# 5A.2

# Large Out-of-Core Tetrahedral Meshing

Aurélien Alleaume, Laurent Francez, Mark Loriot, and Nathan Maman

Distene SAS, Pôle Teratec – Bard-1, Domaine du Grand Rué,
91680 Bruyères-le-Chatel, FRANCE
{aurelien.alleaume,laurent.francez,mark.loriot,nathan.maman}@distene.com

**Summary.** We present an effective strongly uncoupled method, which, given a closed and watertight surface mesh, generates its Delaunay constrained tetrahedrisation into sub-domains. Uncoupled means here that once chosen, a sub-domain interface will not be changed anymore. This method aims at preserving in the final tetrahedral mesh the properties of the mesh which would have been produced by the original sequential algorithm, namely the constrained input surface mesh and the quality and properties of the volume mesh. To achieve these objectives, our method reorders internal vertices insertion (applying the sequential constrained Delaunay kernel) such that the data can then be fully decoupled in sub-domains without introducing more constraints. Moreover, the interfaces are carefully chosen such that the load is roughly the same in all parts, and such that the interfaces separating the sub-domains are "invisible" in the final mesh. Successfully applied to out-of-core large scale mesh generation, this method allowed us to generate in double precision a 210 million tet mesh on a 4GB core memory single processor machine, and a 680 million tet mesh for a very complex geometry.

## Introduction

In order to represent accurately complex geometries and to capture fine physical phenomena, high fidelity computational simulations involve more and more quality and density constraints on meshes. Generating large and high quality meshes can be achieved using a parallel approach, in order to gain performance in both memory consumption and CPU time.

A fast and robust sequential mesher like [7] has typical scalable speeds of about 5 million cells/minute on industrial cases in sequential mode. With such a mesher, the size of a mesh is not a problem as long as it fits into core memory. This tends to move the main focus to core memory limitation rather than pure meshing speed. Indeed, hardware and manufacturing limitations make non-distributed large core memory machines very expensive therefore justifies research of out-of-core or distributed parallel alternative methods for the generation of very large computational meshes.

Much effort must also be put on getting meshing properties as close as possible to the ones obtained with classical methods, such as density, number of cells, and quality of cells, but also reliability, so as to limit as much as possible the side effects of using these alternative methods.

Our method targets therefore **memory usage distribution** (rather than speed gain, which will eventually be addressed afterwards), a **full sequential code re-use** to ensure maximum reliability and the **same mesh properties** as the sequentially generated mesh.
To reach these targets, the *keys* of our method are :

1. The **internal edges division process reordering**, in order to **localize data** and to achieve **sub-domain decoupling**.
2. Built up of sub-domain interfaces such that they **do not interfere** with further insertions of vertices through the **constrained Delaunay kernel**.

Following the classification of parallel meshing techniques found in [1], the method presented here is a discrete domain decomposition method for Delaunay constrained mesh generation. Larwood et al. ([10]) propose to decompose the input surface mesh by creating interfaces with two-dimensional Delaunay triangulations. To make their method more robust and effective they also reject some cutting regions depending on angle considerations. Said et al. ([12]) decompose the initial coarse mesh by re-meshing an interface made of coarse tetrahedra faces.

What we propose goes beyond these approaches : in order to decompose the initial coarse mesh, we create the sub-domain interfaces through a three-dimensional Delaunay tetrahedrisation. This allows us to avoid imposing new faces into the mesh, by simply "freezing" some tetrahedra faces to create an interface. This generates interfaces which have a minimal impact in the resulting mesh, as if they had been produced by an *a posteriori* mesh partitioning method applied to the assembled mesh.

The method is based on a recursive bisection. An important feature of our approach is that each bisection determines the final interface between the two newly created sub-domains, very much in the way it would have been produced in the original sequential core-memory mesher. In other words, this method addresses the constraints on the sub-domains interfaces only during their construction and not afterwards, thus allowing a highly uncoupled algorithm.

## 1 Distributed mesh generation

### From sequential to distributed mesh generation

Given a boundary surface mesh, we compute its coarse Delaunay constrained volume mesh. A state of the art implementation of the involved algorithms

comes from INRIA [7]. This is the sequential whole-in-core basis we use for our distributed meshing method.

Following a recursive bisection approach, this tetrahedral mesh is then recursively split into two parts until an arbitrary number $ns$ of sub-domains is reached. Each sub-domain is then saturated to its final density using the original whole-in-core sequential algorithm.
The work flow of the method is summarized in figure 1.

The kernel of our method has two main steps:
1. bisection of a sub-domain mesh $S$;
2. topology information management between sub-domains.

To be efficient, the first step involves more or less complex sub-steps while the second mainly requires careful design and implementation.
We will describe them with more details in the following sections. An illustrated example can be found in section (2.1)

### 1.1 Bisection

This bisection step does not target an optimal mesh decomposition. Thus, for sake of implementation simplicity and to assess the interest of the overall method, we chose to cut the mesh around a bisection plane. Our method can however be generalised to other kinds of decomposition (see [2]). The only prerequisits are :

1. The ability to locate a point relative to the cutting surface, i.e. to find on which side it is.
2. The ability to tell if an edge intersects a given width area around the cutting surface or not.

