delaunay method is faster than one simulation step. Nevertheless, this method requires an a posteriori element quality improvement to remove slivers and other local problems, and this process can be really expensive. Moreover, if the simulation is carried out in a parallel architecture, the re-meshing must also be parallelized, and this is not easy for the Delaunay method.

Our proposal is the use of a robust algorithm for element quality improvement as a re-meshing process, considering that a mesh is already available. The time step of the simulation determines the node displacement and, therefore, the quality of the elements. When the time steps are relatively small (as normally occurs), only a few elements are not acceptable. In these cases, the standard approach of re-meshing the entire domain seems unreasonable, and strategies as local meshers sound more attractive.

There are several approaches to improve the element quality in a given mesh. For example, in [22] the authors propose finding the most suitable position of the nodes based in an optimization process, but in practice this algorithm does not solve most of the problems. The strategy that seems to work really well is the one proposed in [3] and applied by other authors like [4,10]. This algorithm is based on a local topological modification process, changing nodes connection in small clusters of elements.

* Corresponding author at: CONICET - National Scientific and Technical Research Council, Argentina.
E-mail addresses: juan.damato@gmail.com, jpdamato@exa.unicen.edu.ar (J.P. D'Amato).

0743-7315/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jpdc.2013.03.007

In this paper, we propose a dynamic mesher using local topological operations that attempt to optimize a given quality criterion. Our re-meshing process is configurable, since the type of operation can be chosen (with/without nodes movement, nodes insertion/deletion, surface improvement, among others) according to the problem addressed. An interesting feature of this process is that operations can be executed simultaneously on multiple processors in a context of global memory, even in GPUs. Under such circumstances, we may obtain interesting speedups, which compete in time with classical algorithms like Delaunay; but the qualities obtained are significantly better.

2. Background

Simulation accuracy and the time step that must be used depend on the size and quality of the elements. If these elements are degraded by a deformation, a re-meshing strategy is mandatory for recovering a minimum quality. There is extensive literature on how to improve the quality of the meshes in adaptive contexts. In [1,14] a survey of quality optimization methods with dynamic meshes can be found.

Probably the best technique for elements quality optimization in three dimensions is the one proposed by T. Coupez in [3], and adopted in other works like [7,19]. The idea is to analyze small clusters of elements, changing the element connectivity and checking if a better quality can be obtained. The ways in which the clusters are selected and the type of connectivity changes applied have a strong impact in the efficiency of the method.

This technique can be employed even on discretization with large differences in element size, thus making the strategy suitable for use in an adaptive context. In more recent works, such as [20], authors have intended to apply a transformation that takes the distorted space to the original space, letting nodes move freely and elements be reconfigured to adapt to the quality criterion proposed. In [11] a mesh adaptation algorithm is proposed as an iterative combination of point insertions, edge collapses, swaps and point of edge smoothing for Aerodynamics problems. The work of [15] proposes a parallel method but taking into account only node insertion (densification).

For large problems where the number of nodes is very high, the Finite Element Method is usually parallelized and the re-meshing step becomes a bottleneck and must also be programmed in a parallel architecture. Many of the algorithms presented from [16] to [17], for working with dense meshes in a parallelizable environment propose the generation of partitions known as Patches with a minimum connectivity between elements using principles of graphs [12]. These methods treat the surfaces shared by two or more patches differently, which adds a significant synchronization time.

In this work, we will present a variant of the dynamic re-meshing method that runs on parallel architectures. The meshes are previously prepared, distributing the elements in multiple threads from the connectivity information. This method works with all the elements at the same time, thus avoiding the use of lock strategies—similar to lock-free algorithms applied in real-time systems [6]- and ensuring the simultaneous access to data. The strategy is primarily designed to run on PCs with a large amount of processing units, and even on GPUs.

3. Quality optimization

The results of the simulations depend on the size and quality of the mesh elements. If the quality of these elements is near to or below 0, the results are meaningless. If, at the same time, nodes move during the simulation, the mesh quality is constantly degraded. It is in this context that maintaining a high quality of the tetrahedra becomes important; and this is achieved by re-meshing, for each time step, those potentially problematic sections of domain.

Traditional methods (such as Delaunay [8] or Advancing Front [9]) often fail in accomplishing this purpose because of two main reasons: first, they regenerate the entire domain at a very high computational cost, and second, the qualities of the meshes are sometimes much lower than expected. Local change strategies resolve these issues, but sometimes they require several iterations (and time) to provide reasonable meshes and fail to solve inverted elements.

We propose supplementing the traditional strategies with one of local improvements, which modifies a mesh in an attempt to optimize a given quality metric at each time step of the simulation. In order to measure the quality of the elements, we use the proposal of [13], which relates their volume and side length. This metric is suitable and efficient for regular elements; it also tends to improve the dihedral angles. As a global criterion for optimization, the worst dihedral angle is used. This angle defines the numerical accuracy as well as the time step that can be applied. In this section, we will provide the details for the framework we propose.

3.1. Iterative improvement

The proposed improvement algorithm consists of an incremental method. It begins by choosing a set of tetrahedra to form a cluster, and then, it performs a series of internal transformations to generate new tetrahedra. If the lowest quality of the new cluster is greater than the quality of the previous one, the elements are accepted; otherwise, they are discarded. The algorithm then continues with another group of elements. During the evaluation process, the original surface of each cluster should be preserved.

