Robust Boundary Layer Mesh Generation

Adrien Loseille[1] and Rainald Löhner[2]

[1] GAMMA Team, INRIA Paris-Rocquencourt, France
Adrien.Loseille@inria.fr
[2] CFD Center, George Mason University, USA
rlohner@gmu.edu

Summary. In this paper, we introduce a 3D local operator that automatically combines typical simpler operators as removal of vertices, collapse of edges or swap of faces and edges. This operator is inherited from incremental methods where the mesh $\mathcal{H}_k$ is modified iteratively through sequences of insertion of a point P:

$$\mathcal{H}_{k+1} = \mathcal{H}_k - \mathcal{C}_P + \mathcal{B}_P,$$

where $\mathcal{C}_P$ is the cavity of P and $\mathcal{B}_P$ the ball of P. We derive two algorithms to compute $\mathcal{C}_p$. The first algorithm is tuned to be a fast point reprojection to the geometry even in the presence of a boundary layer mesh. The second one is tuned to generate boundary layer meshes for complex geometries. We show how quasi-structured elements can be enforced. In addition, enhancements as multi-normals can be incorporated in the process. Both operators can be used with surface and volume point while preserving a given geometry. They rely on the use on an existing initial volume mesh and always produce a valid 3D mesh on output.

Keywords: Boundary layer mesh generation, mesh adaptation, incremental methods, surface projection, CAD.

1 Introduction

A boundary layer mesh is a quasi-structured layered mesh around a given geometry. It is usually considered as an extrusion of the initial surface along its normals [4, 11, 12, 21] or by local modifications of the mesh [17]. These meshes are widely used in Computational Fluid Dynamics (CFD) [23]. Indeed, most of commonly-used second-order unstructured CFD numerical schemes or commercial packages need a boundary layer mesh to accurately approximate the speed profile around a body during a viscous simulation, see the pioneering works [8, 11, 21]. The quality of the mesh is also crucial as it has a huge impact on the evaluation of functionals of interest used in industry

as lift or drag. This is especially true for unstructured flow solver [18], and thus for complex geometries for which defining a structured grid is no more tractable. Consequently, several severe difficulties need to be tackled to generate a boundary layer mesh that is well suited for numerical computations.

State-of-the-art and problematics

The first difficulty is to deal with the required very high aspect ratios of the elements $O(10^3 - 10^5)$ along with a high-fidelity geometry: landing gear, full aircraft, ..., see Figure 1 for some examples. Using the normals as sole extrusion information requires several enrichments to obtain a smooth layers transition on complex surfaces [1, 6, 9, 10]. These techniques rely on a complex pre-processing of the surface in order to extract additional geometric information as convex or concave ridges. For instance, it may be required to handle normal deactivation in areas where the level of details is smaller than the size of the first layer or to add several normals to increase the quality of the boundary layer mesh, see Figure 2. In our approach, we use the optimal normal computation of [2]. We replace the classical laplacian smoothing of normals by defining a normal merging. In addition, the normals are not computed once for all layers, but are reevaluated after each layer insertion.

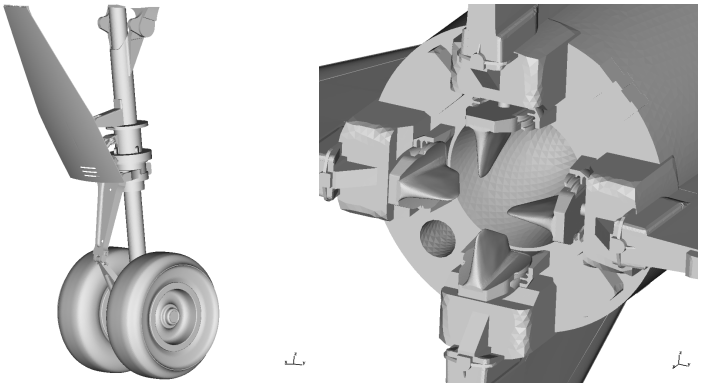

Fig. 1. Examples of complex geometries, a front landing gear (left) and the back of a missile (right). These geometries have a level of details that makes more complex the generation of a boundary layer mesh.

The second difficulty arises once the boundary layer extrusion process is finished. Indeed, in most methods, a global mesh generation process is used to close the volume. Both constrained Delaunay-based methods [7] or frontal methods [13] are susceptible to fail in presence of anisotropic or bad-shaped faces. We remove this weakness by generating the boundary layer mesh with local mesh modification operators. Consequently, we can guarantee that the

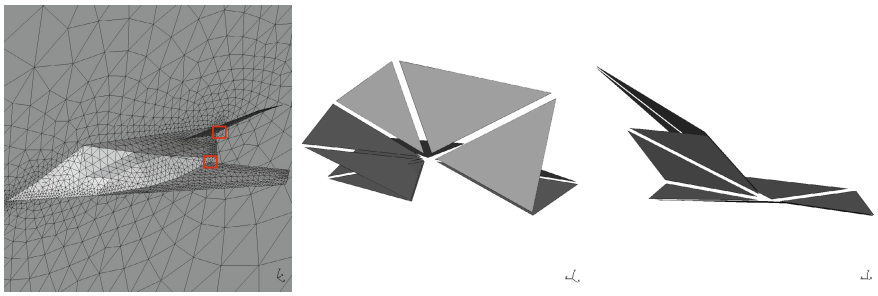

Fig. 2. F117 geometry (left) and closer view around two complex corners where it is impossible to find a unique normal seen by all the faces surrounding the corner (right)

process cannot fail, so that a computation is always possible even though the generated mesh may be sub-optimal.

Finally, in the aforementioned methods, the boundary layer mesh generation is thought as a unique pre-processing procedure that is done prior to any computations. In particular, these methods are not sought to work in an adaptive context where the (surface) mesh is modified iteratively during the adaptation. When an adaptation is performed, simpler strategies have been devised as keeping the boundary layer mesh unchanged [20]. However, in many applications, adapting the (surface and volume) mesh seems mandatory to get the full picture of the physics as anisotropic phenomena (shocks, contact discontinuity, ...) interact with the boundary layer, see Figure 3. In addition, asymptotic mesh convergence (one of the criteria to assess a numerical computation) is rarely observed on complex geometries with unstructured tailored meshes [22]. Consequently, achieving a full coupling between boundary layer mesh generation and mesh adaptation is tedious and remains a challenge. We propose a first improvement in this context by defining a fast point re-projection even if the current mesh has a very thin boundary layer mesh. This operation is mandatory to adapt the surface mesh especially in strategies that keep a valid volume mesh [14, 15, 19]. In addition, the local operator used to generate the boundary layer may be used on anisotropic surface. By using local modification operators, we also remove the need of calling a global mesh generator during the adaptive process.

