A NEW APPROACH TO OCTREE-BASED HEXAHEDRAL MESHING

Loïc MARÉCHAL

SIMULOG & INRIA, Gamma Project,
Domaine de Voluceau, Rocquencourt,
BP 105, 78153 Le Chesnay Cedex, France.
Email: loic.marechal@inria.fr

ABSTRACT

A number of algorithms have been proposed to fulfill the strong demand for fully automated all-hexahedral meshers. Since none of them solved thoroughly the problem, people divided it into simpler ones. They enumerated the major kinds of configurations encountered in industry and tried to develop a set of applications, each of them focusing on a single problem. This paper presents a new approach to the octree-based algorithm and aims to add a new solution to the engineer "hex-toolkit". This method has been introduced by R. Schneiders [15] and is, likewise, done in three steps: *(i)* An octree structure is built around a discrete geometry. *(ii)* A conformal, all-hexahedral mesh is generated by inserting the so-called *conforming patterns* (which are completely new ones). *(iii)* The boundary of the geometry is recovered by a smoothing technique. This rough method is fast and automated for any kind of geometries. Furthermore, mesh adaptation for finite element computation is easily implemented, thanks to the octree hierarchic structure.

Keywords: Volume meshing, Unstructured Hexahedra, Octree, Conforming patterns, Mesh adaptation.

1. INTRODUCTION

The demand for a fully automated all-hexahedral mesher has been strong for the past decade. Various methods have been proposed in those times. Since none of them addressed the problem thoroughly, people focused on a subset of the user's requirements, relaxing one or more constraints. Thus, methods like unconformal hex, semi-automated or non general-purpose schemes evolved into products.

If the perfect hex-mesher is a chimera, it could be possible to develop a set of different algorithms covering a wide range of problems which could do the trick.

Recently, T.Blaker gave a survey of problems encountered and solutions used nowadays in hex meshing [2]. While the purpose of this paper is not to make a complete state of the art in hex-meshing, i have to recall briefly the different kinds of methods currently available:

- block decomposition, which gives good quality meshes but is not fully automated and not of general purpose,
- advancing-front which could produce good meshes on a wide variety of geometries, but is still in the state of research,
- tetrahedra decomposition into hexahedra is the most reliable method, but produces poorly shaped elements,

- grid/octree methods are automated, robust and generic but have quality related problems near the boundary,

Right now, the whole range of available products cannot cover all the needs. Consequently there exists some demand for new methods completing the user's *hex-toolkit*. For instance, some people still require a so-called *meshing-black-box*. An algorithm robust enough to mesh any kind of geometries without human intervention, even though they loose fine control on the element orientation, position and quality. Of course they would rather prefer a perfect quality mesh, but for those people, robustness and ease of use is more important than the shape of hexahedra.

I started developing a new method to fill this lack of robust automated general-purpose, but rough quality hex-mesh generator. The goal is to produce a better quality mesher than a simple tet to hex splitting algorithm (which is the safest way to get hexes), even though it may not match block decomposition methods (in the case they are applicable !).

2. A CRUDE BUT EFFICIENT METHOD

2.1 Why an octree ?

I choose the octree method for several reasons:

- it is a robust method already implemented in many domain decomposition algorithms,
- it is applicable to most kind of geometries,
- it seems natural to use an octree as basis structure for hex-meshing since the *octants* in the so constructed tree are perfect cubes !

Although it has the following drawbacks:

- octants are isotropic elements, thus, very anisotropic geometries will lead to a great number of hexahedra,
- since there are hanging nodes in the octree (Fig. 1), it produces unconformal hexahedra.

2.2 Basic scheme of the method

I take as input a triangulation of the boundary of the domain to be meshed, then I create a volumic hexahedral mesh and a surface quadrilateral mesh on the fly. Here is a quick overview of this three steps algorithm. Each step will be described in the further sections.

Remarque 1 (Topology). The surface mesh is used as a discrete support to describe the geometry. Its topology (vertices, edges, and, obviously, triangles) won't be preserved in the final hex-mesh because the vertices of the quadrilateral surface of the volume mesh will only be projected on the surface triangles.

Octree building: At first, an octree is constructed around this surface mesh so that it represents the details with the right size. At this stage it is possible to intersect the octree with a size map describing the required size of elements in every location of space (see [5] for adaptation process using quadtree/octree). This octree is made of hexahedra only but is an unconformal mesh (Fig. 1).

Octree conforming: Then, some hexahedra are inserted inside the unconformal elements of the octree in order to avoid hanging nodes and obtain a conformal hexahedral mesh. To each configuration of unconformal element of the octree corresponds a set of hexahedra called *conforming-pattern* which connects every hanging nodes.

At this stage the result is an hex-mesh enclosing the bounding box of the object to be meshed, called a background mesh. Its elements are of varying sizes in order to match the details of the geometry or the required precision for a FEM computation. It is now time to recover the real boundary of the object. This is done by inserting the triangles of the boundary into the hexahedra of the volumic mesh. From this set of intersected hexes, I derive a new surface close enough to the geometry and project those hexahedra' faces onto it.

Boundary recovery: A final stage inserts a buffer layer under the surface of the hex mesh in order to improve the quality of the elements which have been distorted to match the geometry. A simple vertex smoothing is applied to smooth the size variations generated by the octree (where an element can have half, twice or the same size of its neighbours).