The process is repeated on all the selected elements of the mesh either until a minimum quality requirement is met or when the amount of changes is not significant. This strategy of partial improvement ensures that, in each successive change, the quality of the mesh will be better and will never worsen the worst quality. One of the disadvantages of this method is its computational cost, since each step requires that almost every element of the mesh is evaluated several times.

3.2. Cluster reconnections

For a dynamic mesher to obtain good quality results, it should prove a significant amount of local topological transformations to repair poor quality tetrahedra. The topological transformations are a set of changes that modify the connections, removing existing elements and replacing them by others which occupy the same space. Many previous works [20,2] suggest a limited set of changes, such as 2–3 or 4–4 edge swapping, vertex removal or face collapse.

In this paper, we propose a more general scheme, in which possible connections are freely evaluated within a cluster limited by its outer surface. Following some ideas originally proposed by [3], the algorithm selects a set of tetrahedra, extracts the surface and evaluates connecting all the surface triangles to every opposite node, thus generating a new configuration. If this configuration optimizes the proposed criteria, the result is stored; otherwise, it is discarded. This method, called CLUSTER OPTIMIZATION, is presented below. If, during the simulation, nodes can be added, the method is also tested with a new central vertex.

```
CLUSTER OPTIMIZATION (Cluster C , Mesh M, bool canInsert)
$C_S$ ← extract surface from $C$
$T$ ← triangles in $C_S$
$V$ ←vertexes in $C_S$
$C_{Opt}$ ← copy(C)
```

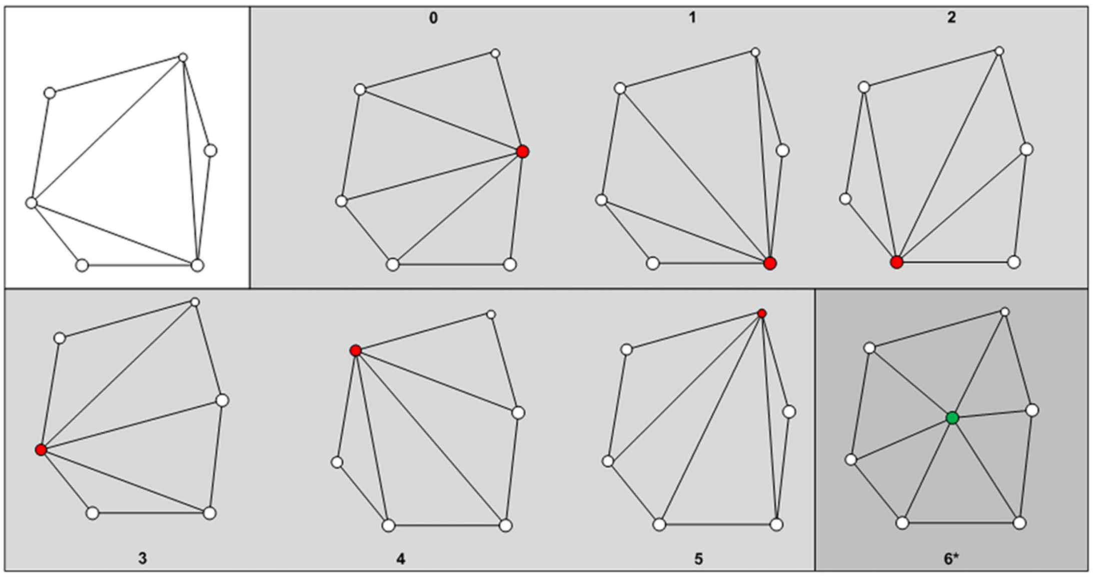


Fig. 1. Cases of clusters configurations.

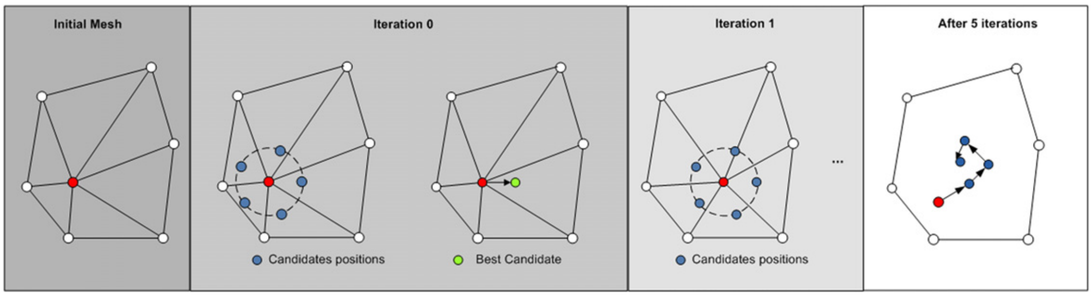


Fig. 2. Best-position search.

```
if (canInsert) V.Add( $C_{center}$ )
for each vertex $v \in V$
        $C_{Temp}$ = new Cluster()
        for each triangle $t \in T$
                tt = new Tetra(t.v0 , t.v1, t.v2, v)
                $C_{Temp}$.Add(tt)
        If $worstQuality(C_{Temp}) > worstQuality(C_{Opt})$
          $C_{Opt} \leftarrow C_{Temp}$
newElements.Add($C_{Opt}$)
removeElements.Add(C)
```

In order to illustrate how this works, a 2D case is shown, where contour is fixed and all the vertexes are evaluated as shown in Fig. 1. The best configuration (case 1) is the one that maximizes the minimum quality with the existing elements. If nodes can be added (case 6), the worst quality and the average cluster are improved even more, at the expense of adding nodes.