Note that we use here a mesh partitioning approach only in order to compute a bisection plane which will be used for local mesh generation,

#### Choosing a good bisection plane

An easy way to choose the cutting plane is to compute the inertia axis of the sub-domain mesh (see [3]). In other words, for a sub-domain $S$, we compute the inertia matrix

$$M = (\tau_{i,j})_{i,j=1\ldots3} \quad \text{with } \tau_{i,j} = \sum_{x \in G} (x_i - g_i)(x_j - g_j)$$

where $g = (g_i)_{i=1\ldots3}$ is the gravity center of $S$ and $G$ is the set of gravity centers of tetrahedra in $S$.

The inertia axis is given by an eigenvector associated to the highest singular value of $M$[1]. This method is very fast and uses the locality of data, *i.e.*,

[1] It can be simply computed by a successive power method.

the more the domain gets divided, the more the inertia axis captures the mesh topology. More information on this subject can be found in [5].

This inertia axis minimises the sub-domain spread (in some sense) from itself. So to minimise the size of the interface, the cutting plane is chosen orthogonal to it with the following considerations in mind:

1. the number of sub-domains may not be a power of two;
   To be able to generate a final number of sub-domains which is not necessarily a power of two, we partition this sub-domain with a ratio not equal to one half.
2. a good load balance may be critical to get a mesh with the smallest number of sub-domains possible and is almost required for most parallel applications that may use such partition.
   To achieve a good load balancing in the subsequent bisection steps, care should be taken that two coarse tetrahedra at this stage will not hold the same number of tetrahedra in the final mesh. Evaluating the target mesh metric and computing tetrahedral volume in that metric gives an acceptable *a priori* final load evaluation, hence a more accurate load is used to select the cutting plane position. For a tetrahedron $T$ we compute:

   $$\int_T d\lambda = \iiint_T \frac{1}{h^3(x,y,z)}\, dx\, dy\, dz$$

   where $\lambda$ is an approximation of the metric in $S$ and $h(x,y,z)$ approximates the desired size at $x, y, z$.
   This step will really make a difference in terms of performance and first shot distribution on a non uniform mesh[2].

### Refining the mesh around the bisection plane

In this step, vertices are inserted around the cutting plane following a volume constrained Delaunay kernel (see [5]):

Let $\mathcal{T}_i$ be the Delaunay tetrahedrisation of a set of points $(P_k)_{k\in\{1..i\}}$. Then, $\mathcal{T}_{i+1}$ , the result of the insertion of point $P = P_{i+1}$ in $\mathcal{T}_i$ can be written

$$\mathcal{T}_{i+1} = \mathcal{T}_i - \mathcal{C}_P + \mathcal{B}_P$$

where :

- $\mathcal{C}_P$ is the set of elements whose circumsphere contains $P$. In order not to lose faces given in the input surface mesh this set can be slightly restricted (and then corrected) such that no required face is an internal face of $\mathcal{C}_P$.

[2] Our method excludes over-decomposition techniques that often compensate load imbalance consequences by computing more sub-domains than desired and mapping several ones on each processor.

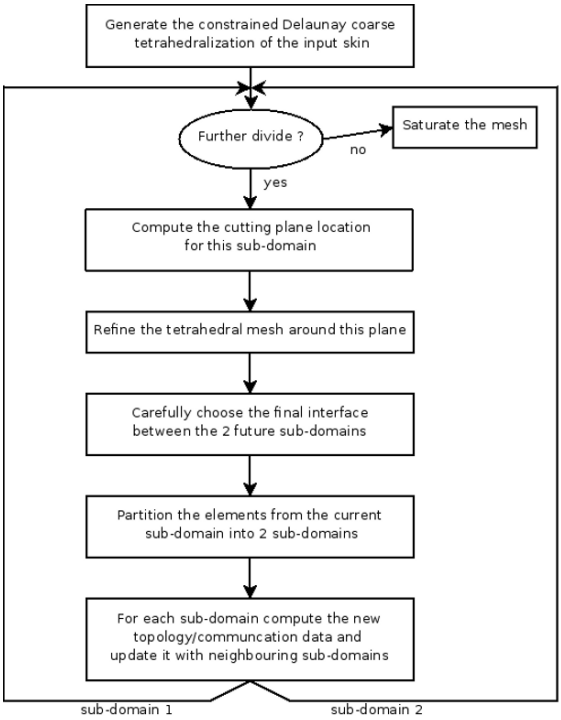



**Fig. 1.** Method work flow

- $\mathcal{B}_P$ is the set of elements formed by joining $P$ with the external faces of the elements in $\mathcal{C}_P$.

To allow graded mesh generation the inserted vertices are computed by dividing the existing coarse edges that intersect a given area around the bisection plane.

Let the refined zone be the area around the cutting plane which is refined to its final vertex saturation.

