This is the accepted version of a paper published in *Computer-Aided Design*. This paper has been peer-reviewed but does not include the final publisher proof-corrections or journal pagination.

Citation for the original published paper (version of record):

Eller, D., Tomac, M. [Year unknown!]
Implementation and evaluation of automated tetrahedral-prismatic mesh generation software.
*Computer-Aided Design*
http://dx.doi.org/10.1016/j.cad.2015.06.010

Access to the published version may require subscription.

N.B. When citing this work, cite the original published paper.

Permanent link to this version:
http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-171086
Implementation and evaluation of automated tetrahedral-prismatic mesh generation software

David Eller\textsuperscript{a}, Maximilian Tomac\textsuperscript{b}

\textsuperscript{a}KTH Aeronautical and Vehicle Engineering, Stockholm, Sweden
\textsuperscript{b}FS Dynamics Sweden AB, Solna, Sweden

Abstract

An open-source implementation of an efficient mesh generation procedure for hybrid prismatic-tetrahedral meshes intended for use in Reynolds-averaged Navier-Stokes solutions is presented. The method employed combines the established, and very fast, Delaunay-based tetrahedral mesh generator TetGen with a novel technique for the creation of a prismatic layer, where constrained global optimization of the envelope is employed. Once a well-shaped envelope is thus obtained, a semi-structured layer of pentahedral elements is extruded between wall and envelope surface. Satisfactory mesh quality is demonstrated by comparing solutions obtained using the new meshes with reference data computed on high-quality advancing-front grids. Mesh generation time is shown to be substantially smaller than with many other methods. Overall, the presented implementation is deemed a valuable tool for cases where many meshes need to be generated for routine analyses and turnaround time is critical.

This is an extended version of the paper presented at the 23rd International Meshing Roundtable in London, October 2014.

1. Introduction

Aeronautical applications are frequently characterized by high Reynolds numbers and significant turbulence, where the overall flow field can be substantially affected by the development of a comparatively thin boundary layer close to the wall surface. Numerical solutions of the Reynolds-averaged Navier-Stokes equations (RANS) are often performed by means of well-established finite-volume methods. Even though such methods use turbulence models to circumvent the need for the resolution of very small scales, the required solution accuracy mandates that the computational mesh is capable of resolving the large velocity gradients present in turbulent boundary layers. An approach which is widely used for industrial applications with complex geometry combines an unstructured tetrahedral mesh in most of the fluid domain with a semi-structured layer of triangular prismatic elements adjacent to the wall surface. This prismatic layer need not necessarily match the extent of the physical boundary layer, as long as the number and interior spacing of prismatic elements is sufficient to resolve the solution.

Even with this hybrid prismatic-tetrahedral approach, the creation of high-quality discretizations remains a challenging task \cite{1-3} to this day. Even on modern computer hardware and with substantial engineering effort, the application of state-of-the-art mesh generation algorithms may still result in inadequate local mesh quality, which can severely affect the resolution of important flow features, e.g. in the wing-fuselage junction region \cite{4}. To address such issues, mesh generation tools for hybrid unstructured grids often expose a considerable number of algorithm configuration parameters \cite{5-7}, many of which have a profound influence on the robustness of the process. The resulting flexibility allows the user to create sufficiently resolved hybrid meshes, although parameter selection often requires a considerable effort even for an experienced user.

URL: dlr@kth.se (David Eller), max.tomac@gmail.com (Maximilian Tomac)
In numerous applications, expending this effort is fully justified in order to obtain high-quality solutions e.g. for the drag evaluation of a commercial aircraft. Nevertheless, there are also cases with very different requirements. Aerodynamic load calculations, for instance, must routinely be performed for very many different and often geometrically complex configurations\(^1\). It is the latter type of problem which the present approach attempts to address by means of mesh generation efficiency and a large degree of automation.

Since an automatic mesh generation procedure cannot rely on user intervention for the resolution of mesh inconsistencies resulting from geometric complications, a robust strategy for the handling of surface geometry features encountered in realistic aircraft configurations must be implemented. One possible, innovative solution is to accept a hybrid mesh with known regions of low element quality, and to then augment the unstructured volume mesh with one or a number of structured overset blocks which can be manually adapted to better resolve critical geometric features \([8, 4]\). Obviously, this approach requires robust solver support for overset grids, which is not necessarily available in all state-of-the-art software for industrial use. Another approach to improve robustness is the transition to a combination of prismatic, tetrahedral and octtree-based mesh generation procedures, which has been shown to allow a surprising flexibility in the presence of difficult geometric features \([9]\). From the information available at this time, it is however not clear how the resulting mixed mesh topology and locally biased edge alignment will affect the solution accuracy for three-dimensional boundary layer flows. Previous investigations have shown that the accuracy of standard finite-volume schemes can be rather sensitive to this particular type of irregularities \([10, 11]\).