Outline

We first end this introduction with the nomenclature and definitions used throughout the paper. In Section 2, we introduce a point re-inserter procedure that can be used to project a point back to a given geometry or to generate a mono-normal boundary layer mesh. In Section 3, we show how this operator is modified to allow to generate quasi-structured boundary layer meshes. The

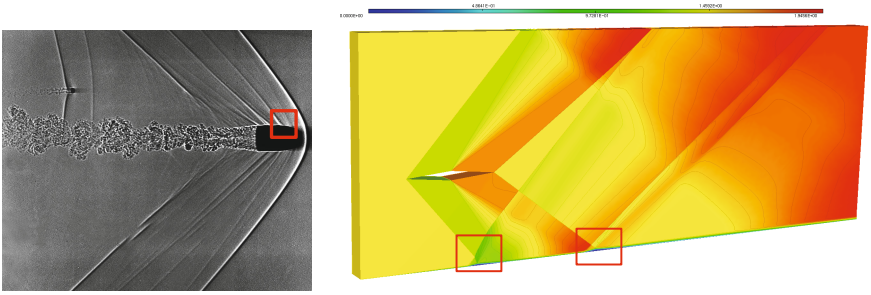

Fig. 3. A supersonic viscous bullet (left) from NASA media library and adaptive shock/boundary layer interaction (rigth) excerpt from [16]. Both examples contain interaction between the viscous boundary layer and anisotropic phenomena (red squares).

procedure to generate one layer (choice of normals, multi-normals, ...) is also recalled. We validate this approach on several complex geometries.

Nomenclature and definitions

In what follows, $\mathcal{H}$ is a 3D simplicial mesh, *i.e.*, composed only of tetrahedra and triangles. $\mathcal{H}$ is **valid** if all tetrahedra have a positive volume. If an hybrid mesh is given as input, it is pre-processed to be decomposed into a simplicial mesh. We follow the procedure given in [5].

Capital letters as A, B, ..., P denote points of $\mathbb{R}^3$, except K that usually denotes an element (triangle or tetrahedron). $\mathcal{B}_P$ is the ball of P, *i.e.,* the list of elements surrounding P (having P as vertex), $\mathcal{S}_{AB}$ is the shell of edge AB, *i.e.,* the list of elements having A and B as vertices. Topological entities as balls and shells are computed on the fly by using the elements surrounding elements storage data structure. Consequently, we assume that for each tetrahedron, we know the neighboring tetrahedra (or boundary faces) seen across the 4-faces. We refer to [7] for more details.

$\mathcal{C}_p$ is the cavity of P. This notion is usually related to Delaunay mesh generation. We use this nomenclature to keep the analogy with incremental method and Delaunay point insertion. However, in this paper, the cavity is just a set of tetrahedra and is not related to any Delaunay criteria. Instead, we assume that $\mathcal{C}_p$ has the following property:

$$\begin{gathered}\text{Given any two tetrahedra } K_1 \text{ and} K_2 \text{ in } \mathcal{C}_p,\\ \text{there exists a path trough faces' of tetrahedra in } \mathcal{C}_p\\ \text{that links } K_1 \text{ and } K_2.\end{gathered} \tag{1}$$

Note that this property holds for the balls of vertices and the shells of edges. We denote by $\overline{\mathcal{C}}_P$ the external faces of $\mathcal{C}_P$. It is composed of boundary faces

(triangles) and internal faces. For each internal face $\overline{\mathcal{C}}_P$, the neighboring tetrahedron viewed from this face is not contained in $\mathcal{C}_P$.

Given an oriented face $[A, B, C]$ and a normal $\mathbf{n}$, the visibility of $\mathbf{n}$ with respect to $[A, B, C]$ is the dot product between $\mathbf{n}$ and the normal to the face. In a similar way, a point P is visible for an oriented face $[A, B, C]$ if the volume of the tetrahedron $[P, A, B, C]$ is positive.

We now state the fundamental property: Given a **valid** mesh $\mathcal{H}$, a point P and a set $\mathcal{C}_P$, if (3) holds for $\mathcal{C}_P$ and P is visible for all faces of $\overline{\mathcal{C}_P}$ then the mesh given by $\mathcal{H} - \mathcal{C}_P + \mathcal{B}_P$ is **valid**. The local mesh modification operators derived in this paper are based on this property.

2 Generalized Point Reinsertion

In any adaptive strategies, one crucial point is the projection of the adapted surface mesh on the geometry. This is usually done as a post-processing where the surface points are moved iteratively into the volume mesh [14, 19]. This requires to solve additional problems (as elasticity) or to use post-processing methods (as radial basis functions) in order to propagate the displacement of the surface points into the volume [3]. However, if a boundary layer mesh is initially attached to an adapted surface mesh, the problem of point relocation becomes more complicated. We propose in this section a local operator allowing to move a point to a given new position. This operator can be easily coupled with an adaptive algorithm to adapt the surface mesh even in the presence of a boundary layer mesh.

Point reinsertion algorithm

Given a mesh $\mathcal{H}_k$, a point P (P_{new} being its new position), the algorithm consists in reinserting P following:

$$\mathcal{H}_{k+1} = \mathcal{H}_k - \mathcal{C}_P + \mathcal{B}_P, \tag{2}$$

where P is now in P_{new} in $\mathcal{H}_{k+1}$. The construction of $\mathcal{C}_P$ should guarantee that for any internal face [A,B,C] in $\overline{\mathcal{C}}_P$, the volume of $[P_{new}, A, B, C]$ is positive. Then, for the remaining boundary faces (triangles) in $\overline{\mathcal{C}}_P$, we recover all the connex components (by surface patches). Each set is either of type (i) constrained: the faces of this set must appear in $\mathcal{B}_P$ (for instance the boundary faces that belong to the surface being extruded), (ii) faces are kept (not connected to P) or (iii) faces are connected to P (plane of symmetry). Sets of type (i) and (ii) must create valid tetrahedra whereas a boundary edge [A,B] of a face in set (i) is triangulated to create a new face $[P_{new}, A, B]$, see illustration in Figure 4. By construction, $\mathcal{C}_P$ built from Algorithm 1 verifies (3). If this algorithm is straightforward for volume point, more care is needed when inserting a surface, line or corner point. Indeed, if for new tetrahedra the volume is the only quantity used to verify that the final set $\mathcal{B}_P$ is valid,

Algorithm 1. Cavity enlargement for P and new position P_{new}

Initialize the cavity with the ball of P: $\mathcal{C}_P = \mathcal{B}_P$

Volume Part:
For each K in $\mathcal{C}_P$

  For each internal face $[A, B, C]$ of K such that $P \notin [A, B, C]$:
    if $[A, B, C]$ is a boundary face then update $\overline{\mathcal{C}}_P$
    else if volume of $[P_{new}, A, B, C] \leq 0$, add neighboring tetrahedron to $\mathcal{C}_P$
    endif
  EndFor