The vertices insertion is conducted such that the refined zone is composed of a layer at least two tetrahedron wide on each side of the cutting plane. This will leave some degree of freedom in the next step for choosing the interface faces inside this area.

After this point insertion step, the refined zone is almost meshed as it would have been in the fully sequential method. With very large and fine meshes, this unbalanced vertex insertion often triggers precision issues in the constrained Delaunay sequential mesher. This is especially true when it comes to cavity computation and points localization. Some sequential algorithms have been slightly tuned to address these issues better.

**Choosing the interface**

Let the cutting zone be a one-tetrahedron layer wide area on each side of the cutting plane. By construction, the cutting zone is included in the refined zone.

The interface between two sub-domains is formed by selecting faces of tetrahedra from the cutting zone. Since this volume zone was refined to its final saturation in the previous step, the selected surface is already consistent with the overall size and does not have to be re-meshed or refined.

The interface between two sub-domains chosen at this step will remain unchanged until the end of the mesh generation. This choice takes therefore into account the following heuristic criteria:

1. As we target minimal communication between sub-domains, we should minimize the number of faces in the interface.
2. As the quality of each interface element will condition the final quality of the final mesh, we need to retain the best shaped ones.
3. To avoid over constrained tetrahedra (i.e. with several faces in sub-domain interfaces or input boundary surface mesh), the interface should be as flat as possible.

These simple heuristics lead to a multi-criteria optimization problem. We solve it with local optimizations. Working on a per tetrahedron optimization basis quickly shows its limits as it cannot optimize some frequent patterns. If we use instead all tetrahedra around an interface vertex, we are able to achieve better interface smoothing.

This method can lower the interface size by approximately thirty percent while retaining the best shaped faces.

### 1.2 About constrained Delaunay meshing

Let $\mathcal{I}$ be the set of all the faces in the interface build in the previous step and $\mathcal{Z}$ be the set of all the elements from the cutting zone with a face in $\mathcal{I}$.

To ensure that our method generates a constrained Delaunay mesh of the input surface (*i.e.* that interfaces do not introduce more constraints regarding the Delaunay property), a sufficient condition is that for any point $P$ that will be inserted in the future, no interface face is an internal face of the cavity of $P$, $\mathcal{C}_P$.
A slightly stronger but simpler condition is that for any point $P$ that will be inserted in the future :

$$\mathcal{C}_P \cap \mathcal{Z} = \emptyset$$

This is because if an internal face of $\mathcal{C}_P$ is in the interface, then at least one of its adjacent tetrahedra is in $\mathcal{C}_P \cap \mathcal{Z}$.

Since no point will be inserted further in the refined zone, a sufficient condition is that the circumsphere of all the elements in the cutting zone is included in the refined zone.

This condition is not explicitly verified near the input (constrained) surface mesh when the refined zone is built but the practice shows that it is not really needed, as the extra layers of tetrahedra in the refined zone around the cutting zone create an area large enough.

### 1.3 Topology information management

Topology information is kept by each sub-domain to allow the merge of all the sub-domains into a single sequential mesh, or the generation of the distributed local communication information which would be required by a parallel solver.

This step does not involve any specific technique but requires careful implementation to assert its correctness and good performance. Basically, after its bisection, each sub-domain sends the topology update to its neighbours. In parallel mode, this step would become a synchronization point.

### 1.4 Memory consumption and method limits

Memory consumption has two local maxima: the first bisection and the last sub-domain meshing step. The latter can be lowered simply by increasing the desired number of sub-domains whereas the first maximum fixes the limit of the largest achievable mesh on a given machine for this method.

For example, we were able to generate with our method more than 211 million cells in double precision (64 bit) on a 4GB machine (see section 2.3).

## 2 Applications

### 2.1 A Simple illustrated application

The first example illustrates how the method works for a decomposition into two sub-domains. The case is a car embedded entirely into a large volume (courtesy of PSA Group).

We deliberately chose here to put the cutting plane on the front of the car, close to but without intersecting it. This is often a difficulty for distributed mesh generation methods which only use surface mesh information.

The figure 2 shows the initial coarse volume mesh before and after it gets refined around the bisection plane. Then figure 3 shows the resulting sub-domain interface with or without faces choice optimisation. Last, figure 4 compares the sequential mesh with the one generated by our method in the previously described configuration. It shows that our interface construction enables us to generate a mesh which is almost identical to the sequential one in terms of density. Moreover, on this final mesh, one cannot identify where the cutting plane lies (see [11] for the planar case analysis in which the interface lines can be seen in the resulting mesh *i.e.* it does not preserve the likewise-sequential property we maintain here; see also [8] for tetrahedral meshes in which the inital coarse mesh surfaces can be seen).