The approach presented here is based on a segregated prismatic/tetrahedral mesh generation procedure, and aims to achieve robustness by means of local geometric modifications. Criteria chosen and algorithmic modifications make use of similar principles as in earlier work \([12–14]\), but are adapted for the specific requirements of mesh generation for aircraft configurations. An existing set of open-source tools is exploited for mesh data structures, file format support, surface mesh generation and the creation of tetrahedral volume meshes.

Surface mesh generation capabilities of the current tool-chain have been presented earlier \([15]\); therefore, the present paper is focused on the procedures employed in the volume mesh generation step. Such a decomposition is possible as the algorithms do not currently exploit any coupling with the surface mesh generation stage and can therefore also be utilized to create a hybrid prismatic-tetrahedral mesh around an existing triangular surface created by any other software, provided that this surface mesh is of sufficient quality. Requirements for surface mesh quality are discussed in Section 3.3.

2. Aim

In contrast to many other mesh generation procedures focusing on mesh quality, the aim of the present effort is to obtain the capability to robustly and with minimal user interference produce hybrid meshes suitable for engineering computations using vertex-centered finite-volume codes for the solution of the RANS equations. The authors acknowledge that there are a multitude of challenging flow problems which still require the use of other mesh generation tools and algorithms in order to create an adequate computational mesh. The present method does not aim to substitute existing software for the manual creation of very high-quality meshes, where a-priori knowledge of the flow field can be successfully exploited to obtain the best possible discretization for a particular case.

Nevertheless, it is anticipated that a fast and comparatively robust tool will allow for significant savings in time and effort where meshes for many different routine flow simulations must be generated. An example of such applications may be the automated creation of a series of meshes for a full aircraft model in a windtunnel, which often requires a separate mesh for each experimental setting in order to correctly capture wall effects and elastic model deformation. Another use case which could possibly benefit substantially is the application to military aircraft with multiple external stores, where potentially as many as hundreds of different geometric configurations need to be handled with manageable effort.

---

\(^{1}\text{e.g. control surface deflections, high-lift device extension, landing gear, air brake or spoiler deployment, external stores}\)
Generally speaking, the purpose of creating a hybrid prismatic mesh is to substantially increase mesh resolution in boundary layers, where high Reynolds number flows exhibit large gradients in the wall-normal direction. It is, however, important to note that the extent of the prismatic mesh layer does not correspond directly to the size of the boundary layer; in fact, the outer limit of the prismatic layer will usually far exceed the boundary layer displacement thickness. This is an intentional feature of the present method which permits a smooth volumetric transition between the uppermost prismatic elements and adjacent tetrahedra. Therefore, only the lower part of the prismatic layer (near the wall) is actually intended to resolve the physical boundary layer, which must be taken into account when considering the number of layers and the wall-normal element size expansion ratio.

3. Method

The mesh generation strategy is based on four phases, starting with the creation of a sufficiently resolved surface mesh. In a second step, the envelope mesh of the prismatic boundary layer mesh is determined; the robustness of this stage is the primary contribution of the present work. Thirdly, tetrahedral elements are generated to fill the volume between the envelope of the prismatic layer and the farfield boundaries, and finally, pentahedral elements are grown between adapted wall and envelope mesh. Note that in this text, the term pentahedron refers to the five-sided triangular prism, not to the square pyramid.

The prismatic mesh envelope generated in the second stage is a triangulated surface of the same topology as the surface mesh. Since pentahedral elements are extruded between the wall surface and this envelope, its geometry governs the quality of these elements. In order to obtain an envelope permitting good quality prisms, a global constrained optimization process is employed. Naturally, the use of numerical optimization in mesh generation is by no means new in itself. The contribution of the current work is to limit the optimization to the envelope shape, which results in an optimization problem with typically two orders of magnitude fewer design variables than a full-mesh optimization. This reduction permits the effective utilization of quickly converging gradient-based optimization algorithms.

3.1. Surface meshes

For the generation of unstructured surface meshes for full aircraft configurations, the open-source program SUMO can be used. The geometry is in this case represented by a moderate number of parametric surfaces, which can be linear-cubic or bicubic polynomial spline surfaces, analytically defined, or non-uniform rational b-spline (NURBS) surfaces.

Triangular surface meshes are then constructed by a method which accounts for a three-dimensional version of the Delaunay criterion. Two mesh refinement passes are used in order to fulfill a set of triangle quality criteria which can either be determined based on heuristics or specified by the user. Heuristics exploit information which is available due to the fact the the geometry modeling component included in SUMO is specialized for aircraft configurations.

If the use of SUMO is not suitable, a triangular surface mesh created with any other mesh generation tool can be used. Such meshes can be imported from files in CGNS, SU2, legacy VTK, the NetCDF-based format used by the TAU code, or the widespread, but less efficient STL format. Spherical far-field boundaries can in this case be generated automatically from a small set of user-defined parameters.