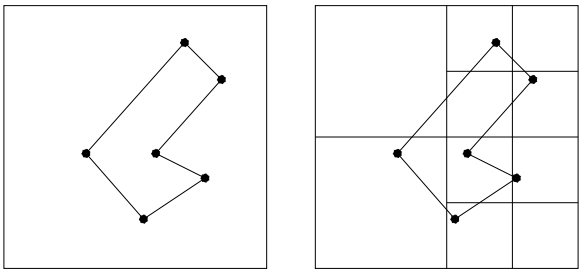

Figure 1. *Left: the geometry to be meshed enclosed in its bounding box. Right: the resulting 2D quadtree (octree in 3D). You may notice that the two biggest quadrilaterals have hanging nodes on their right edge. Consequently, they are unconformal elements. Here I used a simple criterion imposing that each quadrilateral cell contains no more than one vertex of the boundary.*

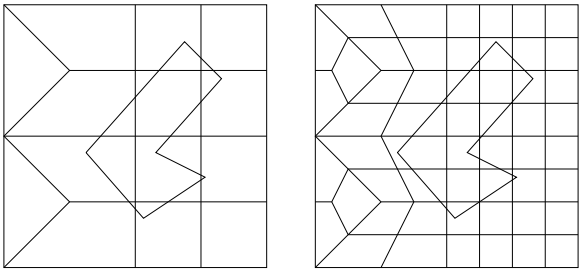

Figure 2. *This figure shows the simplified conforming process in 2D. Mixed-element patterns are inserted (left), then quads are split in four and triangles in three quads in order to obtain a quad only conformal mesh (right).*

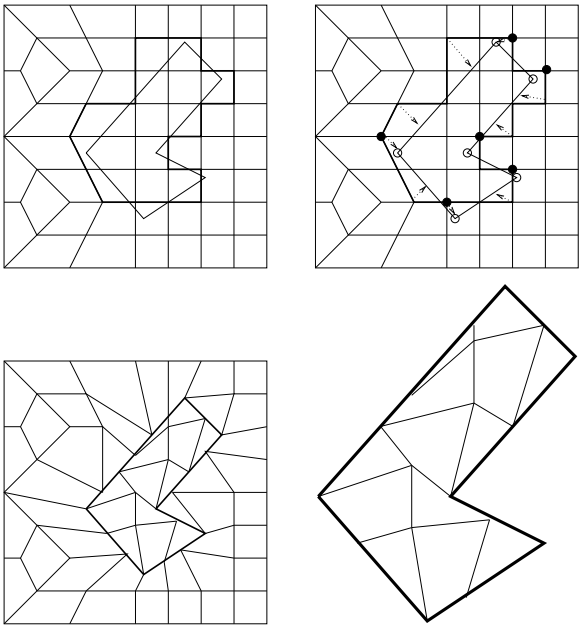

Figure 3. *Recovery of boundary process: Find a boundary piece in the volumic mesh close enough to the geometry to be meshed. Then project those selected faces on the real surface and remove useless elements.*

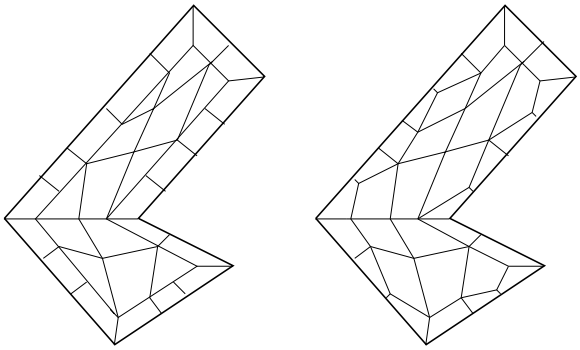

Figure 4. *A boundary layer is inserted (left) so that each element has no more than one edge (one face in 3d) stuck on the surface. Consequently, it is possible to "straighten" the distorted elements near the boundary.*

3. OCTREE BUILDING

The first step is the construction of a bounding box of the object. This box will be the octree's root. This octree is built in order to respect a set of criteria. The purpose of this paper is not to explain the complete octree theory, (see [13], [14] and [6]). Figure 1 shows the process in a simple $2D$ case.

The main characteristics of the octree are the criteria which are used to decide if a cell should be subdivided or not. If an octant is considered too large by at least one of the following criteria, it is be subdivided:

c1. an octant cannot be intersected by two triangles which are disjoint and form an angle greater than a given threshold,

c2. an octant cannot be intersected by two required edges (for geometrical or computational purpose) which are disjoint and form an angle greater than a given threshold,

c3. an unconformal octant (with hanging nodes on its faces and edges) cannot have more than 15 unconformal brothers (elements sharing a face or an edge with a given octant),

c4. an octant cannot have a size greater than the one specified by the size-map (evaluated at the center of the octant or the average size evaluated at its eight vertices).

The third criterion is very important because it smoothes the element size across the mesh. It prevents useless variations that disturb the mesh regularity and quality (Fig. 5). Furthermore, it reduces the number of unconformal elements in the octree and, consequently, the number of hexahedra added in order to "conformize" it.

4. OCTREE CONFORMING

4.1 Overview

On this second stage, I ensure the mesh conformity. An octree structure has a good property: it is composed of perfect quality hexes only. Unfortunately, there are unconformal elements with hanging nodes in the middle of their edges or faces (Fig. 6, drawing 1). This problem is solved by inserting a set of hexahedra in each unconformal elements so that they connect every hanging nodes. The difficulty is to define those set of elements (the so-called *conforming patterns*).