Although we cannot guarantee that all the possible connections are analyzed, we can assert that the number of cases evaluated is much higher than those that have been presented up to now.

3.3. *Optimization based on node movement*

Smoothing is a well-known technique for improving the quality of meshes in an efficient manner. Even tough the connectivity of the elements is preserved, the movement of nodes in the simulation may require that certain data are interpolated; therefore, smoothing is not always applied. In this work, several smoothing strategies were evaluated, such as local Laplacian smoothing [22] and iterative Taubin filter [18], which usually provide good results. Although they tend to improve the average quality, sometimes they introduce inverted elements.

To ensure that no negative elements are generated and that quality is constantly improved, we propose another strategy: the idea of this algorithm is to analyze, in a spatial section around a selected node, a set of "candidate positions" that the node may take. These positions are located at an r distance from the node. If one of these positions improves the quality criteria, it is chosen as the new node's position and the process is repeated. If there is no position that optimizes the quality, the radius is updated as $r \leftarrow r/2$, new candidates are proposed and it is tested again until r is less than a certain *tolerance*. The initial r is taken as half the average length of the cluster edges. Fig. 2 shows how the algorithm works.

This strategy requires, at least, 10 iterations to converge, with a rather high computational cost; however, it ensures better results than the previously mentioned smoothing techniques. The parallel pseudo-code is later explained.

3.4. *Surface operations*

The above mentioned operations work exclusively with tetrahedral elements, while the surface that encloses the mesh is preserved throughout the entire optimization process. In many instances, when the mesh is animated or deformed, the surface of the mesh may worsen, overlapping elements that make tetrahedra

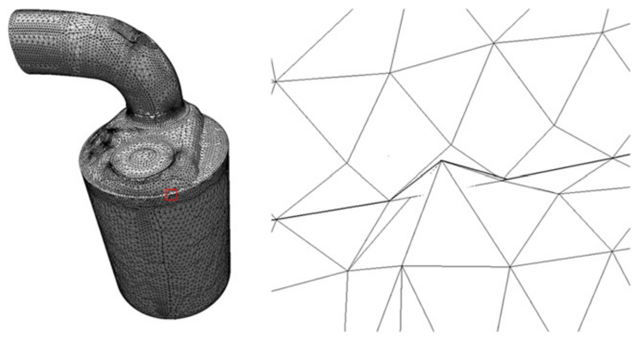


Fig. 3. Cylinder mesh with elements overlapping on the surface.

impossible to be repaired. In such cases, our proposal is to implement operations that improve the surface, such as edge swapping or vertexes removal that optimize the volume criteria. Since such operations might affect the appearance of the mesh, changes must be restricted [7].

Fig. 3 shows a case where these operators should be applied:

4. Operations pipeline

In general, a dynamic mesher like the one we propose here should re-mesh conservatively in order to limit artificial diffusion. Our meshing operations are applied in the order that affects the topology. First, the most restrictive operations are applied (with face or edge reconfiguration), and then, the most aggressive (like smoothing) follow. All operations should always work fast. In order to build clusters efficiently, some connectivity information is needed. We propose that the list of incident tetrahedra, or *elements neighbors*, is stored for each node. This structure is useful to calculate, in a dynamic manner, other types of neighborhoods, such as tetrahedron face neighbors, neighbors by edge and vertex neighbors. These structures are generated at the time of use and are afterwards disposed. As a result, the memory space is effectively reduced. This process is much more robust in parallel processes at an increased computational time.

When some elements are added or removed, the node structure of the elements' neighbors should be instantly updated to ensure mesh conformity. In turn, new and deleted elements should be reflected in the mesh. As shown in the CLUSTER OPTIMIZATION method, two lists of elements are kept: one for the new elements and another for the deleted ones. The mesh is updated only once in a bulk task called CONFORM, reducing memory allocations and accelerating the updating stage.

```
EDGE OPTIMIZATION(Mesh M)
$E \leftarrow$ set of all edges of tetrahedra $\in M$
for each edge $e \in E$
    $C \leftarrow$ Cluster($e$) { Subset of tetrahedra in mesh M with edge e}
    CLUSTER OPTIMIZATION( $C$, $M$ )
Conform(M)
FACE OPTIMIZATION(Mesh M)
$F \leftarrow$ set of all faces of tetrahedra $\in M$
for each face $f \in F$
    $C \leftarrow$ Cluster($f$) {Subset of the tetrahedra in the mesh
                    M that has face f}
    CLUSTER OPTIMIZATION( $C$, $M$ )
Conform(M)
NODE OPTIMIZATION(Mesh M)
$V \leftarrow$ set of all vertexes of tetrahedra $\in M$
for each vertex $v \in V$
    $C \leftarrow$ Cluster(v ) {Subset of the tetrahedra in the mesh
                    M that has vertex v}
    CLUSTER OPTIMIZATION( $C$, $M$ )
Conform(M)
SURFACE OPTIMIZATION(Mesh M)
$E_S \leftarrow$ set of all surface edges of tetrahedra $\in M$
for each edge $e_s \in E_S$
    $min_q \leftarrow$ min( quality($e_s.T1$, $e_s.T2$)
    if swapDiagonal($e_s.T1$, $e_s.T2$) $> min_q$
        CLUSTER OPTIMIZATION( $C$ )
Conform(M)
DYNAMIC IMPROVE MESH(Mesh M , int flags)
for each iteration
    EDGE OPTIMIZATION(M)
    FACE OPTIMIZATION(M)
    If (NodeRemovalFlag) NODE OPTIMIZATION(M)
    If (SurfaceFlag) SURFACE OPTIMIZATION(M)
    If (SmoothFlag) SMOOTH OPTIMIZATION(M)
```

For the tested cases, the quality of the mesh drastically increases in the first three iterations.