EndFor

Surface Part:
Create the sets of connex components of the boundary face of $\overline{\mathcal{C}}_P$
Set type (i), (ii) or (iii) for each component
Check that all faces of type (i) or (ii) produce a valid tetrahedron, otherwise **exit**
For each component Cmp of type (iii)

  For each triangle K in Cmp
    For each edge [A,B] of K with $P \notin [A, B]$
      if [A,B] is a boundary edge of Cmp then test triangle $[P_{new}, A, B]$
        if the triangle is not valid then add shell $\mathcal{S}_{AB}$ to $\mathcal{C}_P$
        endif
      endif
  EndFor

EndFor
if $\mathcal{C}_P$ is modified goto **Volume Part**.

tests are more complex for new triangles. We verify the topology (to avoid multi-connected edges) and also the minimal angle in the triangles in order to increase the visibility of the boundary edges of $\overline{\mathcal{C}}_P$. If Algorithm 1 exits on a constrained face, a Steiner point is added in order to remove this face from $\overline{\mathcal{C}}_P$. Most of the time, it is sufficient to insert a volume point $P_{stei} = P + \alpha PP_{new}$, with α sufficiently large to make sure that the faces that trigger the exit become visible from P_{stei}. To complete the reinsertion of P, the new surface and volume elements are created simply by triangulating each boundary face or boundary edge with P_{new}.

As an academic example, we show a simple wedge geometry where the bottom surface is moved inward and outward to the volume mesh, see Figure 5. It illustrates that with only one call for each point, several layers of vertices and elements are automatically removed (without any preprocessing steps required). We now apply this algorithm to our problem at hand: the creation of a boundary layer mesh.

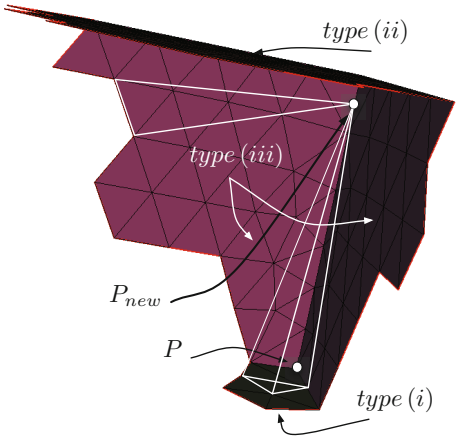

Fig. 4. An example of the boundary faces of $\overline{\mathcal{C}}_P$ for P with new position P_{new}. Each color defines a connex component. Depending on the type, either a tetrahedron is generated or a triangle, see white entities.

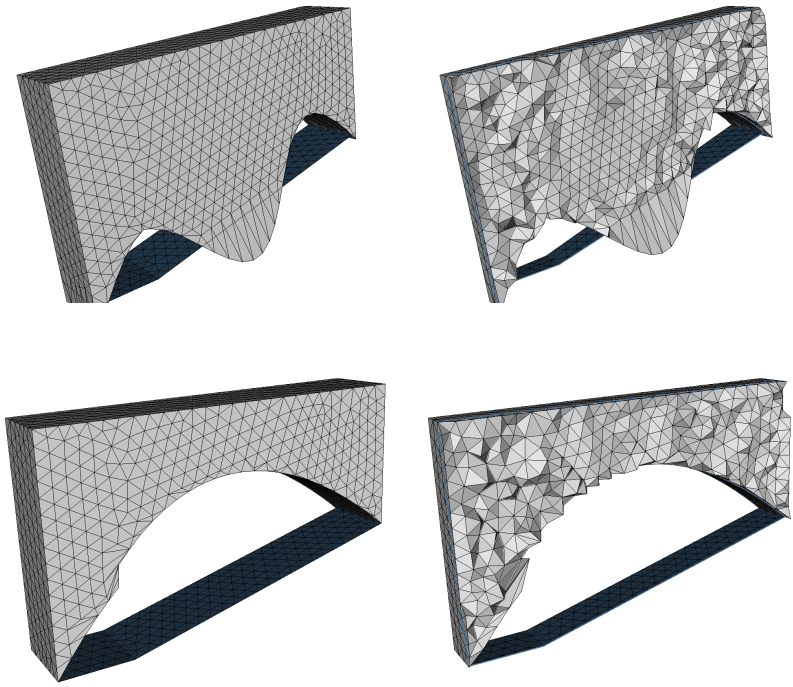

Fig. 5. Example of a wedge geometry where the bottom surface (in blue) is moved along the normals according to a sine (top) and parabola functions (bottom). In one shot, large areas of the volume mesh are automatically removed.

Application to mono-normal boundary layer mesh generation

Note that the previous operator can be used to generate a boundary layer mesh. Indeed, the initial boundary front is moved iteratively to the position of the next layer. The boundary layer mesh is then stored separately from the remaining volume mesh. Each face of the initial front engenders either a prism (if all the points of the face are moved), or a pyramid (if only two points of the face are moved), or a tetrahedron (if only one point of the face is moved). The final mesh is obtained by gathering these two parts. If a simplicial mesh is required for computations, we decompose the hybrid entities into tetrahedra [5]. Note that the front is moved (extruded) along one (normal) direction. Consequently, several passes of laplacian smoothing are done on these directions in order to improve the quality of the layers at opened ridges (along the trailing edge for instance). This strategy is equivalent to moving mesh methods used to generate boundary layer meshes as in [3, 4]. However, in our approach, we do not need to solve a global PDE (elasticity, ...) to propagate the boundary displacement into the volume. The necessary

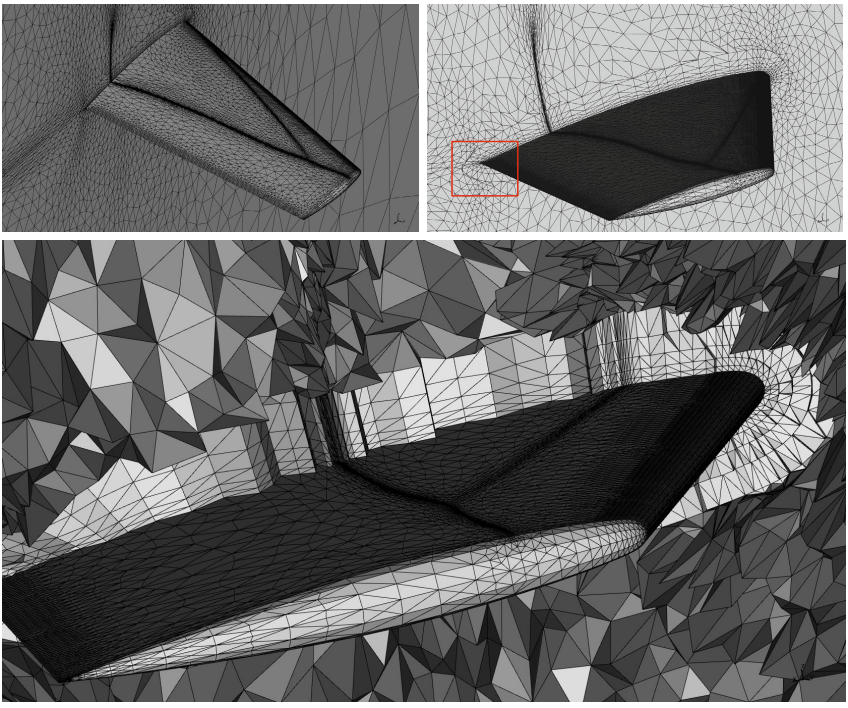

Fig. 6. Example of a boundary layer from an anisotropic m6-wing geometry. Even in the presence of high anisotropy, the local operator succeeds to move the front point. The quality of the layers is decreasing at open ridges despite the use of a normal smoothing (top right).