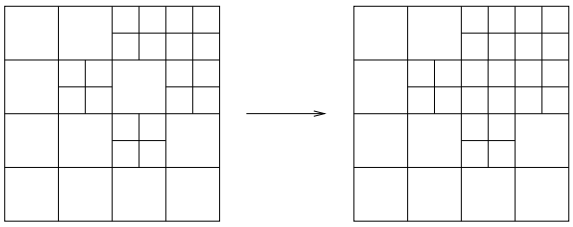

Figure 5. *An unconformal element is surrounded by four smaller one (left). After refining this cell, the mesh looks more regular. Without this criterion, the number of elements inserted to conformize the cell would have overcome the number of elements added by the refinement.*

Let me explain the problem: a hexahedron can have hanging nodes or not on its twelve edges and its six faces. Thus, there are some thousands of combinations and defining explicitly each of them would be too painful. So I choose to subdivide this problem into a four steps scheme in order to reduce its complexity.

4.2 Step 1: Reducing the complexity

Insert a star-shaped pattern inside each unconformal elements (Fig. 6) in order to reduce the number of combinations. An original element is replaced by seven hexes. The center hex of the star-shaped pattern cannot have hanging nodes and the six others, as they share only one face with their "father", can only have hanging nodes in the middle of this inherited face or its four edges. There are now 17 combinations of patterns required to mesh those new elements, but some of them are uneasy to find.

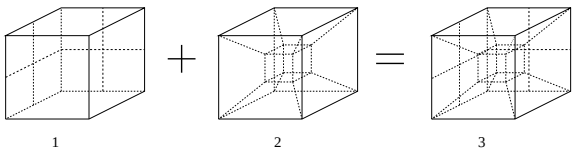

Figure 6. *After the insertion of the star-shaped pattern, there remain six hexes with a hanging node on only one face.*

4.3 Step 2: Still reducing the complexity

The second step's goal is to reduce again the number of combinations. Let's focus on a property of the mesh obtained after the first step: if two

same-sized elements of the octree share the same unconformal face (there are hanging nodes in the middle of the quadrilateral and/or its edges, see the thick face in fig. 7), a star-shaped pattern has been inserted in both elements.

Consequently, the element with the thick face in the fig. 7, drawing 1, and the other one in fig. 7, drawing 2, share the same face and are symmetrical since none of their other faces or edges can have hanging nodes.

Those two hexes will be merged by removing their common face, giving a decahedron (Fig. 7, drawing 5), which will be meshed with six conformal hexahedra (this operation is called a *two to six* split, Fig. 8).

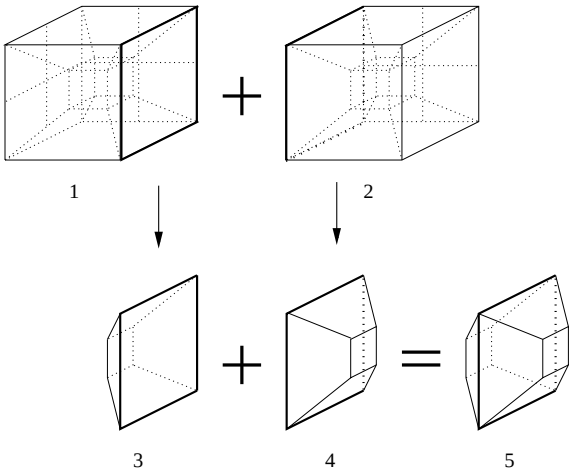

Figure 7. *Two neighbour by elements have two neighbours and symmetric sons. Those sons are merged to give a decahedron.*

Now each of those six hexes share only one edge with the two unconformal elements they come from (the thick face in fig. 8 is the same as the one shared by both elements in fig. 7, drawing 1 and 2). Thus, the elements generated by those two steps are either conformal or have only one unconformal edge with a hanging node (the thick edges in fig. 8). So only one conforming-pattern is needed in order to connect one edge's hanging-node to the other hex's nodes (Fig. 9, pattern 2).

4.4 Step 3: Transitional conformal but hybrid mesh

The third step's goal is to create a conforming mesh by inserting patterns inside unconformal elements. But now the number of combinations has

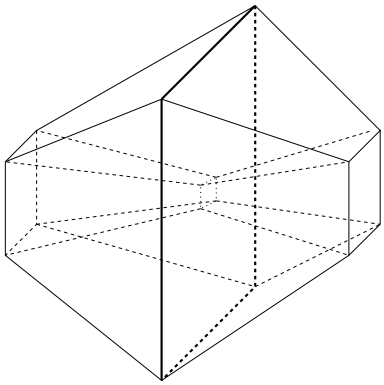

Figure 8. *The* 2 to 6 *pattern, which splits two neighbour hexes into six.*

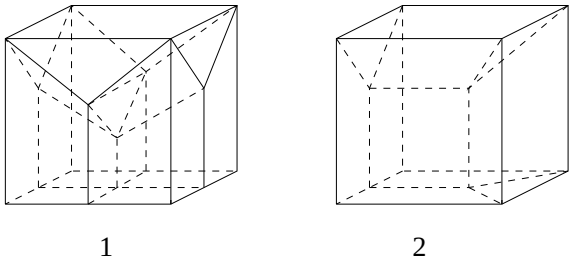

Figure 9. *At the end of step* 2*, there remain only two configurations of unconformal elements: those with a node in the middle of only one face and those with a node in the middle of only one edge .* 1 : *the pattern for an element with a hanging node in the middle of one face.* 2 : *the pattern for an element with a hanging node in the middle of one edge.*

been greatly reduced by the two previous steps. Indeed, there remain only two of them ! At this stage, an unconformal hex can have only one unconformal face or one unconformal edge. Now I only need two simple sets of patterns (Fig. 9 patterns 1 and 2) to conformize the mesh. As you may notice on the figure, the elements of the patterns are not hexahedra only. There are also pentahedra. After the third step the mesh is conformal but hybrid.