5. Parallel meshing

The dynamic meshing method has the advantage that the elements are analyzed in localized groups. As noted in the previous section, each time new connections are generated, neighborhood structures must be updated. These updates in a concurrent access environment should be treated by a synchronization mechanism, such as barriers or semaphores, to avoid inconsistencies. But, when there are many processes accessing the same data, these synchronization mechanisms slow down the overall process. However, it can be predicted that accesses to and changes of two clusters that do not share elements can be treated simultaneously without data locks.

We propose a set of definitions that help generalize the problem. Each of the operations listed in the previous section is a mesh operator known as O. Every operator O works on a cluster of elements C. Depending on the operator type, each cluster is conformed around an element E using neighborhood information (either per vertex, face or edge); this is what we call the Operator Scope or S_O.

The aim of this process is to ensure that there are multiple instances of processing operators, since clusters should not share items. We define *mutual-independence* between two elements E and E' as $I(E, E') = 1$ if it meets $S_O(E) \cap S_O(E') = \emptyset$.

If we wish to process multiple clusters simultaneously using parallel architectures, we need to organize them so that they ensure the condition of mutual independence. To satisfy this condition, it is necessary to make a processing scheduling. The scheduling algorithm will be explained in the following section.

5.1. Assignment method

Since it is expected that all elements can be modified by several independent threads simultaneously, there must be a distribution of elements among these processors that ensures the consistency of the changes. Considering the element neighborhood and applying a temporary order, changes affect only one element at a time.

Using the previously defined notions of Operators Scope, and taking one element E, all elements within the scope of E are marked as visited and they are partly excluded from the analysis. If another independent element called E' is found, it can be assigned to a different thread. In case there are many other elements to process, the same criteria is applied and the available threads are assigned. The elements are sorted by quality in order to ensure that the worst elements are processed first.

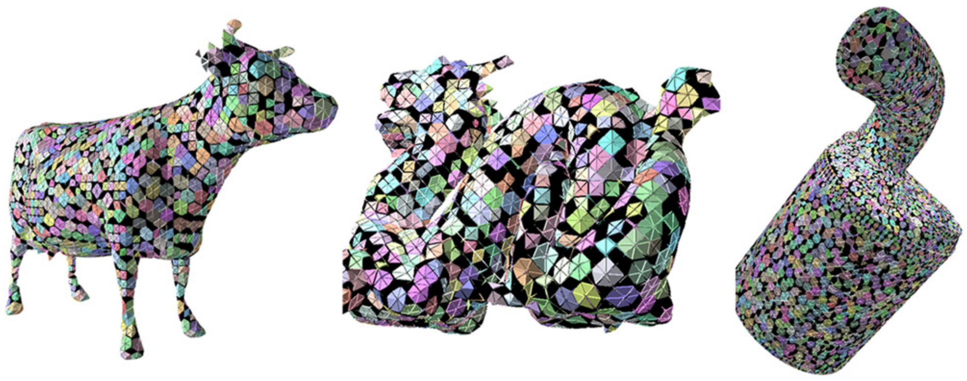


Fig. 4. Independent clusters identified with different colors. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

The proposed algorithm has the following form:

```
ASSIGMENT (Elements E , Operation O)
ResultList R
$E$ ← sortByQuality(E)
for each $e \in E$
        $S_C$ ← Scope (e , O)
        If marked( e ) or markedAny ($S_C$) {If already visited }
        continue
        for each $e' \in S_C$
                mark($e'$)
        Push(R, e)
return R
```

Elements distributed in different mutually independent clusters are shown in Fig. 4. These clusters can be seen as small partitions of the mesh.

When comparing parallel execution to the sequential version, certain unresolved issues arise. The most important one is that all elements that have been discarded for not complying with the condition of independence should later be processed. In this paper, we propose that all the elements that have not been allocated in the first iteration, are allocated in the next stage and the process is repeated until all the elements are visited. On the other hand, if a cluster has been improved, new elements are created, which might need to be evaluated, while those deleted need to be discarded. The new and deleted elements are maintained into two different lists for each cluster until every cluster has been processed. After the processing stage, all memory news/deletes are performed in the CONFORM method and the visit list is updated.

Now, the process proposed to optimize by NODE presents the following form:

```
PARALLEL NODE OPTIMIZATION (Mesh M)
$V$ ← set of all vertexes of tetrahedra $\in M$
While not empty(V)
      $V'$ ← assignment(M, NodeOperation)
      { $V'$ is a list of vertexes }
      Remove($V$, $V'$)
      Parallel for each vertex v $\in V$
                C ← Cluster(v) {Subset of tetrahedra
                                that has vertex v}
                CLUSTER OPTIMIZATION( C )
CONFORM(M)
return
```

This process is carried out with the different types of operations proposed, where only the Scope function and the elements to be visited (nodes, faces, surface faces or edges) vary. In order to include a new type of operator that has not been described here, it is necessary to define the appropriate Scope function. Although we have shown a working scheme where only one type of operator is applied at a time – similar to a SIMD (Single Instruction Multiple Data) architecture – different types of operators (similar to MIMD) could coexist.