3.2. Prismatic envelope

Once a surface mesh is available, the volume to be filled with prismatic elements is determined by constructing a second triangular surface with identical topology at a locally varying distance from the wall mesh. In the following text, this second triangulated surface is called the envelope, as it encloses the entire prismatic layer. In order to handle the multitude of geometric difficulties which can occur at this stage, the shell surface is not a simple extrusion of the surface mesh along local normals. Instead, a sequence of passes are applied as follows:

\[ \text{Strictly speaking, a prism must have two parallel faces; the pentahedra generated here do not fulfill this condition.} \]

\[ \text{Available from } \text{http://www.larosterna.com/sumo.html} \]
1. feature extraction and surface node classification;
2. selective smoothing of growth directions;
3. selective smoothing of layer height;
4. local untangling and warp reduction;
5. global collision avoidance of opposing layers;
6. global constrained optimization;
with the aim of transforming the envelope surface into a suitable upper boundary of the prismatic mesh region.

Theoretically, it would be possible to define a set of objective functions and constraints for the final optimization pass, such that a satisfactory envelope is obtained through optimization alone. However, by processing a number of complex examples, it has been found that the optimization converges much more reliably to a favorable state when initialized with the result of the (very fast) heuristic smoothing and untangling process described below.

3.2.1. Surface node classification

The first step in creating the envelope surface is to detect and classify feature nodes with respect to geometric properties which necessitate special treatment of either local height or growth direction. Local criteria, such as the angle between the normal vectors of adjacent triangles, are evaluated in order to assign a set of flags to each vertex of the wall mesh. Multiple flags may be combined; a vertex can, for instance, be classified as lying on a detected mesh feature line (ridge) and simultaneously carry a convex and concave flag, which would make it a local saddle point.

Some typical examples are vertices which are part of edges (which, again, may be blunt or sharp, convex or concave), or corner vertices. Figure 1 shows the wall surface of the F-16XL aircraft used in the cranked arrow wing project (CAWAPI,[20]), where dark regions mark vertices which carry any flag differing from the one used to indicate a locally smooth surface. One of the factors controlling the classification is a user-provided feature angle. This angle defines a limit for the dihedral angle between triangles, above which edges are considered to be part of feature lines. Therefore, the feature angle should be chosen larger than the maximum dihedral angle occurring between wall triangles in smoothly curved regions. This value is typically known to the surface mesh generator so that it can therefore be exploited automatically.

Note that the feature detection phase only evaluates local geometry, and ignores information such as boundaries between components (mesh regions, boundary patches). This is visible at the junction between the canopy and fuselage or the vertical tail and root of the vertical tail in Figure 1, and fully intentional.
3.2.2. Envelope smoothing

The envelope surface enclosing the prismatic layer is initialized using the vertex-based surface normal vectors obtained by angle-weighted averages of the normal vectors of coincident triangles at each vertex, according to the method of Thürmer and Wüthrich [21]. Furthermore, the local layer thickness is set by computing the mean length $l_j$ of the incident edges at vertex $j$. Then, the initial local layer height $h_j$ is determined from a user-supplied first prism height $h_0$ according to

$$h_j = h_0 \frac{1 - r_n^j}{1 - r_j} \quad \text{with} \quad r_j = \min \left[ \left( \frac{l_j}{h_0} \right)^{\frac{1}{n-1}}, r_m \right],$$

(1)

where $r_j$ is the local height expansion ratio of adjacent prisms. This growth ratio may not exceed a user-supplied maximum value of $r_m$. The height distribution is chosen such that the wall-normal edge length of the outermost pentahedron is as close as possible to the edge lengths of the abutting tetrahedron. The reason for this selection is that, for finite-volume methods employing a vertex-centered discretization, the accuracy of gradient reconstruction algorithms tends to improve when the lengths of edges connected to the same vertex do not vary too much [10, 22].

Due to triangle size variations and considerable local alterations in normal directions, the initial envelope surface is often very irregular in shape. The left side of Figure 2 shows an example; no sound prismatic mesh can be generated between the wall and this boundary surface.

Therefore, three smoothing passes are performed either indirectly or directly on the envelope surface. During the first pass, a modified Laplacian smoothing operator is applied to the height field by means of a small number of Jacobi iterations, where the height modification operation is adapted depending on the value of the vertex flags identified earlier. Secondly, a similar procedure is utilized to smooth the growth directions (initially, vertex normals) with a different pattern of flag-dependent modifications. Modifications based on vertex flags aim to avoid a deterioration of the feature resolution which would result from isotropic smoothing; as an example, normal directions for vertices marked as being part of feature edges are smoothed exclusively with respect to other direction vectors located on the same feature edge.

On the right side of Figure 2, the resulting envelope surface after smoothing is displayed. This surface is sufficiently well-shaped to permit the generation of good-quality pentahedral cells between wall and envelope mesh.

3.2.3. Edge-based untangling

The envelope surface generated in this way will often contain self-intersections which would result in entangled pentahedral elements. To remove such intersections, an algorithm using only local, edge-based geometry is run first, followed by a second, global procedure.