4.5 Step 4: Final conformal all-hexes mesh

The last step will subdivide every elements into hexahedra so that the mesh remains conformal but contains hexahedra only. The hexes will be split into height smaller hexes and the pentahedra will be split into six hexes as shown in the fig. 10.

After those four conforming steps, I obtain a back-

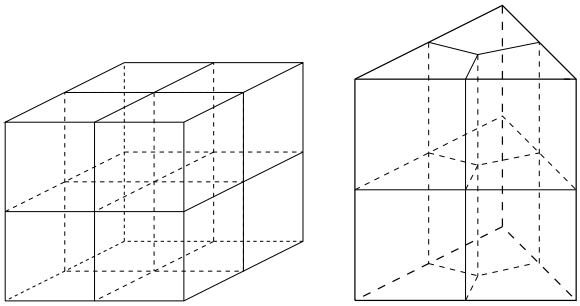

Figure 10. *An hex is split into height hexes and a penta into six hexes.*

ground mesh composed of conformal hexahedra only. This mesh respects the sizes required by the geometry and the computation but it doesn't represent this geometry. I will now enforce the boundary inside this volumic mesh.

Remarque 2. Those patterns allow to mesh any configuration and solve the problem of Schneiders' first set of patterns proposed in [15]. R. Schneiders solved also the problem in [16] with a new approach using "directional refinement".

5. BOUNDARY ENFORCING

5.1 A three steps process

First, the list of hexahedra intersected by the boundary triangles is established. Those hexes are called boundary hexes and some of their faces will be selected as the final boundary. Note that those boundary hexes must define a closed sub-domain, otherwise, the result will be a surface quadrilateral mesh without hexahedra.

The second step is the subdomain colouring. An hex close to the bounding box is chosen as a starting seed. Then a "flood-fill" algorithm colours the neighbouring elements all over the mesh without crossing boundary-hexes. The elements coloured in this way constitute the first sub-domain (the "exterior" that will be removed). The other uncoloured elements define the inside of the object.

At third, the faces shared by the coloured and the uncoloured hexes are set as the new boundary. This surface is called the *staircase boundary*, and will be mapped on the real boundary with a projection. Each vertex of the boundary being normal projected on the nearest triangle. Afterward, a simple smoothing is done in order to improve the quality of the quadrilaterals distorted by the above projection.

In case of multiple subdomains, those steps are recursively repeated.

5.2 Projection algorithm

Each hexahedra intersected by some boundary triangles is processed the following way:

If the hex contains a corner point, choose the closest vertex among those of the hex which belongs to the *staircase boundary*. Then simply move the selected hex's vertex to the corner position.

If the hex is intersected by some ridge-edges, the intersections between the ridges and the hex's faces are computed and the closest hex's vertices from those intersections are projected on the ridge as shown in Figure 11.

5.3 Distortion of boundary elements

The above process dramatically modifies the shape of the elements near the boundary. A hexahedron can degenerate into pentahedron if two of its faces are mapped on a plane or into a tetrahedron in case of three faces (Fig. 12).

At the end of this step, I obtain a volumic hexahedral mesh with several subdomains defined by quadrilateral boundaries. I am now able to enumerate the subdomains and to keep those of interest (interior or exterior for example). Figure 3 shows the process of boundary and subdomains recovery.

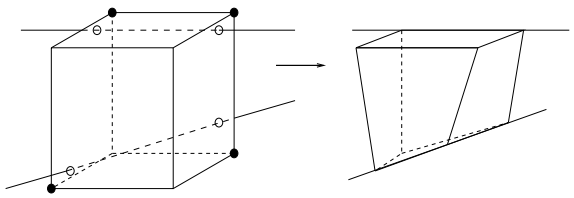

Figure 11. *Left: two ridges intersect this hex. The intersection between the ridges and the quadrilaterals are represented as small empty circles. The closest hex's vertices to be projected are represented as filled circles. Right: the distorted shape of the hex after projection of the selected vertices on the ridges.*

6. QUALITY IMPROVEMENT

The last step inserts a buffer-layer along the boundary elements in order to improve the quality

of the hexahedra which have several vertices stuck on the surface and consequently cannot be freely smoothed.

6.1 Buffer layer insertion

The insertion is done in three steps:

1. duplicate each surface vertices, duplicate inherit the position and attributes of their models,
2. connect the new vertices to the boundary using flat hexahedra and remove the boundary attribute of former boundary nodes,
3. smooth the former boundary nodes so that they get away from the duplicated nodes (Fig. 4).

This process is similar to an advancing front scheme, thus, it is not a 100 % reliable algorithm. In case of very sharp angles or very thin and complex geometry, the layer may be as distorted as the elements near the boundary. In such cases, invalid elements will remain in the final mesh. Hitherto, I could not fix this major problem. A fall back solution could be the modification of the geometry so that it is meshable without invalid elements.