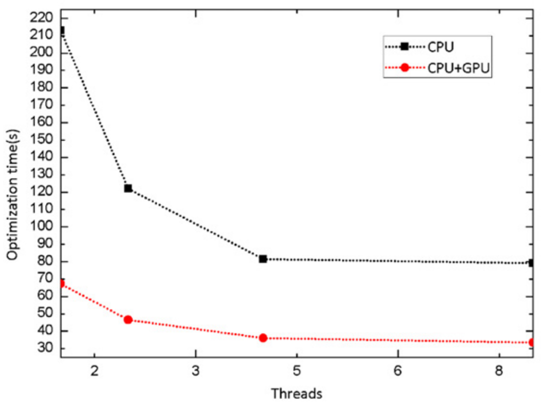
The total mesh optimization time is composed by an *assignmentTime* plus *optimizationTime*. The *optimizationTime* is fully parallelizable, so it is expected to decrease linearly when running on more processors. Since the assignment has a linear cost, it is time-invariant in different hardware architectures; but its simplicity ensures that the execution time is several times less than the evaluation time.

5.2. Smoothing on the GPU

The use of the GPU as a processing unit operates efficiently in parallel for numeric calculation, but it has serious limitations when dynamic memory managing is required. In this project, we suggest the use of OpenCL, which has not yet integrated functions such as new or delete data that CUDA already provides. Under these conditions, the structures updates are not trivial and the accelerations with respect to a multi-core CPU implementation are not as good as expected.

On the other hand, the proposed smoothing strategy intensively evaluates the positions of the nodes without changing their connectivity and promises to be the most favored function. An implementation of this method on a GPU is easily carried out using OpenCL with the advantage of portability to multiple CPUs or GPUs architectures. Since a node movement affects the quality of incident elements, the independence condition should be verified, for which the assignment scheme is previously applied. The parallelization of the reconnectivity step is not so easy, and may be an architecture dependent implementation will be necessary.

```
PARALLEL SMOOTH KERNEL (Vertex V , float radius ,
                        Tetra* tetraNeighbors)
$C_{OPT}$ ← $tetraNeighbours[V]$ Build a Cluster
                        with neighbors
$q_{best}$ ← $getClusterQuality(C_{OPT})$
$radius$ ← $meanEdgeLength(C_{OPT})$
While radius $> \epsilon$
      $V_{Candidates}$ ← computeCandidates(V, radius)
      for each vertex v $\in V_{Candidates}$
          $V_{POS}$ ← $v_{pos}$
          $q_{temp}$ ← $getClusterQuality(C_{OPT})$
          if ($q_{temp} > q_{best}$ )
             $v_{best}$ ← $v_{pos}$
             $found$ ← $true$
      if (not $found$ ) $r$ ← $r/2$
      $V_{POS}$ ← $v_{best}$
return
```



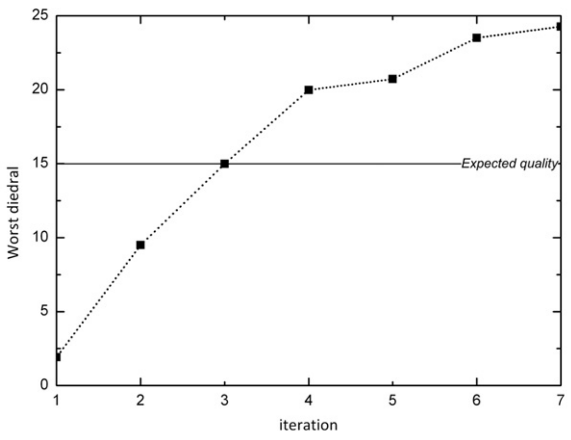


Fig. 5. (Left) Optimization time per thread. (Right) Amount of elements improved per iteration.

```
PARALLEL SMOOTH OPTIMIZATION( Mesh M , float radius)
$V$ ← set of all vertexes of tetrahedra $\in M$
$t_N$ ← tetrahedra vertex neighborhood$\in M$
CopyDataToGPU ($V$, $M$, $t_N$)
While not empty(V)
    $V'$ ← assignment(M, NodeOperation) {$V'$ is a list of vertexes}
    Parallel for each vertex v $\in V'$
        PARALLEL SMOOTH KERNEL (v , radius, $t_N$ )
ReadDataFromGPU($V$)
return
```

The copying times from GPU to CPU memory, which are supposed to be insignificant, are considered in the optimization time.

6. Benchmarks

The efficiency of the optimization strategy presented above is tested with a simple optimization case. A cubic cavity is considered, which is defined in the range (0, 0, 0) to (1, 1, 1), refined with 6×10^5 tetrahedral elements with an initial quality of 7.26°. This cavity is deformed by certain function, which decreases the quality to 0.05° and produces the appearance of 48 inverted elements with negative volume. In order to optimize the quality of this mesh, 7 iterations of parallel processing are applied using all the steps of volume (EDGE, FACE, NODE, SMOOTH). SURFACE OPTIMIZATION is omitted since the mesh surface is preserved. As the SMOOTH step is executed in the GPU, times are taken separately. We expect to obtain a minimal quality of 15°.