In the first phase, an edge-based limiter for the prismatic layer height is applied. For each edge $\vec{e}$ in the wall mesh, the angles $\beta_1$ and $\beta_2$ between the edge direction and the vertex normal vectors in the endpoints of the edge are computed. Using $\gamma = \pi - \beta_1 - \beta_2$, the maximum permitted height at the edge endpoint $j$ is then found as

$$h_j < |\vec{e}| \frac{\sin \beta_j}{\sin \gamma} \quad \text{when} \quad 0 < \gamma < \pi.$$  

(2)

This condition can be interpreted as the maximum generatrix length of a cone with apex half-angle $\beta_j$ originating in the edge endpoint $j$ which still ascertains that both cones ($j = 1, 2$) cannot intersect. Since the normal directions of two wall nodes sharing an edge do not usually differ considerably after the previously applied smoothing stage, the height limit given by (2) is often rather permissive. Only near sharp features, such as the location where a sharp wing trailing edge intersects a blunt fairing surface, will this criterion impose a significant restriction. Once the local height is reduced accordingly, a simple smoothing of the height variable in the ring-2 neighborhood of all modified vertices is performed in order to soften the transition.

3.2.4. Warp reduction

The initial prismatic layer can contain cells which are strongly warped or even inverted. A warped pentahedron is shaped such that the angle between a triangle on the envelope and any one of the three
growth direction is small. In some cases, such as the one shown in the left part of Figure 3, envelope triangle normal vectors may even point toward the wall when the wall-normal sides of the pentahedron intersect. The reduction of variation between envelope direction vectors in the vicinity of sharp features performed earlier does not solve this problem in general, as illustrated by the central part of Figure 3. In order to eliminate such degenerate cases, the local height at the affected vertices is reduced to the largest possible value that avoids warping. The permitted height is found by means of bisection such that a positive minimum pentahedral corner angle is obtained, as indicated by the grey arrows in the rightmost part of Figure 3. Since the limit of zero local height always yields an acceptable pentahedron, a reduction of the height must converge to a permissible solution. It is apparent from Figure 3 that the use of smoothed growth directions instead of surface normal vectors permits the use of substantially larger warp-free envelope height.

In plane geometry as depicted in Figure 3, the edge-based untangling of Section 3.2.3 would actually already resolve the warping. This is, however, not so in three dimensions since the envelope triangle can be rotated such that (2) is fulfilled even though the envelope macro-element is excessively warped.

3.2.5. Encroaching bodies.

In some instances, the geometry of the envelope surface cannot be determined by purely local geometric considerations. An example is the region between a rear-mounted engine nacelle and the fuselage shown in Figure 4 and Figure 5, where the unmodified prismatic region envelope enclosing the nacelle would intersect the layer around the fuselage.

As is visible in Figure 4, the potential collision was detected and the layers on both sides were retracted, i.e. their height reduced in order to alleviate the problem. Efficient detection of self-intersection is performed by means of a point search data structure, namely a balanced binary tree bounding volume hierarchy. This
particular data structure is used to perform fast queries of the geometric neighbourhood of a point on the envelope to test for the presence of other points within a given critical radius. If any such point is then categorized as indicating collision by comparison of the corresponding local normal directions, the local envelope height is reduced accordingly. As the relation between height and intersection state can be rather irregular for complex geometry, the process of envelope modification and testing for self-intersection is repeated until no further collisions are detected. Often, just two or three iterations are required; for very intricate configurations, as many as 20 repetitions can be needed which still does not account for a significant amount of computational effort.

Most of the envelope modification procedures process each vertex independently and could therefore be performed in parallel, should the need arise. At present, however, the (serial) generation of the tetrahedral mesh region dominates the overall mesh generation time, which is why only the most expensive steps of the above envelope construction have been parallelized so far.

3.2.6. Global optimization

In many simpler cases, the envelope obtained from the above process is already usable for the generation of a well-formed prismatic mesh. For more complex geometries however, substantial improvements can be obtained from a further global optimization pass. The non-linear optimization problem solved here is of the form

$$\min_z \sum_i f_i(z)$$

subject to

$$z_j,l \leq z_j \leq z_j,u$$

and

$$\sum_k g_k(z) \leq 0$$

with

$$G^+ = \{ k \mid g_k(z) \geq 0 \},$$

where each $f_i(z)$ is a scalar, non-negative objective function to be minimized, $z$ is a vector of design variables fully defining the envelope surface shape, where each design variable $z_j$ is limited by lower and upper bounds $z_j,l$ and $z_j,u$. Finally, and most significantly, the solution $z$ is constrained to fulfill a set of scalar non-linear constraints $g_k$ which are formulated such that they evaluate to positive values if violated and zero otherwise. Hence, the combined constraint function is only fulfilled if $g_k \leq 0 \forall k$.

Note that this particular form of the constraints (5) is chosen in order to allow for a memory-efficient application of the particular solver used. It is less than ideal in the sense that it forces the constraint functions to be written in a form which is only $C^0$-continuous even though the underlying geometric condition is at least $C^1$-continuous. However, expanding the single constraint (5) into a large number of component equations increases the memory required by the numerical optimization library in an unacceptable fashion since the available implementation does not permit the passing of constraint derivatives as sparse matrices.