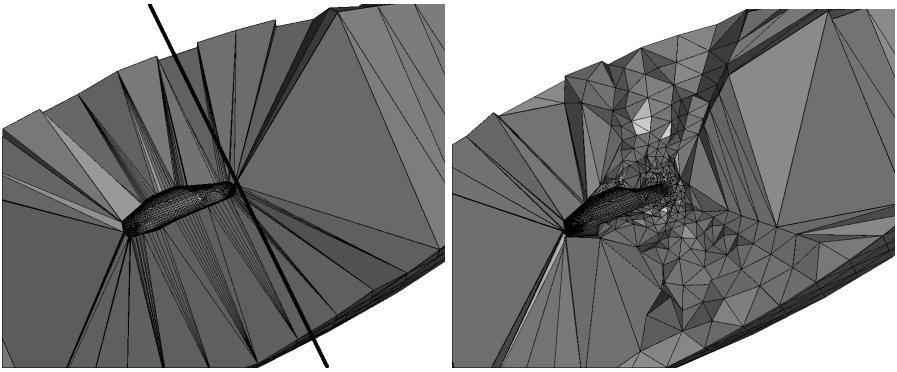


**Fig. 2.** Initial coarse mesh (on the left, the black line gives the location of the cutting plane) and mesh refined around the cutting plane (right). View is a cut orthogonal to the bisection plane.

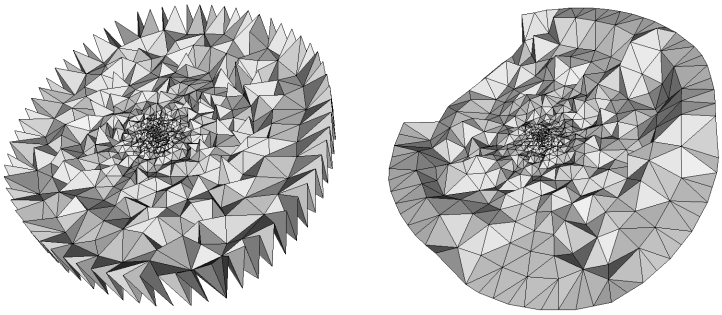


**Fig. 3.** Natural (left) and optimized (right) sub-domain interface

### 2.2 Sequential versus out-of-core meshing comparison

In this section we present a more complete comparison of the sequential mesh and the one generated with a 4 domain decomposition.

For this purpose we consider 5 industrial or non trivial test cases. They are illustrated on figures 5, 6, 7, 8, and 9.

The most interesting one is the one on figure 9 representing a cube with 1,000 balls of various size randomly placed in the volume (such geometries are used for the simulation of concrete, where the balls model the aggregates, and the volume between the spheres models the cement). This is a typically challenging case for a mesher because of the numerous randomly placed mesh constraints generated by the balls, and even more challenging in a distributed paradigm such as the one used by out-of-core or parallel meshers.

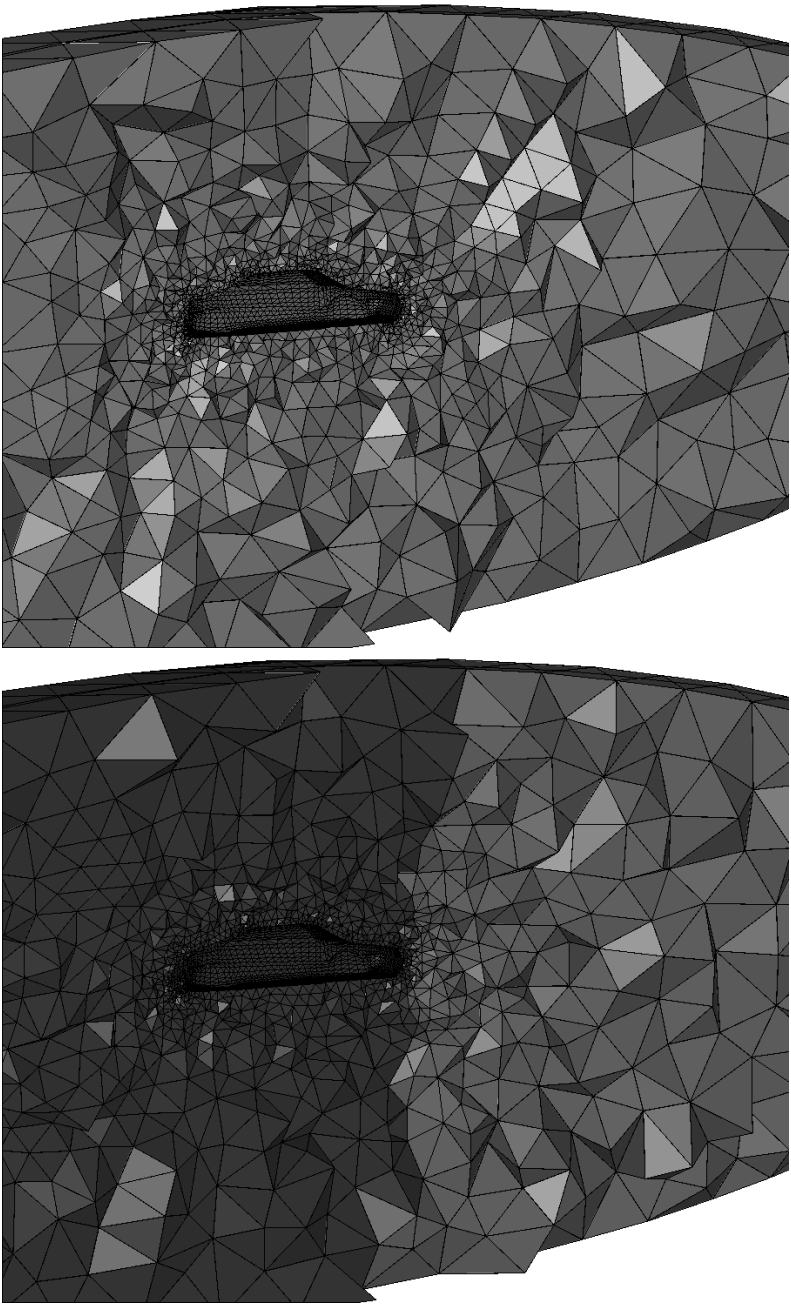
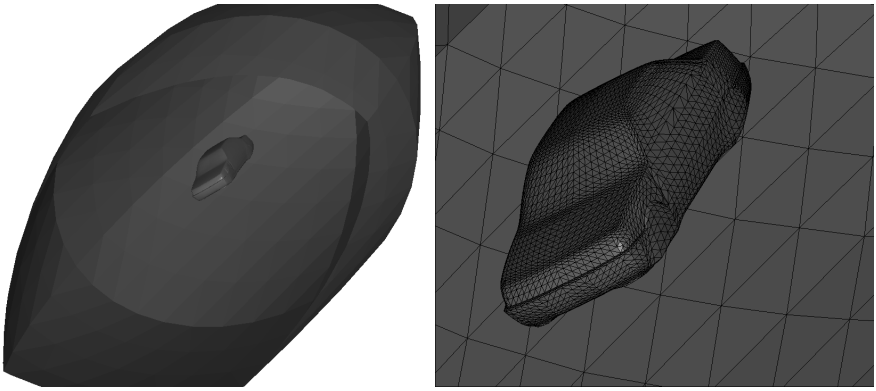


