Advances in Octree-Based All-Hexahedral Mesh Generation: Handling Sharp Features*

Loïc Maréchal

Gamma project, I.N.R.I.A., Rocquencourt, 78153 Le Chesnay, France
loic.marechal@inria.fr

Abstract. Even though many methods have been suggested to meet the challenge of all-hexahedral meshing, octree-based methods remain the most efficient from an engineering point of view. As of today, its robustness, speed and robustness are still unmatched. This paper presents advances made in the *Hexotic* project, especially in terms of sharp angles meshing, non-manifold geometries and adaptation.

Keywords: octree, meshing, hexahedra, adaptation.

1 Introduction

1.1 Motivation

Engineers facing the need for hexahedral meshes can access today only two kinds of software:

- Semi-automated block decomposition based methods like super block mapping, extrusion, sweeping algorithms, etc... These produce good quality meshes and enable fine grain control over the mesh, such as element position, orientation and size, but require a big human time cost. To begin with, a good deal of time is needed to learn how to use the software and acquire some knowhow. Afterwards, meshing each object requires a sizable amount of time, from hours to months depending on its complexity and the user's skill.
- Fully automated software, mostly based on the octree method. These products offer high speed, robustness and ease of use at the cost of mesh quality.

Consequently, researchers eager to improve an engineer's condition have but two ways to do so:

- the first is to reduce the human intervention required by semi-automated methods,

* This work was funded by the Pôle system@tic under the project E.H.P.O.C.

- the second is to improve the quality of meshes produced by octree-based methods.

This paper addresses the second alternative: what can be done to improve the capabilities of octree methods?

1.2 Limitations of Available Octree-Based Software

Most available products suffer from one ore more of these limitations:

- Non-conformity: hanging vertices are left in the final mesh, adding a burden to the solvers.
- Hybrid meshes: tets, pyramids and prisms are generated along with hexes. These software are often boasted as being hex-dominant, because they produce meshes made of 51% hexes... which is far from satisfactory for many solvers.
- Invalid elements: some products favor geometry accuracy over elements quality and as such tolerate some degenerated hexes (concave, negative volume).
- Smooth geometry: sharp features are smoothed out. No right or acute angles can be represented in the hex mesh. Such rough geometrical accuracy may be enough when dealing with M.R.I based data, but proves to be unsatisfactory for mechanical simulations.
- Constant sized-elements: all hexes are of the same, user-defined, size. Such a constraint generally improves quality but requires too many elements in order to capture small features.

1.3 Objective

We aimed at high objectives when starting this project, even though we knew that, in the end, we would have to water them down. Ideally, we would like to design a new approach to octree which would enforce the following features:

- all-hexahedral meshes,
- full automation, no intervention or knowledge should be required from the user,
- variable element size to capture each geometrical feature with the lowest amount of elements,
- valid meshes, 100% of hexes must have a positive jacobian,
- multiple subdomains and non-manifold subdomains, required in many complex mechanical devices,
- sharp angle meshing,
- adaptive meshing.

To achieve that goal, we chose to combine a modified octree, dual-mesh generation, extensive use of buffer-layers and a new vertex smoothing scheme. These efforts were undertaken under a project named *Hexotic*.

The overall scheme unfolds as:

1. Octree:
   a) analyze and check the input surface mesh
   b) compute the object's bounding box as a starting octant for the octree
   c) refine the octree according to a set of geometrical, physical or user-set criteria,
   d) balance and pair-up the octants,
   e) intersect octants with the triangulated surface,
2. Conforming:
   a) connect octree hanging vertices with the help of arbitrary polyhedra,
   b) build the dual-mesh from the polyhedral mesh,
3. Subdomains
   a) color topological sub-domains,
   b) control topological sub-domain thickness and topological pinches,
   c) insert a buffer layer of hexes around each volume subdomain,
   d) smooth vertices to improve elements quality,
4. Ridge meshing
   a) capture sharp edges on surface elements,
   b) group sharp-edged elements in surface pseudo-subdomains,
   c) insert a buffer layer of hexes around each surface subdomain,
   d) perform a final vertex smoothing while projecting surface vertices on the real geometry.

This paper will guide you through the whole set of meshing steps and present you with some results.

2 Octree Building

2.1 An Almost Regular Octree

Hexotic is based on a regular octree with little, but crucial, modifications:

- compute the object bounding box and use it as first octant,
- recursively subdivide octants according to a set of geometric and computational criteria,
- eventually, subdivide some octants to follow additional topologic rules (balancing, pairing).

Since the scope of this paper is not about fundamentals of octree meshing, reading [13] and [14] should be considered for more information.

2.2 Subdivision Criteria

Two subdivision criteria have been used:

Geometric diameter

Octants are split until they match the local geometry thickness.

Indeed, half this size is required to ensure that each feature is two-element thick. Meshing small features (shell or beam-shaped) with two hexes across allows for much less element distortion in the final boundary recovery process.

Furthermore, it gives the smoothing algorithm much more flexibility, since elements have one to four free vertices inside the volume. One-element thick meshes produce all-vertices locked hexes, which make the mesh extremely stiff in the final projection of surface vertices onto the geometry.

On the other hand, splitting octants once more than required increases the number of elements by a factor of eight in thin areas.

Size-map

In a mesh adaptation scheme, element size is driven by *a priori* or *a posteriori* error estimates. Defining the right size in each area of the mesh is achieved through a so called size-map.

It is made of a set of vertices mapping the whole mesh area, each vertex being associated to a target size. It is up to the solver to derive this size-map from the computation results. Once this size-map is generated, it should be provided, along with the surface mesh, to the hex-mesher. A very simple example is shown in section 10, fig. 16.

Other criteria could be used such as curvature angle or surface mesh triangles size. They are called "geometrical criteria" since they basically associate a size to any area within the bounding-box ($size = min_{i=1..n}(f_i(x, y, z))$, where $f_i(x, y, z)$ is a size criterion) and split octants until their sizes match the target.