The design variables $z$ in (3) are chosen as the three coordinates $(u_j, v_j, h_j)$ of the envelope vertex in a fixed coordinate system local to each wall vertex. While the value $h_j$ designates the local height of the envelope (i.e. the thickness of the prismatic layer), the remaining two values $u_j, v_j$ are chosen as coordinates along two arbitrary orthogonal axes in the tangent plane. In this way, bound constraints for $h_j$ can be imposed directly such that collision of opposing prismatic layers as illustrated in Figure 4 can be prevented. Implementing an equivalent geometric constraint for this condition would be possible, but its derivative – which is required for the gradient-based numerical optimization – is excessively expensive to compute. Therefore, the height values resulting from the layer collision avoidance process 3.2.5 are imposed as upper limits in (4). Furthermore, height values are enforced to be larger than the user-supplied thickness of the first layer of prismatic elements.

The solution of the problem (3 - 5) is implemented such that different functions $f_i$ and $g_k$ for objective and constraints can be applied. The present form imposes two objective terms $f_1$ and $f_2$ and two constraints $g_1$ and $g_2$. To define these functions, the prism connecting each wall-triangle with the corresponding triangle
on the envelope is labeled a macro-element. Both constraint terms are defined as a sum over contributions from such macro-elements. The first term \( g_1 \) imposes a non-inversion condition for each prism \( m \) according to

\[
g_{1,m} = g_{1,\text{min}} - \frac{\hat{n}_w \cdot \hat{n}_e(z)}{||\hat{n}_w||^2}.
\]  

(6)

Here, \( \hat{n}_w \) is the normal of the wall triangle and \( \hat{n}_e \) the normal of the envelope triangle which depends on the value of the design variables \( z \) while \( \hat{n}_w \) does not. The constant \( g_{1,m} \) is a small value to protect against numerical cancellation errors. For each macro-element \( m \), the value of this constraint becomes positive (active) when either the angle between wall and envelope triangles progresses beyond \( \pi/2 \), or when the envelope triangle becomes nearly degenerate in the sense that its vertices become co-linear or coincident.

The second constraint protects against self-intersecting macro-elements. Similar to the above, the second constraint is also a sum over non-negative contributions

\[
g_{2,m} = - \min_{k=1\ldots3} \Delta s_k(z) \cdot \hat{n}_w.
\]  

(7)

In this case, \( \Delta s_k \) is the macro-element edge connecting the \( k \)th vertex of the wall triangle with the corresponding one of the envelope triangle. This constraint term becomes active only for those macro-elements where one of the envelope vertices lies below the plane of the wall triangle, which is clearly not acceptable.

The objective function combines a macro-element quality criterion \( f_1 \), which, again, consists of a sum over terms for each element, and a height-maximization term \( f_2 \). To quantify macro-element quality, a measure of orthogonality according to

\[
f_1 = \frac{1}{3n_c} \sum_{m=1}^{n_c} \sum_{k=1}^{3} \left( 1 - \frac{\hat{n}_w \cdot \Delta s_k}{||\Delta s_k||} \right)
\]  

(8)

is employed. This term is positive and becomes minimal for prisms where all three vertical edges \( \Delta s_k \) are parallel to the unit normal \( \hat{n}_w \) of wall triangle \( m \). Obviously, it is not possible to drive \( f_1 \) to zero unless the entire wall is a single plane surface.

Objective (8) combined with the constraint (6) has a tendency to globally reduce the envelope height. Therefore, a second objective

\[
f_2 = \frac{1}{n_v} \sum_{j=1}^{n_v} \left( 1 - \frac{h_j}{h_{j,u}} \right)
\]  

(9)

is added. This simple form has the effect of pushing the envelope height towards the local maximum \( h_{j,u} \). Due to the bound constraint (4), the terms in the sum (9) must be positive.

As is often the case for this form of multi-objective optimization, the results can vary with the weighting factors applied to the individual terms [17]. The present weighting, which in some sense normalizes the contributions with the number of individual terms, must therefore be regarded as a preliminary solution that may need to be reviewed.

The full non-linear optimization problem (3) is solved by means of the Method of Moving Asymptotes (MMA) [23, 24] as implemented in the NLopt source-code library [25]. This algorithm is particularly well-suited here as it permits the solution of problems with a large number of variables with limited memory requirements. Since there are three optimization variables per wall vertex, the number of values to optimize will often exceed one million, well beyond the capabilities of many other algorithms.

When starting with an unfeasible initial solution, that is, in the case that the result of the smoothing and untangling procedure described in 3.2.1 - 3.2.5 does not fulfill constraints (6) and (7), the implementation in NLopt favors to first reach a feasible point where \( g_1 = g_2 = 0 \), even at the cost of increasing the value of the minimization objective. From a practical point of view, this behavior is desirable for the present application. A feasible, but non-optimal solution is often found rather quickly, after which the non-linear optimization progresses to reduce the objective functions.