**Fig. 4.** Comparison of sequential (above) and out-of-core (below) meshes ; colors identify the two sub-domains on the latter. View is a cut orthogonal to the bisection plane.



**General view** **Close view of car**

| | Sequential | Out-of-core | Difference |
|---|---|---|---|
| $q \in [1..2[$ | 107,790 | 84,757 | -23,033 |
| $q \in [2..3[$ | 8,993 | 6,539 | -2,454 |
| $q \in [3..4[$ | 648 | 626 | -22 |
| $q \in [4..5[$ | 103 | 129 | 26 |
| $q \in [5..6[$ | 30 | 26 | -4 |
| $q \in [6..7[$ | 9 | 9 | 0 |
| $q \in [7..8[$ | 1 | 3 | 2 |
| $q \in [8..9[$ | 3 | 2 | -1 |
| $q \in [9..10[$ | 4 | 3 | -1 |
| $q \in [10..12.37[$ | 4 | | |
| $q \in [10..12.59[$ | | 2 | -2 |
| Worst quality | 12.37 | 12.59 | |
| Number of cells | 117,585 | 92,096 | -25,489 |

**Quality comparison**

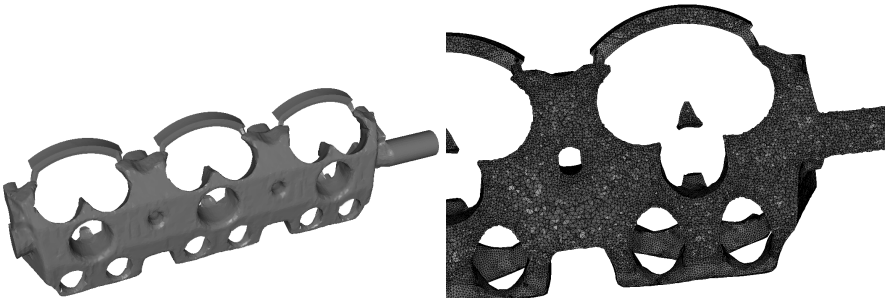
**Fig. 5.** Case #1, Peugeot Car (courtesy of PSA Group)

The tables below the figures expose the quality diagrams of the 5 test cases. For each case, they give the distribution of the tetrahedra in terms of quality, the worst tetrahedron quality and the total number of generated elements.

The element quality we consider here is the finite element quality

$$q = \alpha \frac{h}{\rho}$$

where: $h$ is the element diameter, $\rho$ the inradius, and $\alpha$ a normalisation coefficient. This quality varies from 1 to $\infty$. The closer $q$ is to 1, the better the element is.