2.3 Balancing and Pairing Rules

These rules are also called topological subdivision criteria, as opposed to geometrical criteria presented in the previous section. They set octants sizes depending on their neighbors sizes, enforcing topological rules, hence their name.

Hexotic applies two such criteria to the octree:

Balancing rule

This makes sure that no octant is more than twice or less than twice smaller than neighboring elements (sharing one or more vertices). In other words, the difference between two neighboring octants subdivision level cannot be greater than one.

Pairing rule

This one is quite unusual and its purpose will be fully explained in the following section. Basically, if an octant's son is to be split, its brothers (octants belonging to the same father element) should be split along with it.

Figure 1 shows the four steps an octree mesh goes through: single element bounding-box, octree subdivided according to geometrical rules, then after balancing rule, and finally, after pairing rule has been applied.

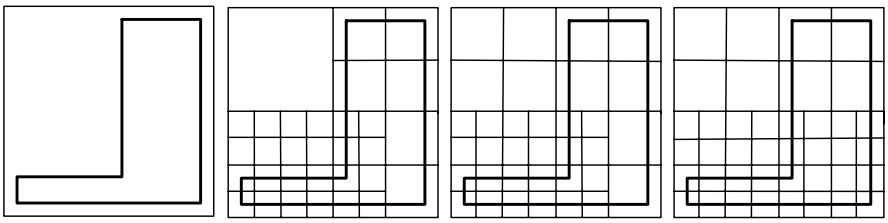

Fig. 1. (From left to right) Object inside its bounding-box; octree matching local features size; octree after balancing; octree after pairing.

After building the octree, the mesh is made only of hexes and each element size matches the size of geometry features in the corresponding area. Still, only the whole bounding-box is meshed and neither subdomains nor geometry boundaries are represented in this hex mesh. Furthermore, size variations between octants leave hanging vertices throughout the mesh, making it non-conformal. The next section will deal with this last issue.

3 Polyhedral Cutting

3.1 Connecting Hanging Vertices

Octree generation leaves us with non-conformal elements, that is, some octants have vertices hanging in the middle of neighboring faces or edges.

Connecting these hanging vertices using all-hex patterns has been a problem for many years.

Several solutions have been suggested (see [15] and [16]) some of them quite satisfactory, but we could do even better using a polyhedral and dual-mesh scheme.

The main idea is to connect hanging vertices using polyhedral elements. Any type of polyhedron and polygonal face can be used, which offers a much greater range of patterns to fill in non-conformal octants. The only restriction stands in the connectivity: the degree of vertices must be eight (there must be only eight elements sharing the same vertex), likewise, each edge can be shared by only four elements. Such a constraint is easily enforced if a cutting process is used to connect hanging vertices (see [2] and [6]).

Furthermore, the cutting process is split into as many steps as there are geometric dimensions. It is known as directional refinement (see [16]).

3.2 2-Dimensional Case

As a starting point, hanging vertices must be tagged according to the dimension they belong to. As depicted in figure 3, vertices hanging at the end of a vertical line are tagged with a small black dot, conversely, horizontal hanging vertices are tagged with a bigger white dot.

Then, we proceed with cutting elements with vertical tags as shown in figure 3 (left). This first cutting step can produce triangles and pentagons.

Finally, elements with horizontal tags are cut, like in figure 3 (right). This second step might cut previously cut polygons, thus cutting hexagons from pentagons.

Thanks to the pairing rule, the number of configurations is reduced to two in 2-dimension. Complex case like the one on the right in fig. 2 are avoided. An octant can only have one subdivided (non-conformal) edge (octants 1 and 3 in fig. 3) or two adjacent subdivided edges (octant 2 in fig. 3).

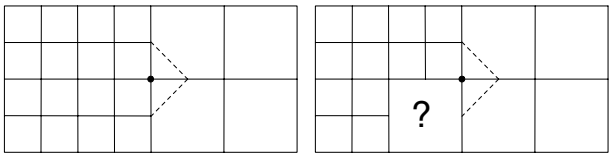

Fig. 2. (**left**) Correctly paired situation; (**right**) unauthorized unpaired situation, element "?" cannot be properly cut.

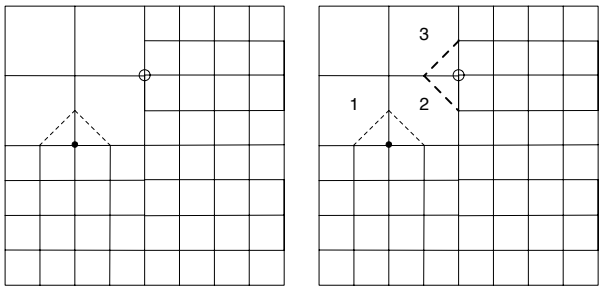

Fig. 3. (**left**) Polygonal mesh after vertical cutting; (**right**) and after horizontal cutting.

After the two cuttings steps, the mesh is conformal and it is made of polygons, ranging from triangles to hexagons. Vertices are shared by four elements (2-dimensional case).

3.3 3-Dimensional Case

The 3-dimensional case unfolds in the same way, but adds a third cutting step.

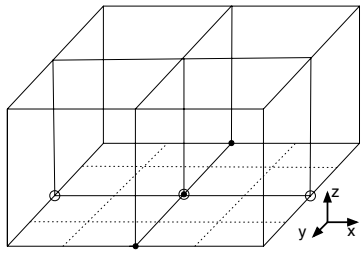

Fig. 4. Octree mesh before cutting: yz plane hanging vertices are tagged with small back dots and xz ones are tagged with bigger white dots. Dotted lines belong to neighboring smaller octants. Note that the vertex in the middle of the non-conformal face belongs to both planes.