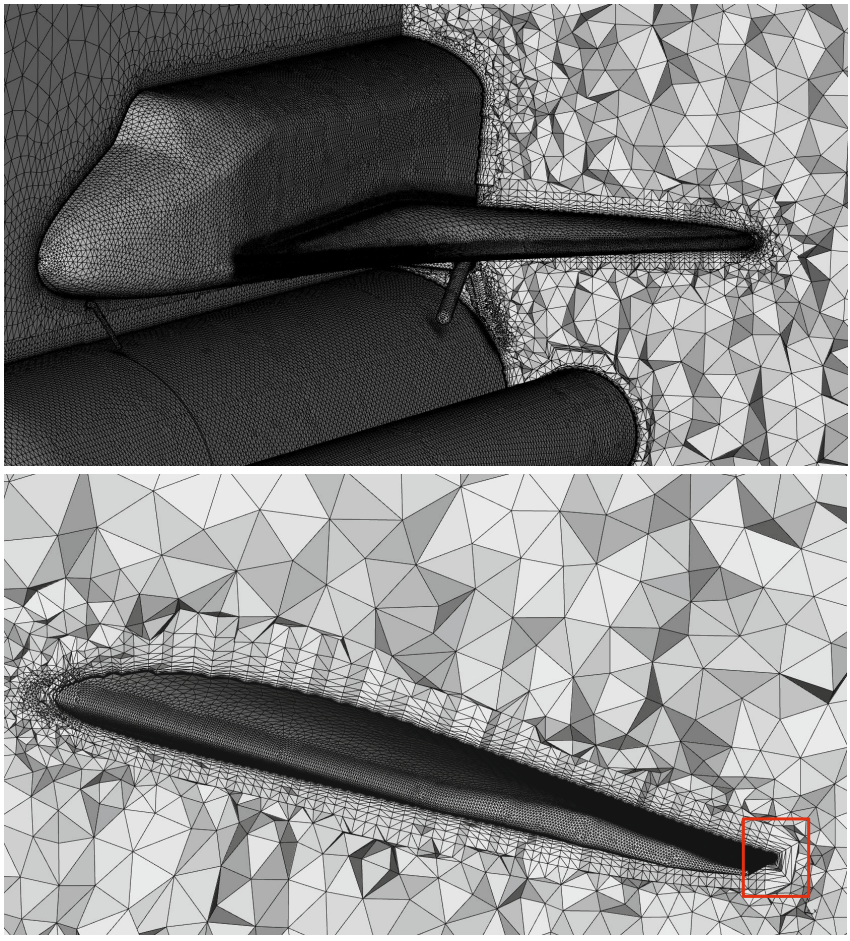

Fig. 7. Example of a boundary layer from a shuttle geometry based on the displacement of a front layer along one normal. The quality of the layers is decreasing at opened ridges despite the use of a normal smoothing (bottom).

room to move the point is created automatically by the local operator. We give two examples of boundary layer mesh generation applied to an initial anisotropic m6 wing and to a shuttle, see Figures 6 and 7. Despite the high-level of anisotropy (on the surface and volume mesh), the reinserter succeeds to move the front. For each case, we emphasize typical areas where multi-normals seems mandatory to improve the overall quality of the boundary layer mesh, see the red squares in Figures 6 and 7. The CPU time for each case is less that 1 min on a macbook pro laptop equipped with an i7 core at 2.66Ghz with 8 Gb of Ram.

The main drawback of this approach is to handle two separate meshes. Adding multi-normals enhancement for instance is also tedious with this

approach. Indeed, as soon as the surface front is modified, the matching with the two meshes is not trivial anymore. Consequently, we prefer to define a new local operator that has much more flexibility and that removes the need to store two meshes.

3 Boundary Layer Mesh Generation

We consider in this Section the generation of a boundary layer mesh starting from a given surface mesh. We focus only on the generation of the volume mesh meaning that the surface mesh that supports the boundary layer is kept constant during the whole process. No hypothesis is made on the features of the surface mesh. In particular, the process should handle anisotropic surface meshes.

We first modify the previous reinserter to become an inserter. We then exemplify specific choices of $\mathcal{C}_P$ leading to the generation of quasi-structured elements. The main improvement from the method described in Section 2 is to avoid to handle two distinct meshes while offering more flexibility in the generation of the boundary layer.

Boundary layer mesh generation by point insertion

Previous inserter is now turned into a constrained point inserter. We detail the main modifications.

When P_{new} is inserted along a normal, the previously created elements that are in the boundary layer mesh should be kept. Consequently, a set $\mathcal{K}$ of (constrained) tetrahedra in the boundary layer is created and updated after each insertion. The cavity enlargement procedure of Algorithm 1 can exit if during the process a constrained element is added to $\mathcal{C}_P$. The cavity initialization is also modified to remove from $\mathcal{B}_P$ elements that belong to $\mathcal{K}$. In order to ensure (3), it is necessary to verify that $\mathcal{B}_P$ minus the constrained elements is still connex. From a technical point of view, as P_{new} is not initially (topologically) present in the mesh (contrary to the case where P is moved), the main difficulty is related to the surface part of Algorithm 1. Indeed, every boundary face [A,B,C] will never contain P_{new}, so additional information is required to derive each connex component type. We can now state the main result:

$$\text{In a mono-normal context, previous inserter with initialization } \mathcal{C}_p = \mathcal{B}_P - \mathcal{K}, \text{ automatically generates quasi-structured elements.} \tag{3}$$

We first remark that if P_{new} is inserted along a normal direction issued from P, the final mesh will contain the edge PP_{new}. Then if face $[A, B, C]$ belongs to $\overline{\mathcal{C}}_P$, then tetrahedron $[A, B, C, P_{new}]$ is created. Consequently, given a face $[A, B, C]$ with the extruded vertices $[A_{new}, B_{new}, C_{new}]$, insertion of A_{new} will create tetrahedron $K_1 = [A, B, C, A_{new}]$, insertion of

B_{new} will create $K_2 = [A_{new}, B_{new}, B, C]$ and insertion of C_{new} will create $K_3 = [C_{new}, A_{new}, B_{new}, C]$. The union of K_1, K_2 and K_3 forms the prism $[A, B, C, A_{new}, B_{new}, C_{new}]$. Note that K_1, K_2 and K_3 should be added to $\mathcal{K}$ after each insertion. The update of $\mathcal{K}$ is based on the different sets of hybrid elements than can be created, see Figure 8. We illustrate in Figure 9, different prisms construction around the ball of a point. Note that changing the order of insertion of the extruded points will lead to different decompositions of the prismatic mesh. Consequently, the point insertion can be *a priori* optimized in order to favor the creation of the smallest diagonal edges when a quadrilateral face of a prism is decomposed. In addition, according to the current configuration, the remaining points cannot be inserted in any order as this may lead to an invalid decomposition (known as the Schönhardt's prism). It is thus necessary to loop over the remaining points in order to solve this issue. Usually, no more than 10 iterations are required to insert all the points.