The first calculations were performed on an AMD 6-cores at 2.6 GHz with an NVIDIA GTX550 graphics card. The parallelization was carried out with OpenMP and OpenCL. The time required to run the process using an increasing number of processors and adding the GPU to the calculation is shown in Table 1 and it is presented in a graphic in Fig. 5. The results show that the optimization algorithm exhibited good performance in parallel, but it is not the best. Also a good speed up is obtained when the smoothing step is carried out on a GPU. The problem is now in the cluster assignment process and the actualization of data structures necessary when the nodes connection changes. These stages of the algorithm remain sequential and are the new bottleneck. Both processes can also be parallelized and they are a topic for future work.

The mesh after seven iterations has a quality of 25.10° but three were enough to eliminate all inverted elements and get 15° as the minimal dihedral.

Table 1
Speedups obtained. (Left) The whole process in the CPU. (Right) Smoothing on the GPU and re-mesh in the CPU.

Optimization times						
#Cores	CPU time (s)			CPU + GPU time (s)		
	Smooth (CPU)	E + F + N (CPU)	Total	Smooth (GPU)	E + F + N (CPU)	Total
1	300.17	81.77	381.94	10.73	80.35	91.08
2	168.23	51.61	219.84	9.74	50.89	60.73
4	97.49	37.42	134.91	9.80	37.86	47.60
6	77.53	34.22	111.75	10.09	34.59	44.39

Table 2
Optimization times compared to its sequential version.

Large meshes test					
Mesh	#Elements	Conf1		Conf2	
		Time (ms)	Speedup	Time (ms)	Speedup
Dragon	32 959	2 169	3.80×	3 363	3.42×
Sculp	50 391	3 179	4.22×	4 907	3.84×
Staypuft	102 392	6 459	3.71×	11 896	2.81×
Cylinder	378 018	21 341	5.67×	33 799	5.10×
Cavity	607 297	32 293	6.61×	48 248	6.36×
F1	2 080 457	105 437	4.75×	191 562	3.70×

6.1. Large mesh optimization

It is important to test the algorithm in meshes with a great amount of elements. The second benchmark tries to optimize the quality of six examples (some of which have already been used in other works), with a smaller mesh of 3×10^4 elements and a larger one of 2×10^6 elements. Fig. 6 shows some of the examples used.

Two different PC configurations were used with different parallelization APIs. One is an Intel i-5 (2nd generation) with 4 cores at 3.3 GHz with 8 GB of RAM and a NVIDIA GTX 560 with 1.5 GB of RAM using Intel Thread Building Block (TBB) called *configuration*1; the other is an AMD 6-cores at 2.6 GHz with an NVIDIA GTX550 with 1 GB of RAM running with OpenMP called *configuration*2. We used in both OpenCL for GPU step. Table 2 shows the best times obtained in each case after three iterations and the improvement of the worst dihedral angle with respect to the incoming quality. Fig. 7 summarizes in images the qualities obtained for the cases proposed.

The obtained qualities were the same for both PC configurations. With *configuration*1, we obtained very good speedups, even with less cores, thanks to a newer technology and a better GPU. At the same time, *configuration*2 was limited by GPU memory size; the biggest mesh that could be allocated was the $F1$ one. It was also observed that the larger the mesh, the better the speedup, as more

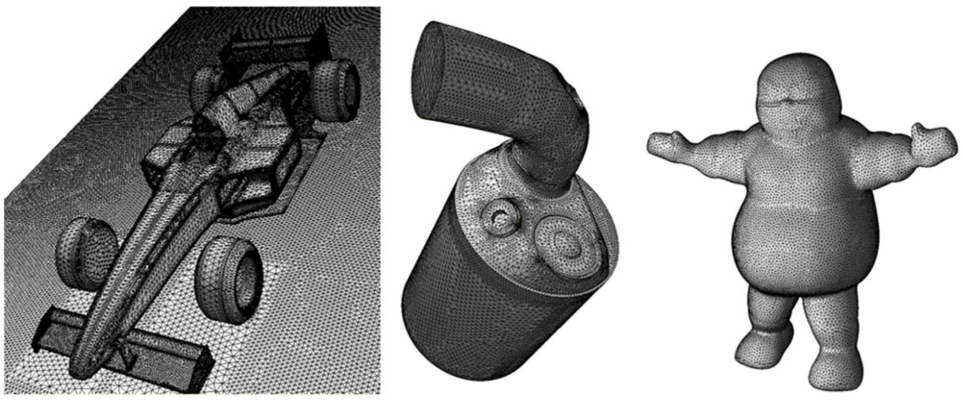


Fig. 6. Large meshes tested.