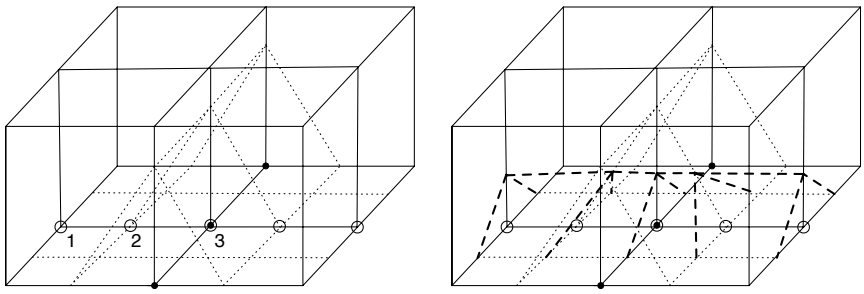

Fig. 5. (**left**) Polyhedral mesh after cutting along the yz plane. Vertices cut in the middle of an edge whose two vertices belong to the same cut plane, inherits the same plane tag. Since vertices 1 and 3 belong to the xz plane (white dots), the vertex 2, cut in the middle of edge 1-3, inherits a white dot tag. (**right**) Polyhedral mesh after two directional cuttings. Fine dotted lines come from the first cut (yz plane), and thick ones come from the second cut (xz plane).

Vertices are first tagged according to the plane they hang from: xy, xz or yz as shown in figure 4.

In 3-dimensions, a new rule must be set: when a vertex falls into the middle of an edge whose vertices have the same tag, then the newly cut vertex inherits this tag (see fig. 5).

The three cuttings steps produce a conformal mesh which is made of polyhedra.

These polyhedra range from four to sixteen polygonal faces, which range from triangles to hexagons.

Thanks to the cutting process, vertices are shared by eight polyhedra, twelve faces and six edges. Likewise, edges are shared by four polyhedra and four faces.

After this step, the mesh is conformal, but made of polyhedra, the next step will make it all hexahedral.

4 Dual-Mesh Generation

4.1 2-Dimensional Case

Generating a dual-mesh consists in creating vertices from primal-mesh elements, and elements from primal-mesh vertices.

A dual vertex is simply built in the barycenter of each primal element.

Creating a dual-element is a little more complicated. For each primal vertex, the ball of elements is computed (that is, the list of primal elements sharing this vertex). From this list of primal elements, the list of their dual-vertices is built. Finally, these dual-vertices are connected to form a dual-element. The process is shown in fig. 6.

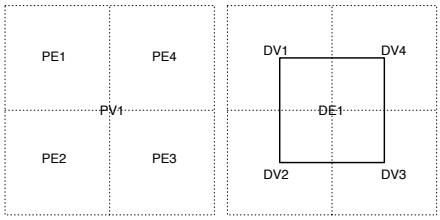

Fig. 6. (**left**) Four primal-mesh elements (PE) sharing a primal-mesh vertex (PV1); (**right**) the primal-mesh is in dotted lines and the dual one in thick lines. Dual vertices (DV) comes from PE, and the dual element DE comes from PV.

Consequently, the nature of elements in the dual-mesh depends on the vertex connectivity of the primal-mesh. Since primal vertices have a degree of four, then dual-mesh elements will be quadrilaterals. This way, an all-quad mesh is derived from a (not so) arbitrary polygonal mesh.

Figure 7 shows the only two possible configurations in the 2-dimensional case.

Remark

Having a look at the left figure, it appears that we just introduced a new kind of size transition. Well-known quadtree size transition rules are $2 \leftrightarrow 1$ and $3 \leftrightarrow 1$. The dual-mesh generation introduces a $4 \leftrightarrow 2$ transition as shown in fig. 8.

4.2 3-Dimensional Case

Dual vertices

They are set at the barycenter of primal-mesh polyhedra. A vertex is the dual of an element.

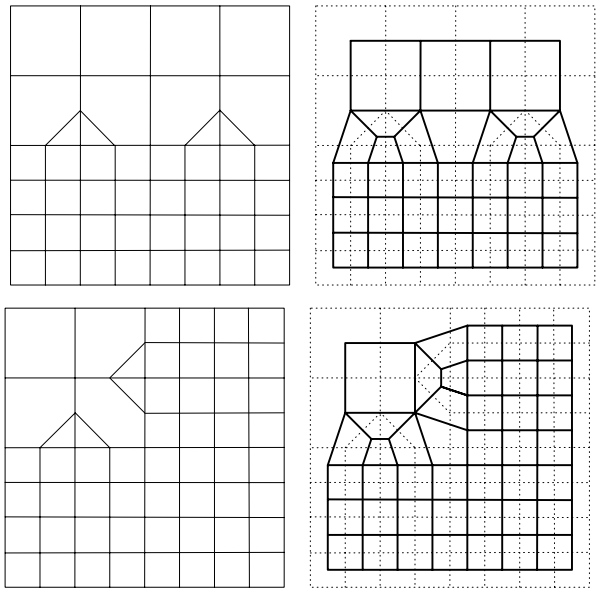

Fig. 7. (**left**) Primal-meshes; (**right**) primal-meshes in dotted lines and dual-meshes in thick lines.

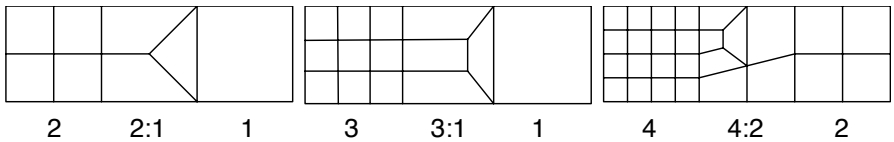

Fig. 8. (**left**) 2:1 size transition using mixed elements; (**center**) Schneider's 3:1 transition; (**right**) 4:2 dual-mesh transition.

Dual edges

They are derived from primal-mesh polygonal faces. A dual edge connects two dual vertices derived from two primal polyhedra sharing the same face. Thus, an edge is the dual of a face.

Dual faces

They are generated from primal edges. The shell (the set of elements which share a common edge) of each primal edge is built, then a dual face is made from the dual vertices associated to these elements. A face is the dual of an edge.

Dual elements

Conversely, they are made from the primal vertices. The ball of polyhedra sharing the same primal vertex is built and the dual element is made from the polyhedra-associated dual vertices. An element is the dual of a vertex.