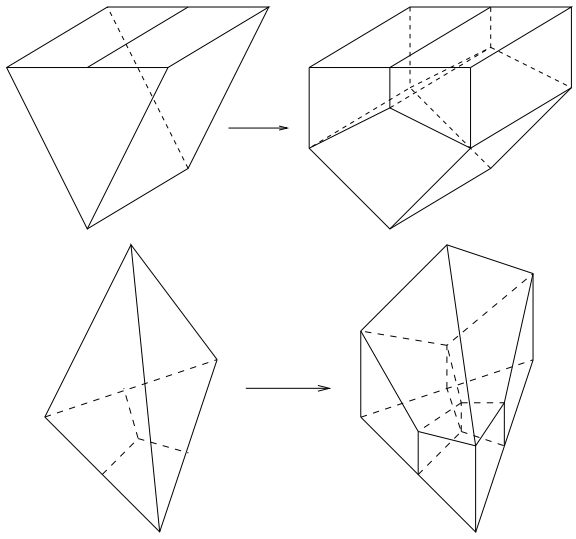

Figure 12. *Two cases of degenerated hexahedra near the boundary. Left: elements have two (upper left) or three (lower left) faces projected on a plane. After the insertion of a buffer layer, the smoothing generally improves their quality.*

6.2 Smoothing

The smoothing scheme is a very simple one as meshes generated with octree methods are "rigidified" by the inherent grid structure. In addition, hexes are very stiff elements. Moving on node twists each quadrilateral faces of a hexahedron so that their four vertices are no more coplanar.

The smoothing has two goals:

- reinflate some boundary elements flattened by the bending process,
- smooth the size gap between small transition elements (inside the star-shaped patterns, for example) and their bigger neighbours.

For that purpose, I need a quality criterion and a smoothing strategy.

Quality criterion In a first implementation I tried a complete criterion (indeed, the product of a set of criteria representing each a single property). Edges and diagonals sizes, faces shape, twist and area, and volume were taken into account. Such a criterion was too strict and the slightest displacement of one vertex lowered at least one of the subcriteria. Consequently, it was very difficult to find a new position for a vertex so that it improves the quality of all surrounding elements.

So, I came back to a basic but efficient criterion which compute only the height mix-products in order to guarantee the validity of the hex. A simple ratio between the min and max volumes gives a fair idea of the element's shape.

$$V_i = (\overrightarrow{N_{i0}N_{i1}} \times \overrightarrow{N_{i0}N_{i2}}) \cdot \overrightarrow{N_{i0}N_{i3}}$$

$$Q_e = \frac{\min(V_i)}{\max(V_i)}$$

Where N_{i0} is node i of the hexahedron and N_{ij} are its three neighbours connected through an edge.

Smoothing scheme

proc *smooth_mesh*
  for $v := 1$ **to** $nb_{vertices}$ **do**
    $push(stack) := vertices(v)$;
  od
  while $stack \neq \emptyset$ **do**

```
        v := pop(stack);
        Q_min := compute_min_quality(v);
        smooth_vertex(v);
        Q_min_opt := compute_min_quality(v);
        if Q_min_opt < α × Q_min
          cancel_displacement(v);
      else
          push_ball(v, stack);
        fi
  od
end
```

```
proc compute_min_quality(v)
  begin
    Q_min := 1;
    for i := 1 to degree(v) do
        if quality(hexahedra(ball(v, i)) ≤ Q_min
          Q_min := quality(hexahedra(ball(v, i));
        fi
    od
  end
```

This program pops a vertex from the stack and try to change its position so that it improves the quality of the worst hexahedra of its ball (the set of elements which share the same vertex). In case of success, the vertices of the ball's elements are pushed on the stack. Thus, the process will try to smooth again the other vertices of the ball as long as it finds better positions.

In the regions where hexes are perfect, the quality criterion will reject any displacement and no vertices will be pushed on the stack. Otherwise, near the boundary where bad elements stand, most of displacements will be accepted. Consequently, the stack will be filled up almost as fast as elements are popped and the vertices will be smoothed as many times as needed.

This stack-based smoothing shares the works out among elements according to their quality. Bad ones are allocated more smoothing-work than the better ones. The quality improvement factor α allows for a fine tuning of the convergence of the algorithm and thus the time spent. α must be included in the interval $[1, \infty]$. A small value means that any smoothing that does little improvement will be accepted. A bigger value means that only dramatic improvement will be accepted, thus pushing few vertices on the stack that will quickly be emptied.

7. SO WHAT ?

I am currently writing an implementation of this method in a piece of software called *hexotic*. In this section I give the first impression on the behaviour of this implementation. Please note that it is not yet fully operational (there is some work to be done in buffer-layer insertion and ridge recovery as well as speed and memory use).

7.1 Advantages of this method

Robustness: Octree method breaks up a global problem into several simple ones located inside octants. There are a finite number of different configurations encountered in an octant, thus, it is possible to enumerate all of them and to associate a solution with each of them. If an octree-based scheme is able to solve any configurations in an octant, it can solve any global problem that can be broken-up into octree. Although the implementation of the algorithm is still under development, I ran a script meshing 110 different geometries without any failure.

The program is also very tolerant about the quality of surface input mesh. It can mesh an unconformal, bad quality elements mesh with holes and gaps due to bad C.A.D. definition. In addition, setting the minimal size of element in the final mesh allows for some rough C.A.D. cleaning on the fly. This greatly reduces the time spent in "C.A.D. preparation".

Speed: The whole meshing speed (including I/O) range from 1,000 hexes/second to 10,000 h/s, the average value being 4,000 h/s. Indeed, the speed is not affected by the number of elements to generate (as the algorithm is almost linear) but depends on the compactness of the geometry. Since octree generates a bounding box enclosing the object, the more compact it is, the more useful elements will be kept in the final mesh. This increases the ratio $Speed = \frac{Nb\_useful\_elements}{Total\_time}$.

A general purpose method: *Hexotic* succeeded in meshing various kind of geometry ranging from turbines and planes to cylinder-head and human body.

Full automation and ease of use: The implementation consists of a simple command line program. It only requires one argument: the filename of the geometry to be meshed ! Optionaly,

the user may specify a size-map or min/max size of the hexes.