**Geometry** **Volume mesh cutting plane**

| | Sequential | Out-of-core | Difference |
|---|---|---|---|
| $q \in [1..2[$ | 2,560,869 | 2,507,554 | -53,315 |
| $q \in [2..3[$ | 43,911 | 45,461 | 1,550 |
| $q \in [3..4[$ | 406 | 420 | 14 |
| $q \in [4..5[$ | 93 | 86 | -7 |
| $q \in [5..6[$ | 34 | 25 | -9 |
| $q \in [6..101.8[$ | 22 | 25 | 3 |
| Worst quality | 101.8 | 101.8 | |
| Number of cells | 2,605,335 | 2,553,571 | -51,764 |

**Quality comparison**

**Fig. 6.** Case #2, Cylinder Head (courtesy of IFP)

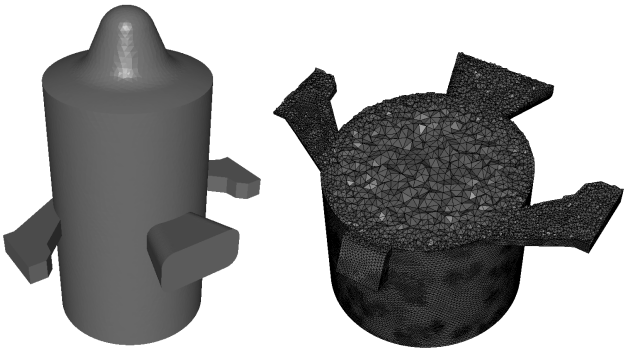
These tables show that the meshes generated by our method with a 4 subdomains decomposition are very close to the sequentially generated ones in terms of size and quality comparison.

All these comparisons demonstrate our method's ability to produce meshes very close to their sequential counterpart (when it exists).

### 2.3 Large scale out-of-core meshing

For the last application, we were interested in testing our method for the generation of very large meshes, such as the ones one will eventually use to simulate with great accuracy the behaviour of a given concrete mixture under stress. Ultimately, these simulations will use tomographic experimental information for the distribution and shape of the numerous aggregates, making it very important to be able to mesh volumes with a very large number of constrained polygonal surface cells of various shapes.

These geometries are very challenging for several reasons: the large number, the varying sizes, the distribution and proximity of the aggregates impose very strong constraints firstly to the sequential volume mesher used, and secondly to our method. The level of details required also induces very large meshes, making them practically impossible to generate with classical methods.

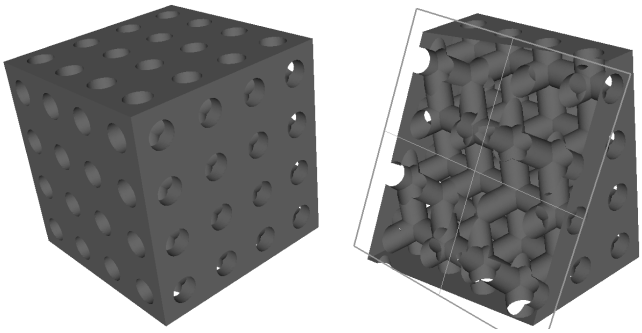


**Geometry** **Volume mesh cutting plane**

| | Sequential | Out-of-core | Difference |
|---|---|---|---|
| $q \in [1..2[$ | 1,502,094 | 1,401,389 | -100,705 |
| $q \in [2..3[$ | 22,826 | 23,518 | 692 |
| $q \in [3..4.597[$ | 18 | | |
| $q \in [3..4.051[$ | | 17 | -1 |
| Worst quality | 4.597 | 4.051 | |
| Number of cells | 1,524,938 | 1,424,924 | -100,014 |

**Quality comparison**

**Fig. 7.** Case #3, 2 Stroke Engine

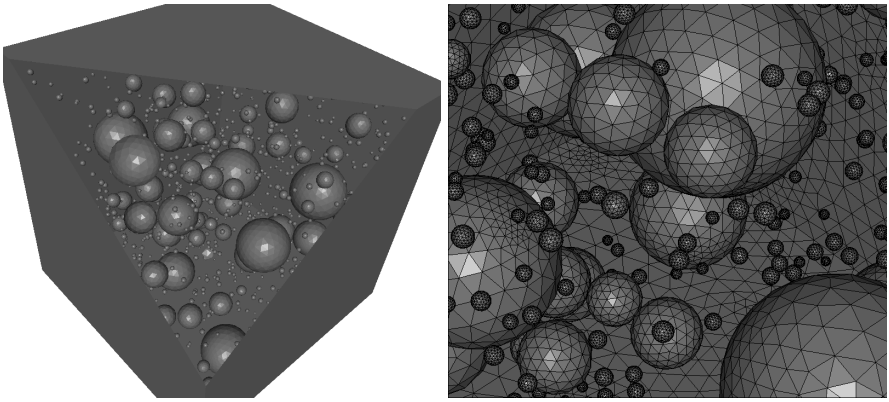


**General view** **General cut view**

| | Sequential | Out-of-core | Difference |
|---|---|---|---|
| $q \in [1..2[$ | 10,613,508 | 10,358,106 | -255,402 |
| $q \in [2..3[$ | 149,971 | 100,248 | -49,723 |
| $q \in [3..7.495[$ | 117 | | |
| $q \in [3..6.717[$ | | 67 | -50 |
| Worst quality | 7.495 | 6.717 | |
| Number of cells | 10,763,596 | 10,458,421 | -305,175 |