Optimization is terminated either when

1. the relative change in \( z \) drops below \( 10^{-6} \); or
2. a user-supplied computation time limit is exceeded.

In some cases, a first-order quasi-optimal solution, i.e., a $z$ such that the gradient norm becomes very small, is found in $O(100)$ iterations. Even if that is not the case and MMA only makes slow progress in reducing the value of the objective, the latter criterion still allows for the use of the global optimization stage. In such a situation, the solution generated does not represent an optimum, but it is nevertheless useful as it provides for an improvement over the initial state.

Other mesh optimization methods exist. Within the scope of the present work, the MESQUITE toolkit was evaluated as well [16, 26]. For the limited task of envelope optimization, however, MMA achieved better results within practical optimization time limits. Due to the use of low-memory algorithms such as conjugate-gradient, MESQUITE can be applied to the entire mesh even on distributed-memory computers, and could therefore be used in a mesh post-processing stage on the solution cluster.

3.3. Tetrahedral mesh

Before prismatic elements are generated in the region between wall and envelope layer, tetrahedral elements are created in the space between the envelope and the farfield boundaries. The efficient tetrahedral mesh generation program TetGen\(^4\) developed by Han Si [27, 28] is employed for this purpose. A number of element quality parameters can be passed to TetGen in order to control the level of refinement. In many cases, these element quality criteria cannot be satisfied unless some of the boundary triangles are split to allow for smaller adjacent tetrahedra. As the method used to create prismatic elements builds upon the assumption that wall and envelope mesh have the exact same topology, such triangle splits need to be propagated to the wall mesh as well.

In order to relate vertices inserted by TetGen on the envelope to the wall surface, each boundary triangle passed to TetGen is tagged with an integer identifying the corresponding wall triangle. Newly created envelope triangles inherit the same tag, so that it is possible to split the wall mesh by inserting a new wall vertex at the barycentric coordinates of the envelope vertex created by TetGen. The lookup procedure used to determine whether a vertex received from TetGen corresponds to a previously defined envelope vertex or requires a wall triangle split makes use of the same search data structure also employed in resolving collisions between encroaching envelopes as explained in Section 3.2.5.

Most solvers for high-speed flows require that the mesh is sufficiently resolved in the vicinity of shocks. Insufficient mesh resolution leads to excessive numerical dissipation which has the effect of smearing out the shock, thus resulting in inaccurate pressure distributions. Adaptive mesh refinement is known to be an effective way to mitigate this problem in a more-or-less automated fashion; nevertheless, the additional steps to be performed by the user add to the (already large) complexity incurred by using CFD. Furthermore, aggressive solution-based mesh adaptation is best suited for problems with essentially steady flow.

In order to obtain a reasonable resolution of shocks without mesh adaptation, the tetrahedral mesh generation phase can be split into two TetGen passes. In the first pass, a relatively low-quality background mesh is created quickly by specifying a relatively large permitted circumsphere radius-to-edge ratio of around 1.5. Then, this mesh is employed to specify a nodal element size map which approximates the smooth element size transition desirable for a finite-volume mesh. In a second pass, TetGen is run in refinement mode with the nodal element size map on the background mesh as an input. The resulting tetrahedral mesh is characterized by a gradual element size transition visible in Figures 8 - 10.

3.4. Extrusion of prismatic elements

Starting from the wall surface, prismatic elements are finally generated by filling the space between wall and envelope with a prescribed number of prismatic element layers. The height of the first prismatic cell is a user-defined value which is applied throughout the entire mesh. Starting from the second cell, a constant growth ratio is maintained such that the last cell reaches the local layer thickness. Hence, the growth ratio may differ substantially between regions of the mesh with different triangles sizes. However, the element

\(^4\)Available from \url{http://www.tetgen.org}\)
generation process is extremely fast since the prismatic layer is essentially structured in the wall-normal direction, meaning that the number of pentahedral layers is identical everywhere.

In some areas, the layer thickness may be so small that the above simple procedure would result in a growth rate below one, that is, the cell height would diminish away from the wall. An equally spaced distribution of cell heights across the layer is selected if such a situation arises. A typical geometrical feature exhibiting this pattern is the narrow lateral gap between two control surfaces.

Mesh quality is further improved by aligning the first several pentahedra (nearest to the wall) not with the previously determined smoothed growth direction, but starting with the actual wall normal vector and transitioning gradually. This leads to almost perfectly wall-aligned pentahedra in the first layers, but can produce tangled elements when very sharp concave edges are present.