Since the cutting steps produced only degree-four edges, corresponding dual faces have four vertices. Likewise, every primal vertices have a degree of six, producing dual-elements with six quadrilateral faces: hexahedra!

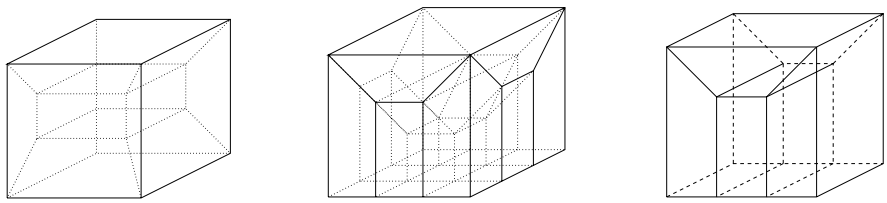

Fig. 9. (**left**) Pattern to mesh a non-conformal octant with one subdivided edge; (**center**) With one subdivided face; (**right**) And with two subdivided edges.

Not surprisingly, the resulting dual-mesh patterns look like Schneiders' original size transitions (see [15]). Only three different patterns are needed to mesh all transition cases (see fig. 9). The advantages over Schneiders' original scheme are:

- smoother size transition, edges are split in two instead of three,
- ability to mesh all configurations, even concave ones, which was impossible with the original method,
- fewer patterns needed.

After this step, the mesh is conformal and hex-only. Still, the mesh fills the whole bounding box and no boundaries nor subdomains are represented. The following steps are about subdomains recovery and boundaries meshing.

5 Subdomains Coloring

The subdomain recovery process begins with coloring concentric main domains. Then tries to color potential non-manifold subdomains within main domains. Finally, some filtering is made to remove one-element thin subdomains and various domain-pinches problems.

Each dual vertex derived form a primal hex intersected by a surface triangle is a boundary one. Conversely, each dual quad which has four boundary vertices is a boundary face.

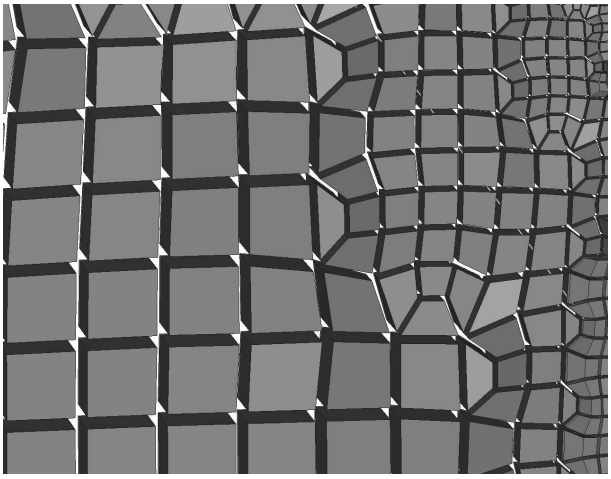

Fig. 10. Cut through a volume mesh: a regular Schneiders' size transition pattern set can mesh any configuration.

6 Geometry and Features Detection

In this step, the hex-mesh boundary quads are to be mapped on the real geometry.

Alongside, some edges of the hex-mesh should be mapped in order to capture sharp angles. These sharp angle edges, called ridges, are set via a user-defined sharp-angle threshold.

Setting objective triangles

Each newly set boundary quad should now be assigned a triangle from the input mesh onto which it will be projected. This process defines a mapping between the generated boundary-quad mesh and the triangular input mesh.

Setting ridges

If two neighboring quads are to be projected on triangles which make and angle greater than a user defined threshold, then their common edge is tagged as ridge. The input triangulated mesh is searched for the closest ridge edge to be assigned as objective feature.

Setting corners

Conversely, if two ridges sharing a common vertex make and angle greater than the threshold, this vertex is tagged as a corner. The closest corner is retrieved from the real geometry to be assigned as objective feature.

7 Buffer-Layer Insertion

Now, we are provided with an all-hexahedral mesh whose hexes are assigned subdomain numbers. Boundary quads, ridges and corners are assigned target geometry position to be projected on. Unfortunately, these boundary vertices cannot be projected yet on the real surface. This is caused by the fact that many boundary hexes have two or three faces to be projected on the same plane, which would make them invalid (see fig. 11). A layer of elements needs to be inserted around boundary hexes to tackle this problem.

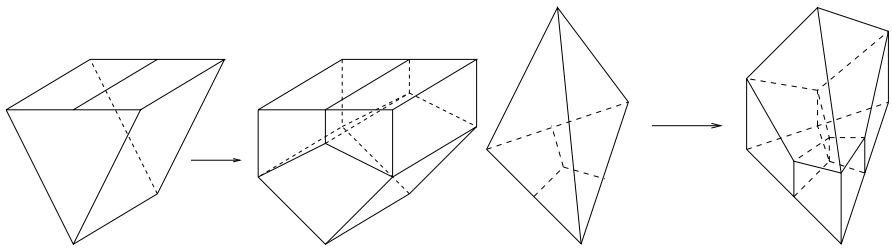

Fig. 11. Two cases of degenerated hexahedra near the boundary: before and after buffer layer insertion.

Inserting volume layers

Each subdomain is surrounded by a boundary layer. A new element is inserted between each boundary quad and the boundary hex it belongs to. This way, some hexes may be surrounded by up to three such boundary elements. Boundary hexes are topologically identical: they have one boundary face, and five internal ones. Thus, they can be freely projected on the real surface while preserving the element quality.

Inserting surface layers

Unfortunately, another problem was left unsolved by the boundary layer insertion. Indeed, some boundary quads have two-ridge edges to be projected onto the same line, making the element invalid (see fig. 12).

This problem is solved by inserting another boundary layer inside the first one. In this case, the subdomains do not come from the geometry, but are built on purpose. First, two-ridge faces are tagged. Then untagged faces standing between tagged ones are also tagged. Finally, adjacent tagged faces are packed together to form the so-called surface subdomains.

Eventually, each surface subdomain is wrapped in a layer a hexes, these new hexes having no more than one boundary face and one-ridge edge.