**Quality comparison**

**Fig. 8.** Case #4, Non Trivial Artificial Geometry



**General cut view** **Inside view**

| | Sequential | Out-of-core | Difference |
|---|---|---|---|
| $q \in [1..2[$ | 17,826,145 | 16,841,679 | -984,466 |
| $q \in [2..12.81[$ | 157,995 | | |
| $q \in [2..6.979[$ | | 226,487 | 68,492 |
| Worst quality | 12.81 | 6.979 | |
| Number of cells | 17,984,140 | 17,068,166 | -915,974 |

**Quality comparison**

**Fig. 9.** Case #5, Geometry for the modelling of concrete (courtesy of CEA-DEN)

For this purpose, we used two geometries and corresponding surface meshes.

### Refined Mesh of Case #5

We refined the input surface mesh of the previous figure 9 case (using the surface remesher Yams[4]) and successfully computed out-of-core sub-domain decomposition using respectively 16, 32 and 64 sub-domains.

The resulting meshes contain approximately 211 million tetrahedra (the inside of the 1,000 included balls was not meshed because the cement behaviour is targetted here). The figure 10 shows the boundary surface mesh of a sub-domain for the 16 sub-domain generation. The computation was carried out in out-of-core mode with our approach on a 4GB Opteron@2.4GHz machine in approximately 5 hours. Generating this mesh with the whole-in-core double precision sequential mesher would have required approximately 60GB of memory.

The memory used at the first bisection was 4GB. At the final sub-domain meshing step it was 3.6GB (respectively 1.8GB) for a decomposition in 16 (respectively 32) sub-domains.

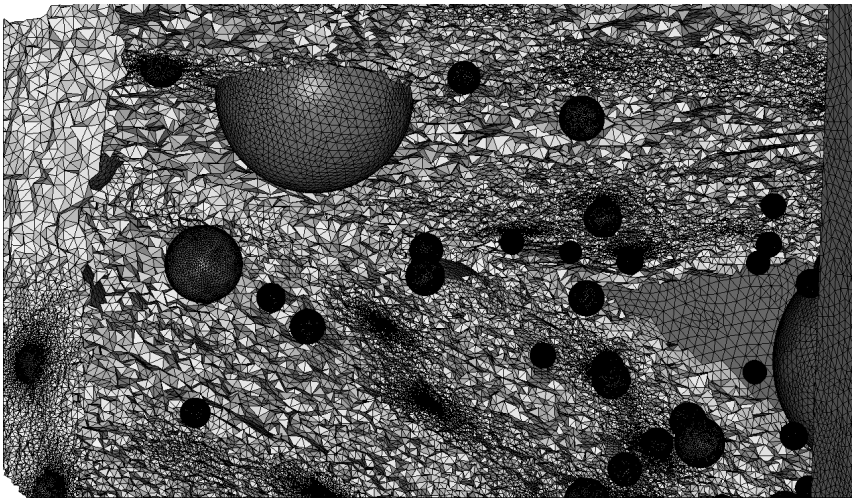
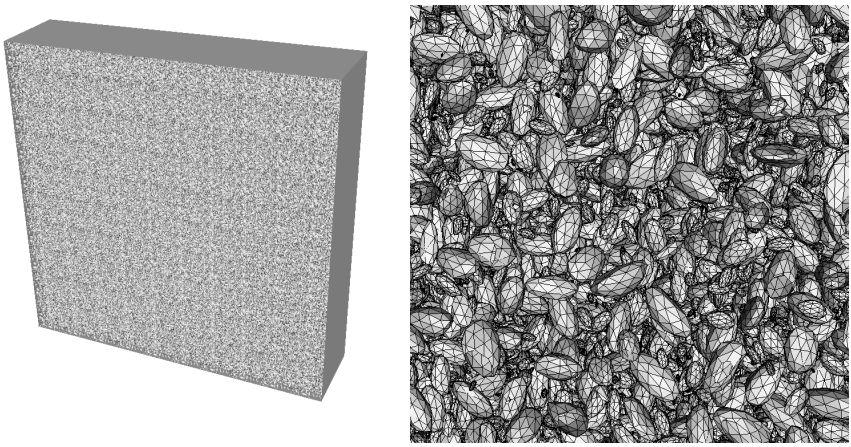


**Fig. 10.** The inside of a sub-domain, 211 million cells case (input boundary surface and interface triangles only)



**General cut view** **Inside view**

**Fig. 11.** Case #6, Geometry for the modelling of concrete (courtesy of C-S)

We can also note that we successfully tested our method on the case #5 geometry with partitions as large as 4096 sub-domains.

### Case #6: 500,000 polyhedral cells concrete model

The last application aims at demonstrating our method's robustness through the very high complexity of the geometry : 500,000 polyhedral cells of various shapes and sizes, randomly placed in a rectangular box.