With this improvement, the mesh node $p_j$ in the $j^{th}$ layer above the wall node $p_1$ is computed as

$$p_j = (1 - t_j)p_1 + t_j(s_j(p_1 + h_j\vec{n}_j) + (1 - s_j)p_n)$$

where

$$t_j = \frac{h_0}{h_j} \left( \frac{r_j^{j-1} - 1}{r_j - 1} \right),$$

$$s_j = \begin{cases} e^{-t_j/c_t} & \forall c_t > 0 \\ 0 & \iff c_t = 0 \end{cases}$$

Here, $p_n$ is the envelope node above $p_1$, $\vec{n}_j$ the unit wall normal, and $h_j$ the layer height. Note that $p_n - p_1$ is often not parallel to $\vec{n}_j$. The wall-perpendicularity parameter $c_t$ controls how the layer growth direction transitions from the wall normal towards the envelope direction; a larger local value of $c_t$ permits more prismatic layers to remain well-aligned with the wall. The default value of $c_t = 0.05$ approximately results in the lowest one-fifth of the prismatic region to be oriented almost perfectly normal to the wall, whereas an increase to $c_t = 0.2$ expands this fraction to about one-third of the envelope height. Due to the exponential expansion, the lower third tends to contain a large percentage of the pentahedral elements.

The effect of the parameter $c_t$ is shown in Figures 6 and 7 for the trailing-edge region of a transonic transport aircraft wing. Even though the envelope direction vectors are far from perpendicular to the wall, most pentahedra maintain fairly good wall alignment here; only the topmost 8 of 36 prismatic layers deviate substantially. With a smaller value of $c_t = 0.05$, the perpendicularity displayed in Figure 5 is achieved, where the lower 18 of 32 prismatic elements deviate by less than a few degree from optimal wall alignment.

Occasionally, the extrusion process can produce self-intersecting prisms even when the corresponding macro-element between wall and envelope is well-formed, in particular for large (non-default) values of
Here, self-intersection is defined as the case of at least one prism vertex being located on the wrong side of the opposite prism triangle. This problem is particularly visible when using comparatively coarse surface meshes of complex geometries, where the local layer height changes substantially across a single macro-element. Such rare self-intersections are alleviated by moving the violating nodes in the direction of the normal of the prism base triangle just enough to resolve the intersection. Since this step is repeatedly applied in a local manner to violating prisms, there is strictly speaking no guarantee of convergence, even if the procedure is very effective in most practical situations.

Whenever the diagnosis of prismatic elements detects unrecoverable self-intersections during the extrusion process, the wall-perpendicularity factor $c_t$ of the corresponding wall nodes is set to zero. In order to avoid large differences in the first-layer normal directions, $c_t$ is reduced in the immediate topological neighborhood as well. With this modification, the entire extrusion process is repeated. Note that this second extrusion pass is rarely needed, and only affects a small number, $O(10)$, wall nodes. Due to the efficiency of the extrusion process, the additional computational cost is small.

### 3.5. Interoperability

Once the prismatic elements are generated, the resulting hybrid mesh is post-processed by merging duplicate nodes. Some simple element validity checks for negative element volume and degeneracy of the dual mesh are performed. Finally, the mesh file can be written in a native, LZ4-compressed binary format or one of the supported mesh exchange and solver formats. At the time of writing, CGNS$^5$, the bmsh file-format for the Edge solver [29], NetCDF-based mesh files for TAU [30] and the text-based SU2 format [31] can be exported.

Many solvers for the Reynolds-averaged Navier-Stokes equations support special semi-implicit methods tailored towards more efficient treatment of problems with resolved boundary layers. Such line-implicit iteration methods (or preconditioners) make use of the strong coupling of variables associated with mesh nodes located on lines approximately normal to the wall. In hybrid meshes generated with the present method, these nodes are already numbered sequentially, thus eliminating the need for an explicit line-identification heuristic on the solver side and potentially improving cache locality in the solution process.

The software presented here is implemented in standard C++ and available under an open-source license. A simple command-line interface is provided in order to facilitate automated processing of surface meshes generated by third-party tools. Furthermore, the same method is also linked into the sumo open-source mesh generation software for preliminary design.

### 3.6. Limitations

Some CAD models and derived surface meshes contain degenerate geometric details, which would not usually exist in an actual product but are caused by modeling simplifications. Such degeneracy can lead to the case where it is not possible to define a vertex normal which sustains an angle of less than 90° with the normals of all of the coincident triangles, which renders the construction of well-defined pentahedral elements impossible. The present implementation collapses the layer into the wall node at such points.

In order to properly handle such geometric degenerate points, multiple vertex normal directions would need to be defined and the algorithm of Section 3.4 would become significantly more complex because different types of elements need to be explicitly generated in the layer in this case (prisms, tetrahedra, hexahedra, pyramids). Furthermore, the constraints and objective function of the optimization procedure which vastly improves envelope macro-elements would need to be augmented: Straightforward application of constraints (7) and (6) macro-pentahedra entails the risk of collapsing macro-tetrahedra produced by point with multiple normals. Nevertheless, introducing multiple normals remains an attractive option for future improvements as it has the potential to very substantially improve mesh quality near acute convex ridges such as sharp trailing edges.