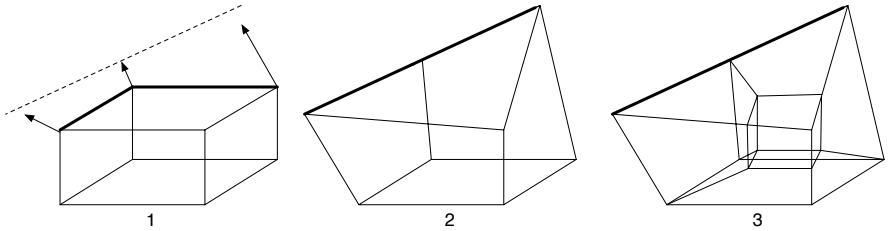

Fig. 12. (**1**) Initial cube with two edges to be projected on a ridge; (**2**) Degenerated element after projection; (**3**) Each hex of the buffer-layer has no more than one ridge-edge.

8 Smoothing

After the insertion of all these layers of elements, the mesh is topologically suitable for projection.

Projection is done on the fly while smoothing the vertices to increase the elements quality. Indeed, surface vertices are not explicitly projected on the geometry. The surface model appears as a constraint when computing the shape of optimal elements in the smoothing scheme.

During the 30 smoothing steps, internal vertices will move so that the quality of inner hexes gradually improves. Surface vertices will slowly be projected on the surface while preserving the quality of boundary hexes. Doing so, it is possible to lower the quality of an element below an acceptance threshold. In this case, we decided to favor element quality over geometry accuracy. Some surface vertices, especially those to be projected on sharp edges, may get stuck during the projection process.

Quality criterion

The quality criterion used is very simple:

$$q = 24\sqrt{3}\frac{V_{min}}{(\sum\limits_{1\leq i\leq 12} l_i)^{3/2}}$$

Where V_{min} is the minimum volume among the two sets of five tetrahedra corresponding to the hex (see fig. 13), and l_i are the lengths of the twelve edges.

Since there are as many hex quality criteria as there are solvers, it was pointless to try to find an ideal criterion. Instead, we came up with this basic one, which ranges from 0 (flat element) to 1 (perfect cube) and is negative in case of invalid element (negative volume).

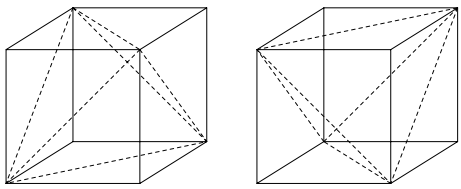

Fig. 13. Two sets of five tetrahedra cut from a hex.

The minimum quality accepted by hexotic is 0.01, which may be unsuitable for some solvers. Setting a higher target may result in poorer geometry accuracy.

A new element based smoothing scheme

This new optimizing scheme loops over elements instead of vertices. That is, for each hexahedron, an optimal element is computed, and its coordinates are added to a set of optimal vertex coordinates. Finally vertices are relaxed toward these optimal positions.

Such a scheme is faster since each optimal hex needs to be computed once. When looping over vertices, a hex optimal shape is computed for each of its vertices, that is, eight times. Furthermore, in this scheme, vertices move all together instead of one by one. When it comes to grid smoothing, a global displacement is preferable.

Computing the optimal element

For each hex, a target optimal element is computed. The main idea is to find the perfect cube, of which vertices are the closest to the original hex one. Thus, minimizing global vertices displacement when moving from initial positions to the optimal one. Here is the 2-dimensional scheme:

- compute the initial quad barycenter (C) and initial base vectors $\overrightarrow{V_1}$ and $\overrightarrow{V_2}$
- find the pair of orthogonal vectors $\overrightarrow{W_1}$ and $\overrightarrow{W_2}$, minimizing the sum:

$$\sum_{1 \leq d \leq 2} (\overrightarrow{V_d} \cdot \overrightarrow{W_d})^2$$

- build a perfect square made up from the center point C and base vectors $\overrightarrow{W_1}$ and $\overrightarrow{W_2}$ (its size being the average length of the original element, see fig. 14).

If a hex has a boundary face, then the perfect cube should be built so that its boundary quad lies on the constrained surface. In this case, one of the base vectors is set with the surface normal. Likewise, the center point C is moved to a position M (see fig. 15) so that the distance between M and its orthogonal projection on the constrained surface is half the size of the target element.

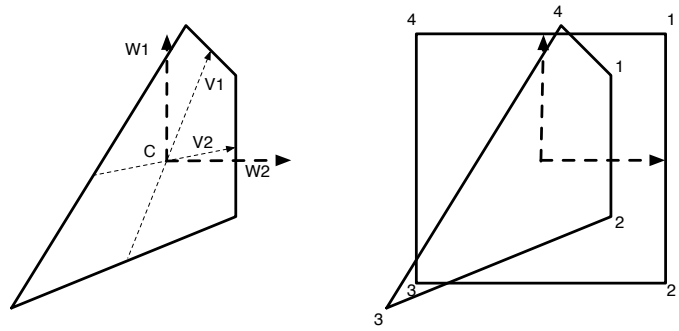

Fig. 14. (**left**) Finding the optimal orthogonal base vectors (in thick dotted lines); (**right**) Building a perfect square in these base.

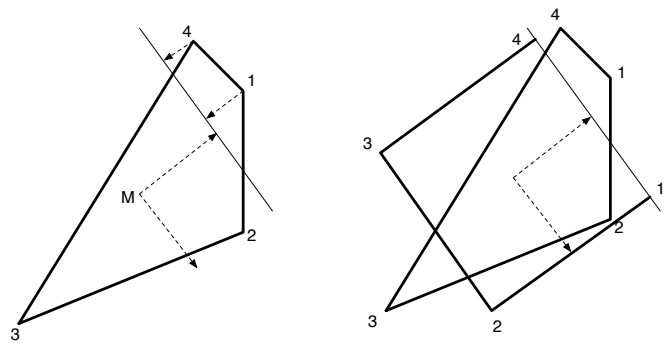

Fig. 15. The edge $1-4$ is to be projected on the geometry (thin line). Thus, one of the base vectors is set with the surface normal vector. Likewise, the origin of the new base is positioned so that the boundary edge of the optimal quad lies perfectly on the real surface (right).