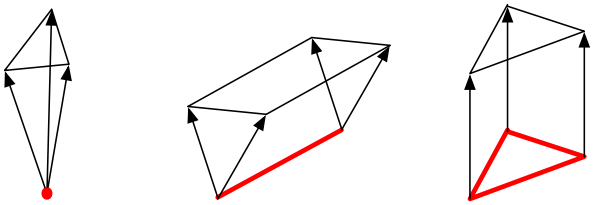

Fig. 8. Depending on the configuration of normals, different kinds of prisms are recovered: vertex-based (left), edge-based (middle) and face-based (right)

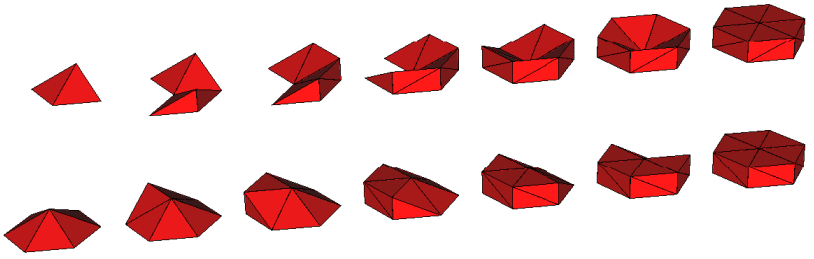

Fig. 9. Tow examples of the automatic process of prisms creation around a point. The final decomposition of prisms (in tetrahedra) depends on the order of the insertion of points.

Possible enhancements with multi-normals and merge

Two main drawbacks of using only one normal per point arise at closed and opened ridges. At closed ridges, the normals may cross, leading to invalid elements. At opened ridges, the quality of the elements decreases as the

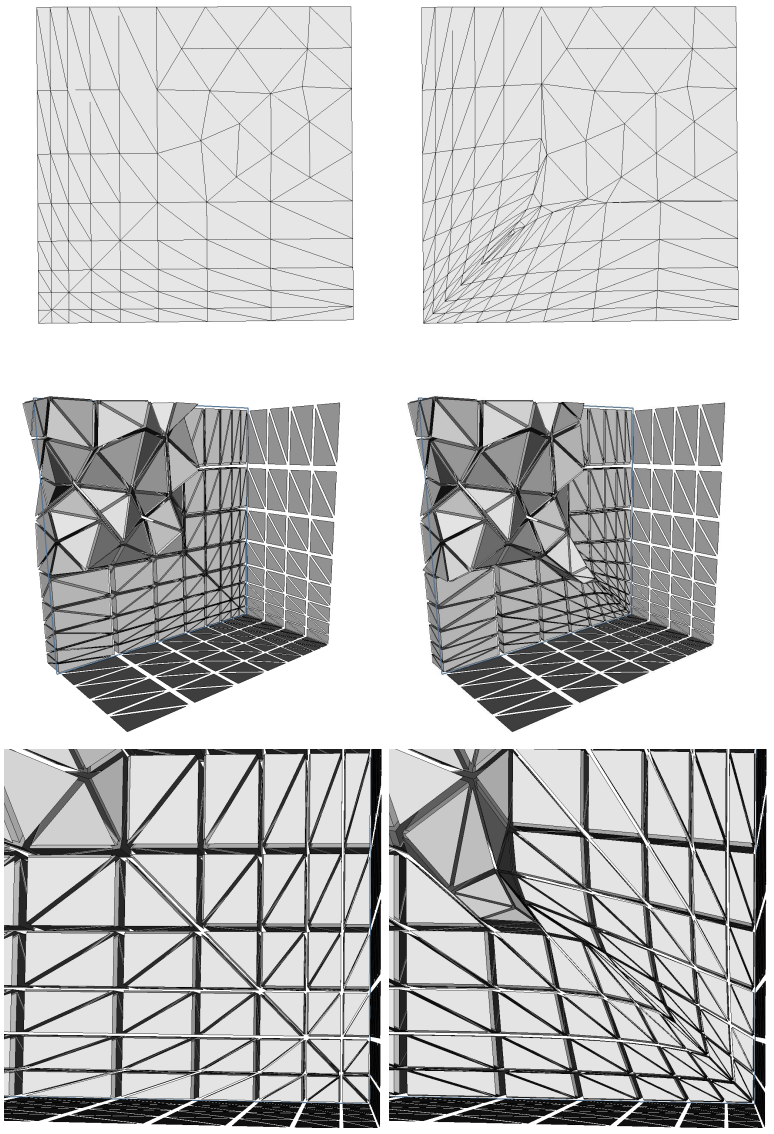

Fig. 10. Example of extruding a boundary layer from a closed ridge. The merge of normals (left) allows to create regular elements (hexahedra) whereas smoothing of normals is required (right) to avoid normals crossings. From top to bottom, trace of the boundary layer mesh in the plane of symmetry, two cuts in the volume meshes.