7.2 Drawbacks

Boundary orientation sensitive: The octree grid is aligned with the three axis x, y, z, thus, the elements cannot respect the orientation of the geometry. Likewise, two identical geometries only differing by a rotation or a translation won't produce the same volumic meshes.

Impossibility to respect a given boundary topology: The final quadrilateral surface only respects an analytical model derived from the discrete surface given as input. Thus, no vertices or edges from the triangulated will remain in the hexahedral mesh.

Isotropic only: Octants are perfect cubes. In order to mesh a thin geometry like a wing, the octree must be refined until the element size reaches the thickness of the wing. The mesh obtained will have much more elements than an anisotropic scheme which would have mesh the wing with a few hexahedra stretched along the main directions of the geometry.

Bad quality elements near the boundary: At present, there is still invalid elements (some of their mix products are negative) near the boundary. Bending and smoothing process have to be improved to remove most of them. But there will certainly remain invalid elements near very sharp angles and ridges unaligned with the x, y, z axis.

8. CONCLUSION

Hexotic does not intend to be the ultimate hexahedral mesher. The main goal is to provide the user with a new tool completing his hex-toolkit. Engineers could use it for investigation purpose in order to produce a first mesh of the studied model in to time. Depending whether the hex-mesh is suitable for computation or not, they could decide to keep it or to create another one with a better quality, but a more time consuming mesher.

9. EXAMPLES

Name	Nb of hexes	Speed	Figure
rod	6,151	4,823	17
falcon	10,992	2,510	18
turbine1	13,314	7,236	
geci	48,355	6,023	14
cow	49,779	6,800	15
helico	70,780	1,730	
af_gating	121,762	3,059	
cylinder head	132,745	4,271	16
commanche	304,329	3,443	
engine40	727,905	3,920	

Table 1. A selection of examples where some are depicted in the following figures. The meshing speeds (including any computation and input/output) are indicated in hexahedra per second.

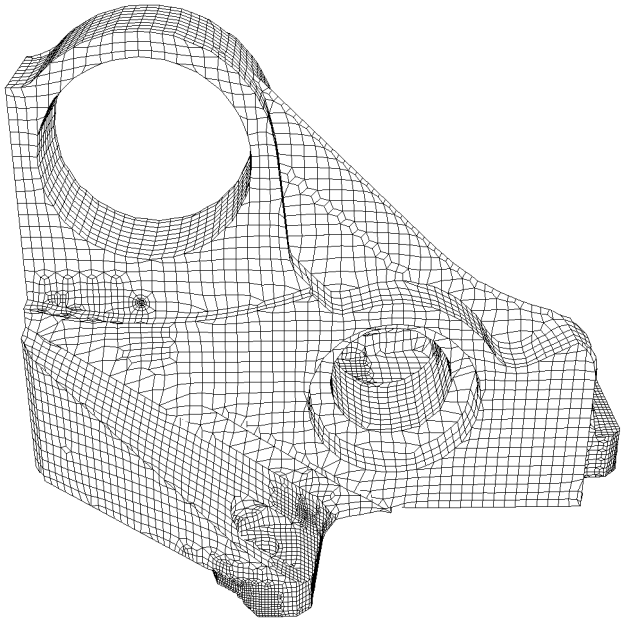

Figure 13. **External view of the hex-mesh. You may see that quadrilaterals are distorted near the ridges. Insertion of suitable patterns is needed as post-processing in order to avoid degenerated elements (not implemented at this time).**

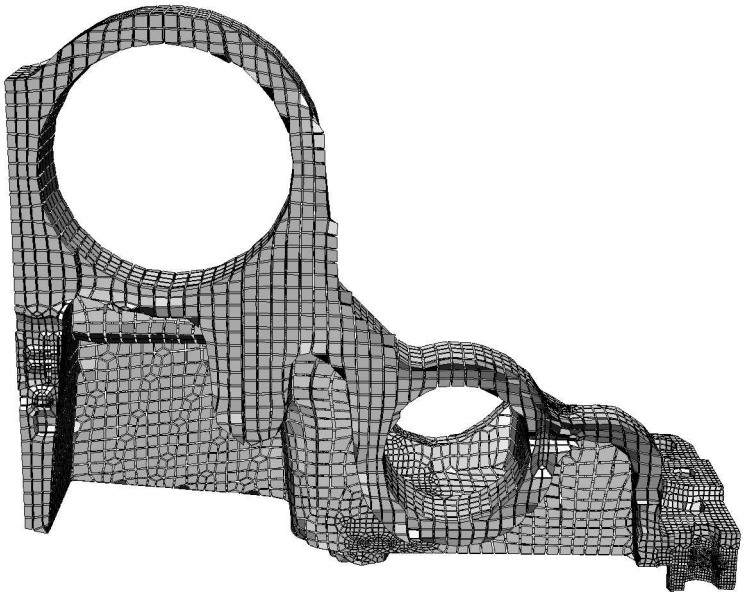

Figure 14. *Cut through a mechanical part. Hexes are shrunk around their center for better visualization. Since the buffer-layer is not yet inplemented, there are some degenerated elements near the boundary.*