9 Conclusion

Let's be honest, *Hexotic* is still far from the hex-meshing Holy Grail.

It is robust, fast (2 million elements per minute on a 2.4 ghz core2 duo) and generates all-hexahedral, conformal and valid meshes while capturing (not so) sharp angles greater than 30°.

It still behaves poorly with thin geometries (the octree generates too many elements) and cannot mesh accurately angles sharper than 30°. Thus it may not be considered as a true general purpose hex-mesher.

However, such a software is able to mesh many real life mechanical geometries as is shown in the last section examples.

Two directions are being investigated to overcome these limitations: using anisotropic smoothing to help handling thin geometries and the insertion of a limited number of tetrahedra and pyramids in very sharp angles.

10 Some Results

Here are some sample results, ranging from simple, but somewhat tricky, geometric pieces, to real life mechanical parts and adapted meshes. All tests have been run on a 2.8 ghz octo-core mac xserve. The smoothing step is multithreaded and takes advantage of all eight cores.

These stats are summed up in the following table:

Figure	Name	# of hexes	Meshing time	Hex quality	Min. q.
Fig. 17, top	nasty cheese	3,556,915	67.11 sec.	0.718	0.010
Fig. 17, bottom	TetTet7	32,995	0.78 sec.	0.735	0.013
Fig. 18, top	anc101_a1	166,280	3.67 sec.	0.620	0.004
Fig. 18, bottom	asm007	121,532	3.12 sec.	0.633	0.011
Fig. 19, top	bm1_90base	818,780	22.74 sec.	0.671	0.011
Fig. 19, bottom	gps25	1,427,535	24.37 sec.	0.721	0.010

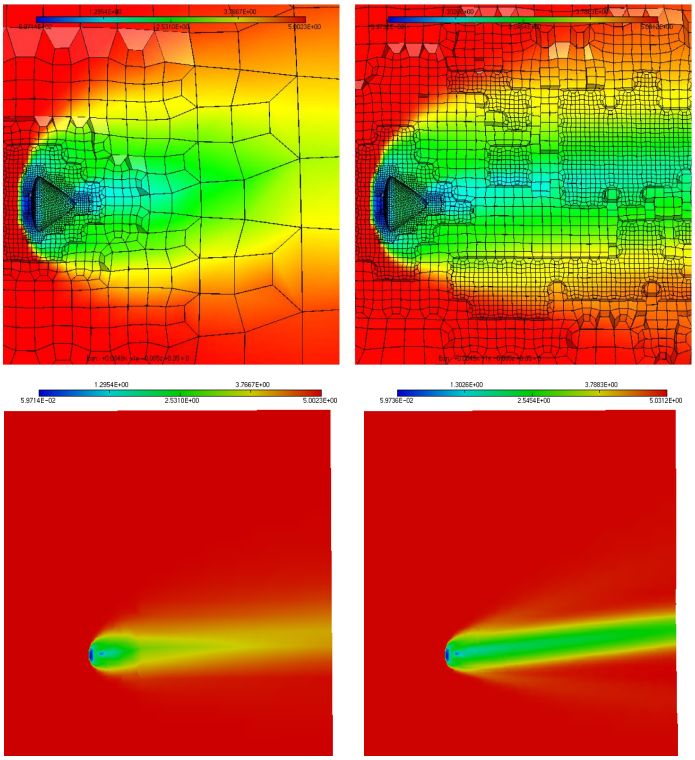

Fig. 16. Mesh adaptation scheme coupling a CFD simulation of an Apollo capsule entering atmosphere (Euler inviscid solver *Wolf* [1]) and *Hexotic*: (**top left**) cut through the initial mesh (66,000 elements); (**top right**) final adapted mesh (538,000 elements); (**bottom left**) initial pressure field; (**bottom right**) The final iteration pressure field is much better captured.