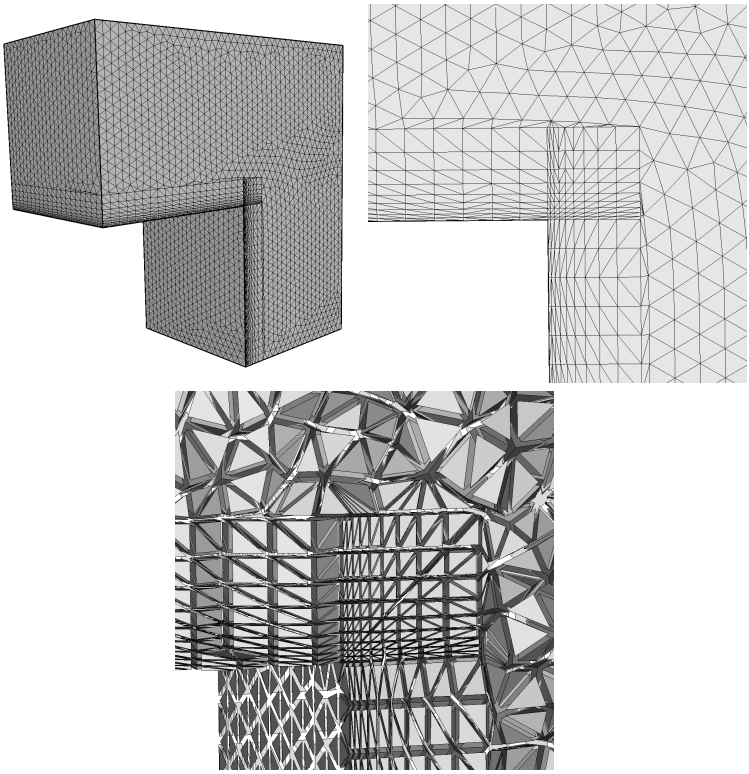

Fig. 11. Example of extruding a boundary layer from an opened ridge. The use of multi-normals allows to create regular elements (hexahedra): global and closer views around the ridge of the surface mesh (top left and right), view in the volume mesh (bottom).

deviation between the normals may be large. A common practice to solve this issue is to smooth the normals [1]. In our approach, we can choose different initialization of $\mathcal{C}_P$ in order to improve the boundary layer mesh quality.

In order to reduce the number of invalid elements, we pre-process the given normals and current size and predict the volume of elements. For each invalid element found, we attempt to collapse the normals until no more negative element is found. For a face, if three normals are merged, a tetrahedron is created, whereas a pyramid is created when two normals are merged. Normals are merged according to a minimal distance criterion. If we denote by $(P_i)_{i=1,k}$ the list of points associated with a given list of merged normals, the cavity is then initialized by $\bigcup \mathcal{B}_{P_i} - \mathcal{K}$. The standard inserter is then called with this initialization. We illustrate this operator on a simple cube geometry where 2 faces support the boundary layer mesh. Theses surfaces are adapted

to follow the normal size distribution. Consequently, without normal smoothing, normals cross each other at each step. We can see that the quality of the generated boundary layer mesh is improved with the merge of normals, see Figure 10.

The multi-normal case is the most complicated. The way the cavity $\mathcal{C}_P$ is initialized is crucial to ensure the desired connectivity. Given P and a (minimal) set of normals, we start from the list of the (interface) faces that surround P. Interface faces are the boundary triangles for the first layer and then become the internal faces defining the frontier between the previous and current layers. Note that this faces are part of $\overline{\mathcal{C}}_P$. We then assign each face to a normal by trying to maximize the visibility. In addition, for a given normal, the list of faces should be adjacent by edges. If two many normals are given, remaining normals with no more faces are not inserted. For a normal $\mathbf{n}$ associated to the list of face $L = (F_i)_i$, the cavity is initialized by $\mathcal{C}_P = \bigcup_{AB \in F_i, F_j} \mathcal{S}_{AB}$, where AB is an edge shared by two faces of L. The inserter is then called normally with this initial choice of $\mathcal{C}_P$. We illustrate this procedure on a simple opened ridge. Fully structured elements are automatically created as in the merged case, see Figure 11.

The main difficulty when using merged and multi-normals consists in the recovery of constrained elements in order to update $\mathcal{K}$. In the mono-normal context, only face prisms are recovered whereas in this case, constrained elements can be also point-based and edge-based, see Figure 8. Algorithm 2 summarizes the complete process.

Remark 1. Currently, the mono-normals are kept, a better option not yet tested may be to create new optimal normals based on the new set of boundary faces and not just on the initial faces.

Remark 2. In the case of closed ridges, it is necessary to pre-adapt the surface mesh in order to generate perfectly matching layers contrary to opened ridges.

Algorithm 2. Boundary layer mesh generation

While (1)

1. Recover the interface surface mesh (between current layer and previous layer)
2. Compute normals and multi-normals
3. Fictive extrusion of the boundary-layer : optimize the layer with the merge of normals
4. Insert extruded points in the following order:
   - along merged normals
   - along multi-normals
   - along mono-normals to close the boundary layer volume
5. Optimize the current layer: diagonal swapping and point smoothing

EndWhile

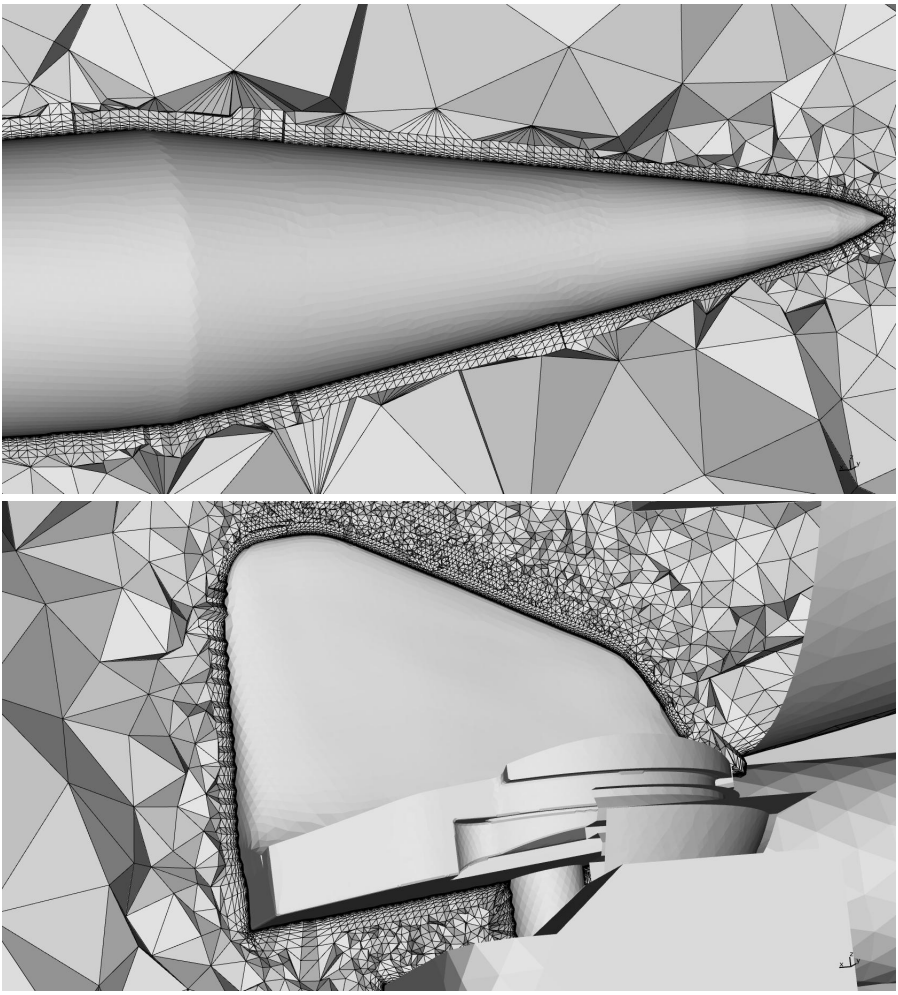

Fig. 12. Boundary layer mesh generation around a complex missile geometry starting from a coarse initial volume mesh