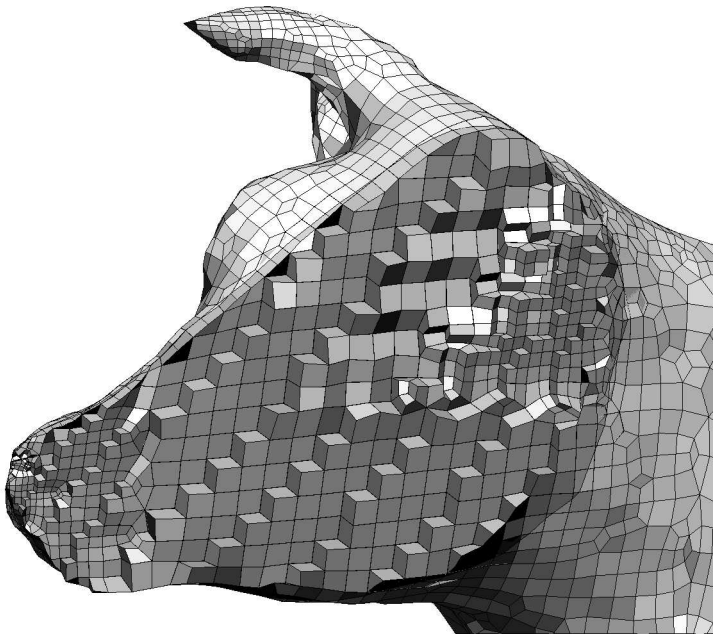

Figure 15. *Cow-head, cut through the hexes to show the transition elements. Hexes are bigger in the inner head and are getting smaller in the horn and the muffle*

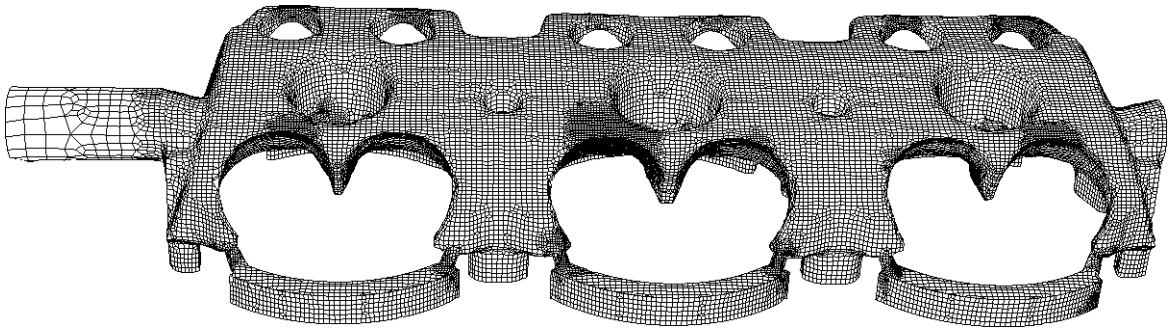

Figure 16. *Cylinder-head, resulting quadrilateral mesh. Notice the compactness of the refined regions. This mesh looks more like a structured-block mesh than an octree.*

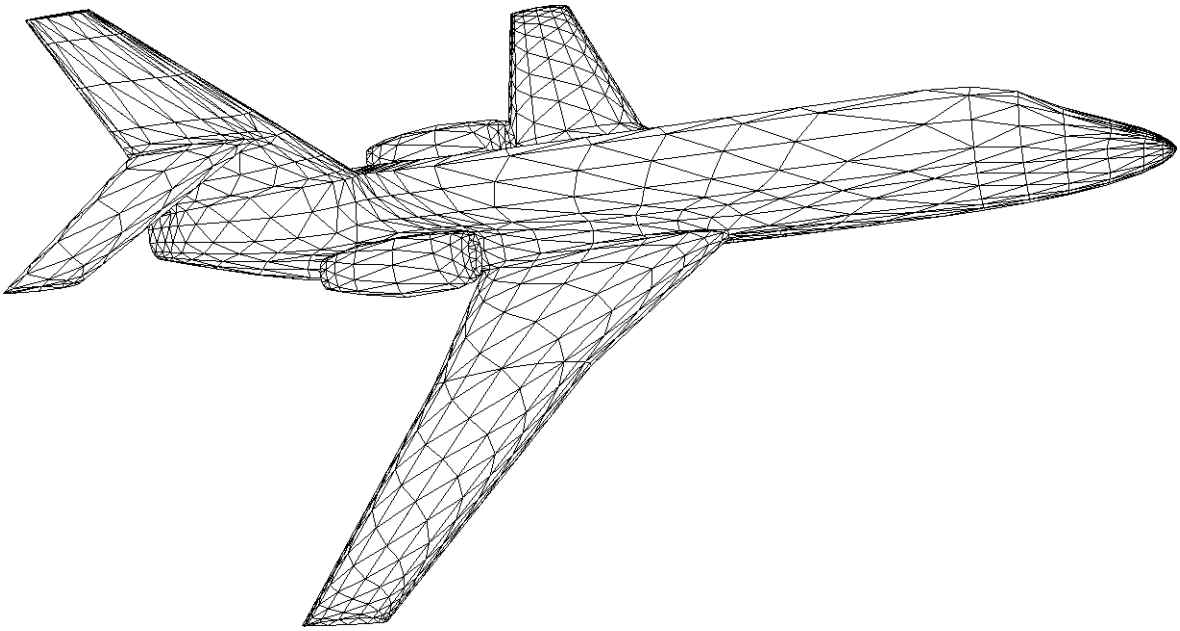

Figure 17. *The input triangulated surface of a falcon. If no user-specified sizemap is given, the mesher tries to mesh this geometry with as few elements as possible (trying to mesh each detail with the largest possible hexes).*