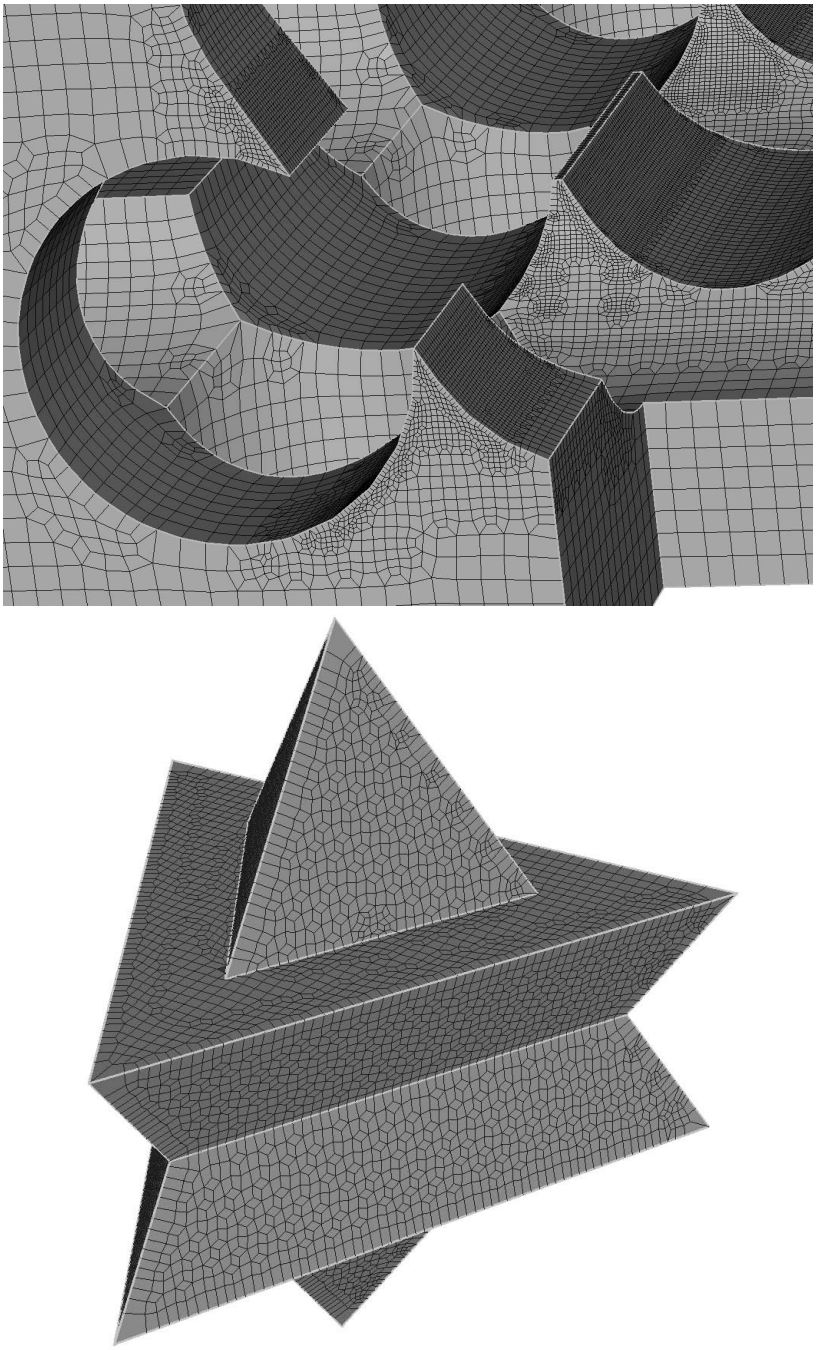

Fig. 17. Test case geometries. The upper picture is a close-up view of nasty cheese, a well known test-case featuring 30° dihedral angles.

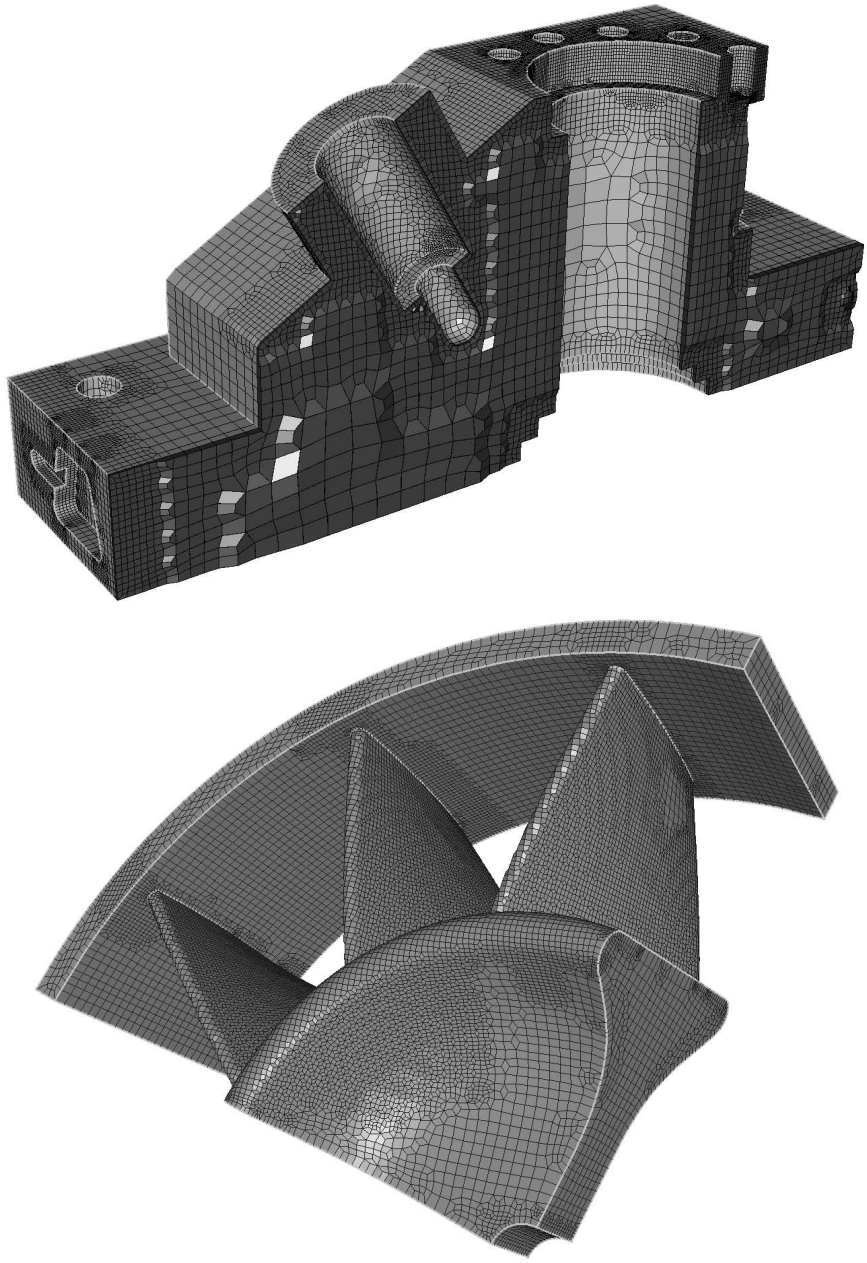

Fig. 18. More complex mechanical parts. Picture above shows a cut through the volume mesh (dark elements are hex faces).