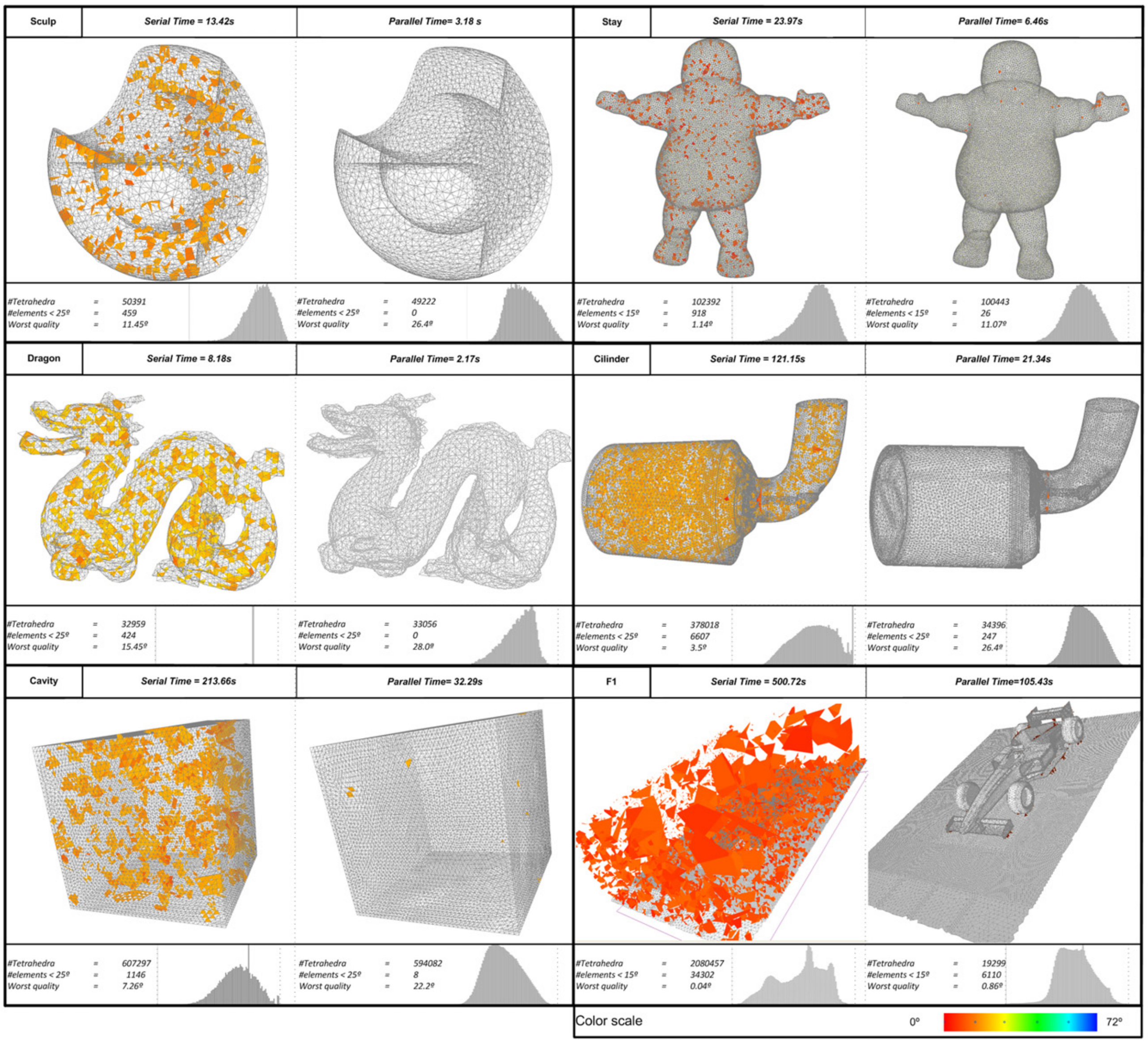


Fig. 7. Six meshes before and after improvement. In each box, the left mesh is the input, the right mesh is the optimized. Histograms show the distributions of dihedral angles and the minimum and maximum dihedral angles in each mesh. Colored elements are tetrahedrals that have a dihedral angle below 15°. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

clusters are processing in parallel. On the other hand, poor quality meshes (*F*1 and *staypuft*) take down this rate because clusters have more elements. Top speedups of 6.61× were obtained for the *cavity* mesh which has a rather good initial quality.

Table 3
Comparing times and quality with other re-meshing library.

Large meshes test						
Mesh	Worst dihedral		Average dihedral		Time (s)	Our method
	(Stellar)	(Our method)	(Stellar)	(Our method)	(Stellar)	Speedup
Dragon	27.11°	28.11°	51°	54°	28.0	12.9×
Sculp	30.05°	26.40°	45°	55°	110.0	30.7×
Staypuft	18.10°	11.07°	50°	60°	198.0	34.6×

Finally, an open access re-meshing library, Stellar [20], was used to compare execution times and element quality. In order to get comparable results, Stellar and our proposal were limited to a fixed amount of improvement iterations. We evaluated the cases, already published in their work, measuring minimum and mean dihedral angles. In Table 3, we show the obtained results.

In general, Stellar obtains better minimal angle qualities, but worse average angles. The speedups were not linear, since the operations pipeline and termination criterion are different. It can also be observed that the time required for a large mesh such as *Staypuft* is too high (more than 3 min) and it is not applicable in an adaptive context.

7. Conclusions

This paper presents a meshing algorithm designed to be used in a highly parallelizable environment. The implementation of the solution proposed here is relatively simple, since it is based on loop parallelization functions that are provided by open libraries such as OpenMP or TBB.

The procedure presented has three main components. The first is a quality metric which determines the elements that should be processed. The second is a set of optimal regeneration strategies of elements within a local environment from the metric chosen. The third is an administrator of these clusters which allows simultaneous modification of multiple groups.

The proposed framework is flexible in the sense that any of these components, especially the operators, can be modified or extended. Meeting the proposed scope definitions, the method ensures that the new components can be effectively parallelized. This scheme has already been successfully applied to surface meshes composed of triangles, and it may be extended to other types of elements. To improve performance even more, the method can be applied only to clusters that have poor quality (below a given tolerance).