The geometry is shown on figure 11 and provided courtesy of C-S. The input surface mesh contains 40.5 million triangles and 21.2 million vertices. This kind of geometry with a very large input boundary surface defining a rather small volume is one of the worst possible situations regarding memory consumption of our method (see section 1.4), the initial coarse mesh already being very dense.

The resulting volume mesh generated in out-of-core mode contains approximately 680 million tetrahedra (the inside of the 500,000 polyhedral cells was meshed in this case).

The computation of this case was carried out in out-of-core mode on a 64GB Bull Novascale system in double precision for 8 sub-domains. It took approximately 3 days to compute, with almost a third of it spent reading and writing the files, and required 60GB of memory. Generating this mesh with our whole-in-core double precision sequential mesher, would have required approximately 140GB of memory. This is a typical example of extreme geometry hard to mesh in sequential, and even more challenging in parallel because of the constraint imposed by the surface mesh, which must be preserved exactly in the volume mesh. It is also particularly hard for our method, because of the fact that a large amount of the final vertices are already present in the input mesh.

## Conclusion and future works

We presented in this article a strongly uncoupled domain decomposition method that relies heavily on an existing sequential mesher with very little modifications. It is able to generate distributed meshes in which parallel interfaces are almost the same as if they came from an *a posteriori* partition of the full sequential mesh.

This method was successfully applied to generate, in an out-of-core mode, large and good-quality meshes on limited core-memory resources, thus increasing the memory capacity by an order of magnitude.

The out-of-core method presented here can be easily adapted to implement a fully parallel message-passing approach, allowing the generation of very large tetrahedral meshes on distributed memory parallel systems more quickly.

It will also offer the advantage of providing an overall behaviour very similar to the original whole-in-core sequential algorithm.

Some adaptation will be made to the vertex generation and insertion, and the cutting zone mesh optimisation by vertex moving to get a consistently better mesh.

*Acknowledgement.* This work has been partially funded by the Pôle System@tic's IOLS project and achieved with Distene/INRIA tools TetMesh-GHS3D and Yams. The authors would like to thank Institut Français du Pétrole (IFP), Communications et Systèmes (C-S), and Commissariat à l'Énergie Atomique (CEA-DEN) for the test cases, and Bull S.A. for providing access to a Novascale system.

## References

1. Chrisochoides, N.P. (2005) A survey of parallel mesh generation methods. Tech. Rep. BrownSC-2005-09, Brown University. Also appears as a chapter in *Numerical Solution of Partial Differential Equations on Parallel Computers* Eds. Bruaset, A.M., Bjørstad, P.E., Tveito, A., Springer Verlag.
2. Farhat, C. and Lesoinne, M. Automatic Partitioning of Unstructured Meshes for the Parallel Solution of Problems in Computational Mechanics *International Journal for the Numerical Methods in Engineering*, 36, pp.745-764
3. Fezoui, L., Loriot, F., Loriot, M., and Regere, J. A 2-D Finite Volume / Finite Element Euler Solver on a MIMD Parallel Machine *Proceedings of "High Performance Computing II" Montpellier, Oct. 1991, M. Durand and F. El Dabaghi Ed., Elsevier Science Publishers, North-Holland, (1991).*
4. Frey, P.J. (2000) About surface remeshing. *Proceedings of 9th International Meshing Roundtable*, pp. 123-136.
5. Frey, P.J., and George, P.L. (2000) Mesh Generation, application to finite element. Hermes, Paris.
6. Galtier, J., and George, P.L. (1996) Prepartitioning as a way to mesh subdomains in parallel. *5th International Meshing Roundtable*, Sandia National Laboratories, pp. 107–122.
7. George, P.L., Hecht, F., and Saltel, E. (1991) Automatic mesh generator with specified boundary, *Computer Methods in Applied Mechanics and Engineering*, 92:269–288
8. Ito, Y., Shih, A.M., Erukala, A., Soni, B.K., Chernikov, A., Chrisochoides, N., and Nakahashi, K. (2006), Parallel Unstructured Mesh Generation by an Advancing Front Method, *Mathematics and Computers in Simulation*, (accepted).
9. Ivanov, E.G., Andra, H. and Kudryavtsev, A.N. (2006) Domain decomposition approach for automatic parallel generation of tetrahedral grids, *Computational methods in applied mathematics* , Vol.6, No.2, pp.178-193.
10. Larwood, B.G., Weatherill, N.P., Hassan, O., Morgan, K. (2003) Domain decomposition approach for parallel unstructured mesh generation, *International Journal for Numerical Methods in Engineering*, Vol. 58, Issue 2 , pp.177-188
11. Linardakis, L., and Chrisochoides, N. (2007) Graded Delaunay Decoupling Method for Parallel Guaranteed Quality Planar Mesh Generation, *SIAM Journal on Scientific Computing*, 27, submitted December 2006.
12. Said, R., Weatherill, N.P., Morgan, K., Verhoeven, N.A. (1999) Distributed parallel Delaunay mesh generation. *Computer Methods in Applied Mechanics and Engineering*, 177(1-2), pp.109-125.