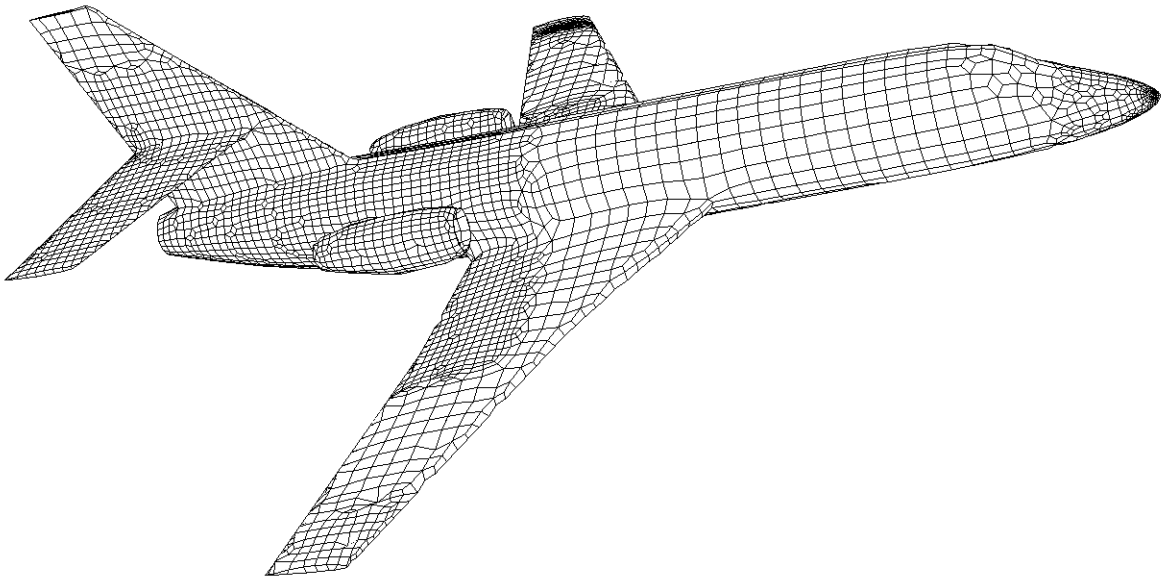

Figure 18. *And the resulting quadrilateral mesh. Notice that the size of quadrilaterals does not matches the size of triangles, but derives from the thickness of local details.*

REFERENCES

[1] C.G. ARMSTRONG & M.A. PRICE, Mat and associated technologies for structured meshing, *ECCOMAS*, 2000.

[2] T. BLACKER, Meeting the challenge for automated conformal hexahedral meshing, *IMR 9*, pp. 11-19, 2000.

[3] N.A. CALVO & S.R. IDELSOHN, All-hexahedral element meshing: Generation of the dual mesh by recurrent subdivision, *Comp. Meth. Appl. Mech. Engng*, Vol. 182, pp. 371-378, 2000.

[4] N.T. FOLWELL & S.A. MITCHELL, Reliable whisker weaving via curve contraction, *IMR 7*, pp. 365-378, 1998.

[5] P.J. FREY & L. MARÉCHAL, Fast adaptive quadtree mesh generation, *IMR 7*, pp. 211-224, 1998.

[6] P.J. FREY & P.L. GEORGE, Mesh Generation, *Hermes Science publishing*, chapter 5, 1999.

[7] M. LAI, S. BENZLEY & D. WHITE, Automated hexahedral mesh generation by generalized multiple source to multiple target sweeping, *Int. J. Numer. Meth. Engng*, Vol. 49, pp. 261-275, 2000.

[8] M. MÜLLER-HANNEMANN, Hexahedral mesh generation by successive dual cycle elimination, *IMR 7*, pp. 379-393, 1998.

[9] S.J. OWEN & S. SAIGAL, H-Morph an indirect approach to advancing front hex meshing, *Int. J. Numer. Meth. Engng*, Vol. 49, pp. 289-312, 2000.

[10] A. SHEFFER, A. RAPPOPORT & M. BERCOVIER, Hexahedral mesh generation using the embedded Voronoï graph, *IMR 7*, pp. 348-364, 1998.

[11] A. SHEFFER & M. BERCOVIER, Hexahedral meshing of non-linear volumes using Voronoï faces and edges, *Int. J. Numer. Meth. Engng*, Vol. 49, pp. 329-351, 2000.

[12] J. SHEPHERD, S. BENZLEY & S. MITCHELL, Interval assignment for volumes with holes, *Int. J. Numer. Meth. Engng*, Vol. 49, pp. 277-288, 2000.

[13] M.A.YERRY & M.S.SHEPHARD, A modified-quadtree approach to finite element mesh generation, *IEEE Computer Graphics Appl.*, Vol. 3, pp. 39-46, 1983.

[14] M.S. SHEPHARD & M.K. GEORGES, Automatic three-dimensional mesh generation by the finite octree technique, *SCOREC Report n° 1*, 1991.

[15] R. SCHNEIDERS & R. BÜNTEN, Automatic generation of hexahedral finite element meshes, *Computer Aided Geometric Design*, Vol. 12, pp. 693-707, 1995.

[16] R. SCHNEIDERS, Octree-based hexahedral mesh generation, *Int. J. of Comp. Geom. & Applications*, Vol. 10, n° 4, pp. 383-398, 2000.

[17] R. TAGHAVI, Automatic, parallel and fault tolerant mesh generation from CAD, *Engng. Comp*, Vol. 12, pp. 178-185, 1996.

[18] J.C. VASSBERG, Multi-block mesh extrusion driven by a globally elliptic system, *Int. J. Numer. Meth. Engng*, Vol. 49, pp. 3-15, 2000.

[19] D.R. WHITE & T.J. TAUTGES, Automatic scheme selection fot toolkit hex meshing, *Int. J. Numer. Meth. Engng*, Vol. 49, pp. 127-144, 2000.