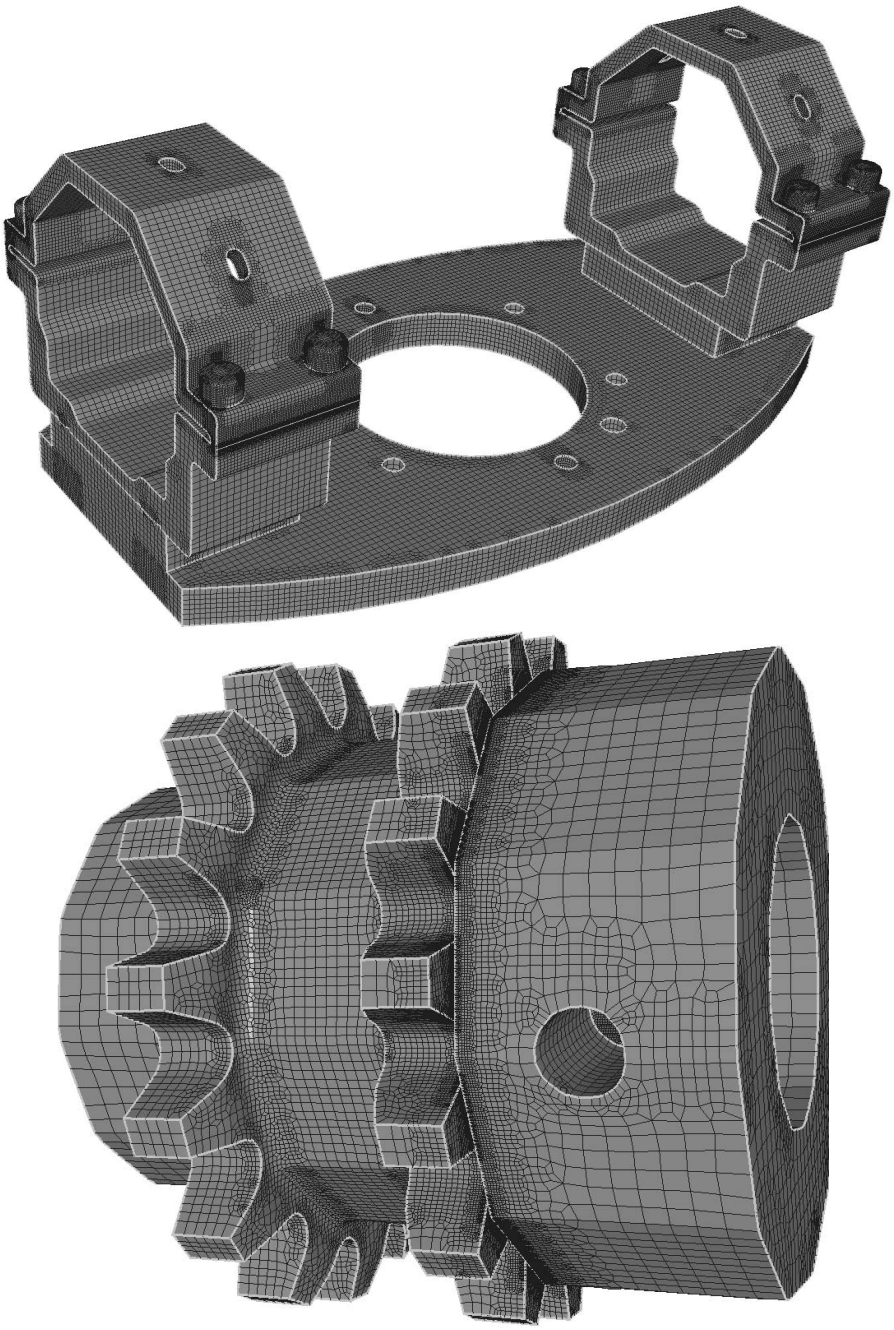

Fig. 19. (**top**) Small features capturing required 818,780 hexes; (**bottom**) A very fine gap between upper part and the top gear forced the octree up to level 9 subdivision. Thus generating 1,427,535 hexes.

References

1. Alauzet, F., Loseille, A.: High Order Sonic Boom Modeling by Adaptive Methods. INRIA Research Report RR-6845 (2009)
2. Armstrong, C.G., Li, T.S., Mckeag, R.M.: Hexahedral meshing using midpoint subdivision and integer programming. Comp. Meth. Appl. Mech. Engng. 124, 171–193 (1995)
3. Armstrong, C.G., Price, M.A.: Mat and associated technologies for structured meshing. ECCOMAS (2000)
4. Blacker, T.: Meeting the challenge for automated conformal hexahedral meshing. IMR 9, 11–19 (2000)
5. Calvo, N.A., Idelsohn, S.R.: All-hexahedral element meshing: Generation of the dual-mesh by recurrent subdivision. Comp. Meth. Appl. Mech. Engng. 182, 371–378 (2000)
6. Dhont, G.: A new automatic hexahedral mesher based on cutting. Int. Jour. Numer. Meth. Eng. 50, 2109–2126 (2001)
7. Folwell, N.T., Mitchell, S.A.: Reliable whisker weaving via curve contraction. IMR 7, 365–378 (1998)
8. Frey, P.J., Maréchal, L.: Fast adaptive quadtree mesh generation. IMR 7, 211–224 (1998)
9. Frey, P.J., George, P.L.: Mesh Generation, ch. 5. Hermes Science publishing (1999)
10. Harris, N.J., Benzley, S.E., Owen, S.J.: Conformal refinement of all-hexahedral element meshes based on multiple twist plane insertion. IMR 13 (2004)
11. Müller-Hannemann, M.: Hexahedral mesh generation by successive dual cycle elimination. IMR 7, 379–393 (1998)
12. Shepherd, J., Benzley, S., Mitchell, S.: Interval assignment for volumes with holes. Int. J. Numer. Meth. Engng. 49, 277–288 (2000)
13. Yerry, M.A., Shephard, M.S.: A modified-quadtree approach to finite element mesh generation. IEEE Computer Graphics Appl. 3, 39–46 (1983)
14. Shephard, M.S., Georges, M.K.: Automatic three-dimensional mesh generation by the finite octree technique. SCOREC Report (1) (1991)
15. Schneiders, R., Bünten, R.: Automatic generation of hexahedral finite element meshes. Computer Aided Geometric Design 12, 693–707 (1995)
16. Schneiders, R.: Octree-based hexahedral mesh generation. Int. J. of Comp. Geom. & Applications 10(4), 383–398 (2000)