Examples

We exemplify the use of this operator on different complex geometries: an ONERA m6 wing, a shuttle, a missile and a landing gear, see Figure 1. The previous examples (m6 wing or the shuttle) give the same results as in Section 2 but with a favorable impact on the CPU time. For the anisotropic m6 wing, the CPU time to insert a layer is less than 1 sec (with the creation of 40 000 prisms / layer). In the case of the shuttle, CPU time to insert a layer is around 2 sec (120 000 prisms / layer). The CPU time for the missile to insert a layer is around 10 sec (600 000 prisms / layer), see Figures 12

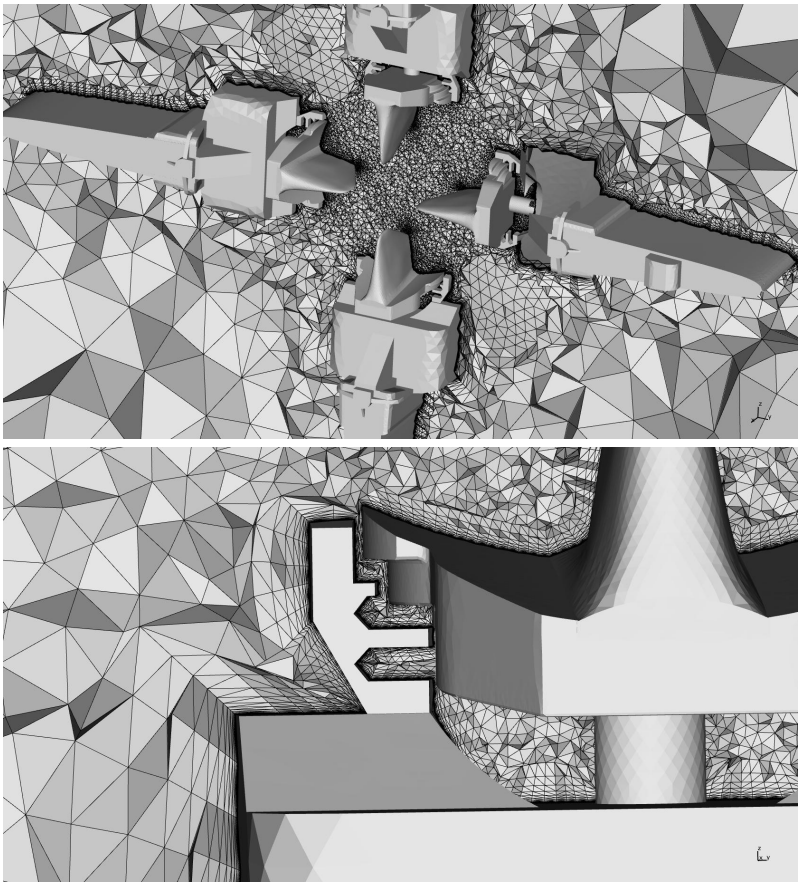

Fig. 13. Boundary layer mesh generation around a complex missile geometry

and 13. For the landing gear geometry, the CPU time to insert a layer spans from 8 sec to 35 sec when all the points are inserted (e.g. 800 000 prisms / layer), see Figures 14 and 15. For 25 layers, the total CPU time is around 7 min. Note that the remaining volume mesh between the boundary layer mesh and the outer surface of the domain is not optimized neither in quality nor in size. This is done in a different optimization step allowing to adapt the mesh with respect to a given anisotropic metric field or to a uniformly graded metric. Consequently, we can see that the process can insert a boundary layer mesh in an already high density mesh, see Figure 14, but also in a very coarse mesh, see Figure 12.

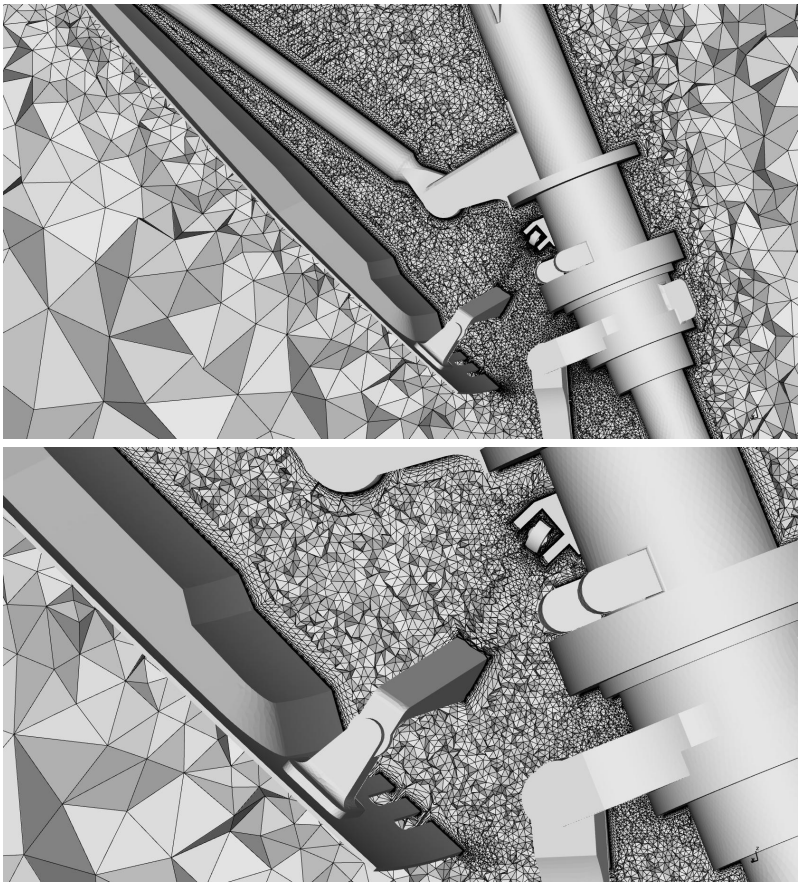

Fig. 14. Boundary layer mesh generation around a landing gear geometry

4 Conclusion

A local point reinsertion operator is introduced. It can be used to project surface points back to the geometry and to generate mono-normal boundary layer mesh. It is based on simple topological principles with iterative checks on the validity of the final surface and volume mesh. This operator is then slightly modified in order to favor the creation of quasi-structured elements. More insight into the initialization of $\mathcal{C}_p$ allows to insert multi-normals and merged normals. This has a favorable impact on the overall quality of the boundary layer grids. The generation of the boundary layer mesh does not require the use of a global mesh generator to close the volume mesh. In addition, there is not check of face-face intersections as usually used when the boundary layer is extruded in an empty space. These features increase the robustness of the method. Finally, the operator keeps its properties and robustness even in the