This study only showed a set of possible configurations of the elements, which allowed us to achieve significant improvements in quality, but did not guarantee that all possible reconnections leading to a global optimum were evaluated. In a future line of work, we should extend the reconnection method so as to consider more possibilities. The other possible line of work consists of using this strategy in a distributed context, with even larger meshes, for which a message passing scheme might need to be added.

References

[1] C.J. Budd, W. Huang, R.D. Russell, Adaptivity with moving grids, Acta Numerical 18 (2009). http://dx.doi.org/10.1017/S0962492906400015.

[2] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, W. Wright, Simplification envelopes, in: ACM SIGGRAPH 93 Conference Proceedings, New Orleans, Louisiana, 1996, pp. 119–128.

[3] T. Coupez, A mesh improvement method for 3D automatic re-meshing, in: 4th International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, 1994, pp. 615–626.

[4] T. Coupez, H. Digonnet, R. Ducloux, Parallel meshing and re-meshing, Applied Mathematical Modelling 25 (2) (2000) 153–175.

[5] F. Dehne, C. Langis, G. Roth, Mesh simplification in parallel, in: Proceedings 4th International Conference on Algorithms and Architectures for Parallel Processing, Hong Kong, 2000, pp. 281–290.

[6] F. Fich, D. Hendler, N. Shavit, On the inherent weakness of conditional synchronization primitives, in: 23rd Annual ACM Symposium on Principles of Distributed Computing, 2004, pp. 80–87.

[7] P. Frey, H. Borouchaki, Geometric surface mesh optimization, Computing and Visualization in Science (1998) 113–121.

[8] B. Hudson, G. Miller, T. Phillips, Sparse parallel Delaunay mesh refinement, in: ACM Symposium on Parallelism in Algorithms and Architectures, California, USA, 2007.

[9] Y. Ito, A. Shih, A. Erukala, B. Soni, A. Chernikov, N. Chrisochoides, N. Nakahashi, Parallel unstructured mesh generation by an advancing front method, Mathematics and Computers in Simulation 75 (2007) 200–209.

[10] B.M. Klingner, Tetrahedral mesh improvement, Ph.D. Thesis, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, California, 2009.

[11] A. Loseille, R. Lohner, Anisotropic adaptive simulations in aerodynamics, AIAA 10-0169, 2010.

[12] I. Moulitsas, Graph partitioning for scientific computing applications, Seminar in Department of Applied Mathematics, University of Crete, Heraklion, 2007.

[13] V.N. Parthasarathy, C.M. Graiche, A.F. Athaway, A comparison of tetrahedron quality measures, Finite Elements in Analysis and Design 15 (3) (1994) 255–261.

[14] G. Rokos, ISO Thesis: Study of Anisotropic Mesh Adaptivity and its Parallel Execution, Imperial College, London, 2010.

[15] R. Rossi, J. Cotela, N. Lafontaine, P. Dadvand, S.C. Idelsohn, Parallel adaptive mesh refinement for incompressible flow problems, Computer & Fluids (2012) http://dx.doi.org/10.1016/j.compfluid.2012.01.023. (in press).

[16] K. Schloegel, G. Karypis, V. Kumar, Multilevel diffusion schemes for repartitioning of adaptive meshes, Journal of Parallel and Distributed Computing 47 (2) (1997) 109–124.

[17] J. Southern, G.J. Gorman, M.D. Piggott, P.E. Farrell, Parallel anisotropic mesh adaptivity with dynamic load balancing for cardiac electrophysiology, Journal of Computational Science 3 (2012) 8–16.

[18] G. Taubin, Estimating the tensor of curvature of a surface from a polyhedral approximation, in: Proc. of Int. Conf. on Computer Vision, 1995, pp. 902–907.

[19] D. Wang, O. Hassan, K. Morgan, N. Weatherill, EQSM: an efficient high quality surface grid generation method based on re-meshing, Computer Methods in Applied Mechanics and Engineering 195 (2006) 5621–5633.

[20] M. Wicke, D. Bryan, M. Klingner, S. Burke, M. Klingner, J. Shewchuk, J. O'Brien, Dynamic Local Re-Meshing for Elastoplastic Simulation, Computer Graphics Proceedings, in: Annual Conference Series, vol. 49, 2010, pp. 1–12.

[21] C. Wojtan, G. Turk, Fast viscoelastic behavior with thin features, ACM Transactions on Graphics 27 (2008) 41–47.

[22] Y. Zhang, C. Bajaj, G. Xu, Surface smoothing and quality improvement of quadrilateral/hexahedral meshes with geometric flow, in: Proc. 13th Int. Meshing Roundtable, 2005.

Juan Pablo D'Amato is a Systems Engineer since 2004 and a Ph.D. from UNCPBA University (Tandil, Argentina) since 2011. He is an assistant professor in Computer Graphics courses and has worked in the development of real time systems and simulators for the Argentine Armed Forces. His main research interests include geometry, quality metrics, high performance, visualization and virtual reality.

Marcelo Vénere is a Ph.D. in Nuclear Engineering from Balseiro Institute, Argentina. He is an Associate Professor in Computer Graphics and Algorithms courses at the UNICEN University (Tandil, Argentina). He is leading several projects in arterial fluid simulation, real time training and computer graphics; among other topics. He is the Co-Director of the PLADEMA Research Institute of the UNCPBA University at Tandil, Argentina.