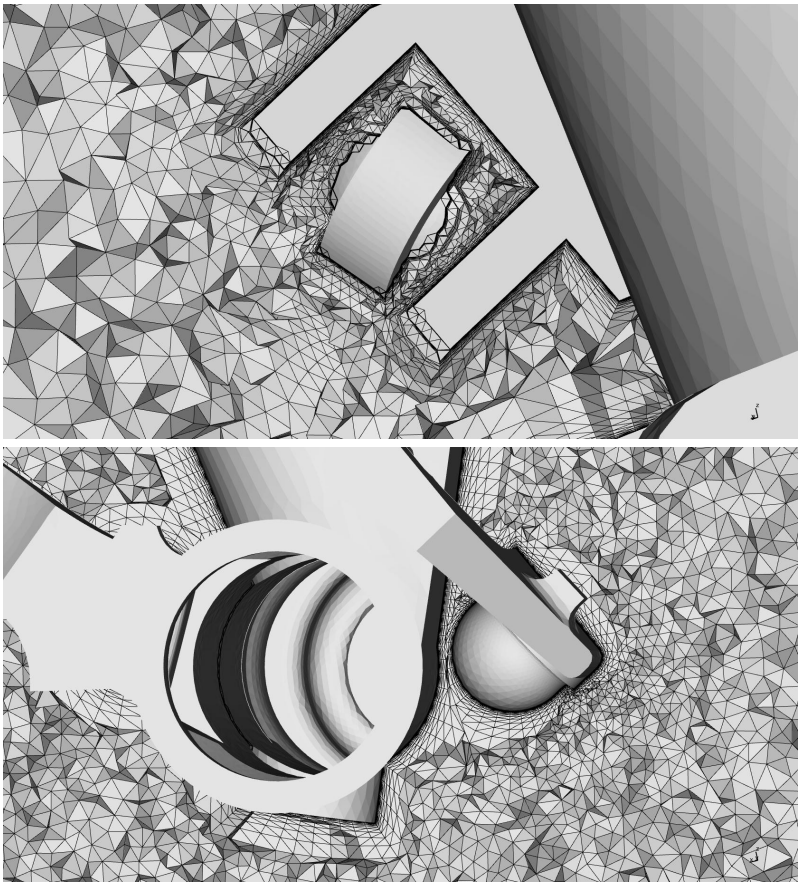

Fig. 15. Boundary layer mesh generation around a landing gear geometry

presence of highly stretched elements in the initial mesh. This point is a necessary property to consider adaptive viscous simulations.

The current work is directed at improving the robustness for the multi-normals and merged of normals. We also seek specific surface mesh adaptation. We also work on improving the transition from the boundary layer mesh to the volume mesh. The final intent is to derive an anisotropic mesh adaptation procedure to accurately capture viscous and anisotropic phenomena.

References

1. Aubry, R., Löhner, R.: Generation of viscous grids with ridges and corners. AIAA Paper, 2007–3832 (2007)
2. Aubry, R., Löhner, R.: On the most normal normal. Communications in Numerical Methods in Engineering 24(12), 1641–1652 (2008)

3. Baker, T., Cavallo, P.: Dynamic adaptation for deforming tetrahedral meshes. AIAA Journal 19, 2699–3253 (1999)
4. Bottasso, C.L., Detomi, D.: A procedure for tetrahedral boundary layer mesh generation. Engineering Computations 18, 66–79 (2002)
5. Dompierre, J., Labbé, P., Vallet, M.-G., Camarero, R.: How to subdivide pyramids, prisms, and hexahedra into tetrahedra. In: Proc. of 8th Meshing Rountable, pp. 195–204 (1999)
6. Garimella, R.V., Shephard, M.S.: Boundary layer mesh generation for viscous flow simulations. Int. J. Numer. Meth. Fluids 49, 193–218 (2000)
7. George, P.-L., Borouchaki, H.: Delaunay triangulation and meshing: application to finite elements. Hermès Science, Paris (1998)
8. Hassan, O., Morgan, K., Probert, E.J., Peraire, J.: Unstructured tetrahedral mesh generation for three-dimensional viscous flows. Int. J. Numer. Meth. Egrg. 39(4), 549–567 (1996)
9. Ito, Y., Nakahashi, K.: Unstructured mesh generation for viscous flow computations. In: Proc. of 11th Meshing Rountable. Springer (2002)
10. Ito, Y., Nakahashi, K.: An approach to generate high quality unstructured hybrid meshes. AIAA Paper, 2006–0530 (2006)
11. Löhner, R.: Matching semi-structured and unstructured grids for Navier-Stokes calculations. AIAA Paper, 1993–3348 (1993)
12. Löhner, R.: Generation of unstructured grids suitable for RANS calculations. AIAA Paper, 1999–1662 (1999)
13. Löhner, R., Parikh, P.: Three-dimensionnal grid generation by the advancing-front method. Int. J. Numer. Meth. Fluids 8(8), 1135–1149 (1988)
14. Loseille, A., Löhner, R.: On 3d anisotropic local remeshing for surface, volume, and boundary layers. In: Proc. of 18th Meshing Rountable, pp. 611–630. Springer (2009)
15. Loseille, A., Löhner, R.: Adaptive anisotropic simulations in aerodynamics. AIAA Paper, 2010–2169 (2010)
16. Loseille, A., Löhner, R.: Boundary layer mesh generation and adaptivity. AIAA Paper, 2011–2894 (2011)
17. Marcum, D.L.: Adaptive unstructured grid generation for viscous flow applications. AIAA Journal 34(8), 2440–2443 (1996)
18. Mavriplis, D.J.: Results from the 3rd Drag Prediction Workshop using NSU3D unstructured mesh solver. In AIAA Paper, volume 2007-0256 (2007)
19. Michal, T., Krakos, J.: Anisotropic mesh adaptation through edge primitive operations. AIAA Paper, 2011–2159 (2011)
20. Park, M.A., Carlson, J.R.: Turbulent output-based anisotropic adaptation. AIAA Paper, 2011–2168 (2011)
21. Pirzadeh, S.: Viscous unstructured three dimensional grids by the advancing-layers method. AIAA Paper, 1994–1417 (1994)
22. Rumsey, C.L., Long, M., Stuever, R.A., Wayman, T.R.: Summary of the first aiaa cfd high lift prediction workshop. AIAA Paper, 2011–2939 (2011)
23. Vassberg, J.C., DeHaan, M., Sclafani, T.: Grid generation requirements for accurate drag predictions based on OVERFLOW calculations. AIAA Paper, 2003–4124 (2003)