---

$^5$32-bit integer CGNS 2.53 based on ADF
4. Example applications

Figures 8-10 are intended to show a selection of example meshes generated with the present approach, ranging from a relatively benign jet trainer prototype case to a simplified Airbus A320 undercarriage assembly.

Unfortunately, purely geometrical element criteria appear to represent poor practical measures of overall mesh quality. Therefore, a comparative study was performed with two selected configurations, where RANS solutions obtained on meshes generated by SUMO were compared with those computed on available grids with similar resolution created with other software systems. The aim of this study was to evaluate whether the quickly generated SUMO meshes yielded similar simulation results in sensitive metrics. In the absence of a more rigorous criterion relating mesh geometry to solution convergence and accuracy, this is regarded as a relevant investigation.

Two very different cases were selected for this comparison. The first is the Common Research Model (CRM) wing-body-tail geometry representative of a wide-body airliner, which should present a comparatively innocuous case. In contrast, the second model used is the F-16XL research aircraft evaluated at a flight
condition known to involve a truly intricate flow topology.

4.1. Common Research Model

The Common Research Model (CRM) is a geometrically fairly simple wing-body-stabilizer wind-tunnel configuration developed for applied CFD validation studies [32]. It is intended to reproduce many of the large-scale flow features present on commercial aircraft currently in service. In this study flow simulations for the CRM were performed for a set of Mach numbers between 0.7 and 0.87 and angles of attack from zero to six degrees at a chord-referenced Reynolds number of 5 million. The prismatic grids generated for this model follow the general grid guidelines from the fourth drag prediction workshop (DPW-4)\(^6\). In order to focus the evaluation on the quality of the volume mesh generation process, a mirrored surface mesh from the DPW-4 website was used, namely the medium-resolution grid generated by DLR, containing 1.2 million surface triangles.

4.2. F-16XL Research Aircraft

The F-16XL aircraft on the other hand is an elaborate configuration that presents many unique aerodynamic challenges across its flight envelope [20]. This vehicle incorporates cranked delta wings designed for efficient supersonic cruise. As a result, the wings are thin, highly swept, and feature relatively small leading-edge radii. The configuration investigated here also includes wing-tip missiles, air-dams, pods, narrow gaps and other details that tend to be a challenge for many prismatic grid generators [33].

In this study the current prismatic grid generator was used to produce a hybrid grid of the F-16XL aircraft for a flow simulation of the very taxing transonic Flight Condition 70 (FC70, [34]) at Mach 0.97 and an angle of attack of 4.3\(^\circ\) and a chord-based Reynolds number of 89 million. At this angle of attack, significant interaction between the shock and a vortex emanating from the sharp inboard leading edge is expected to occur. At the time of analysis, it has been found rather difficult to obtain satisfactory agreement of computational results with flight test data at this particular flight condition with multiple different mesh and solver combinations. As the focus of the present paper is the acceleration of the mesh generation process, no experimental results are shown here. Instead, pressure distributions computed with meshes originating from different generators are compared.

The surface grid used for evaluating mesh quality is the mirrored initial KTH/FOI surface grid [34] with 316 000 surface triangles. 35 prismatic layers were generated with an initial wall distance of 10\(^{-6}\) m ensuring that for the first layer, \(y^+ < 1\) is fulfilled everywhere.

5. Simulation Results

The flow solver Edge, developed at the Swedish defense research establishment (FOI), was used for the comparative simulations [29]. Turbulence was modeled using the Spalart-Allmaras (SA) one equation model for the CRM cases and the Hellsten, Wallin and Johansson \(k - \omega\) explicit algebraic Reynolds stress model (EARSM) for the F-16XL case. Since the purpose of the simulations was not to predict aircraft performance or investigate flow phenomena, but rather to evaluate the dependency of computed results on properties of the volume mesh generation scheme, only steady-state runs were performed. It is understood that this may well lead to inaccurate results for some cases.

Edge is based on a finite-volume formulation where a median dual grid forms the control volumes with the unknowns allocated in the vertices of the primary grid. In version 5.3 employed here, the governing equations are integrated to steady state by means of a line-implicit approach in regions with highly stretched elements (e.g. the prismatic layer) and explicitly elsewhere with a three-stage Runge-Kutta time integration scheme. Convergence to steady state is accelerated by a full approximation multigrid scheme. Solid surfaces were associated with weak adiabatic wall boundary conditions, while weak characteristic exterior conditions were used for the far-field boundary. For the F-16XL case the engine inlet and mixing plane conditions were set to the same values as presented by Boelens et al. [33]. Details of the mesh element and node counts can be found in Table 2.

---
\(^6\)http://aasc.larc.nasa.gov/tsab/cfdlarc/alaas-dpu/Workshop4/gridding_guidelines_4.html
5.1. Transonic wind-tunnel model CRM

The initial mesh for the CRM geometry contained a comparatively coarse tetrahedral region with 7.7 million elements, generated with a tetrahedral quality criterion (circumsphere radius to edge length ratio [27]) of 1.2. For this mesh, some differences in pitch moment were observed when compared with the advancing-front mesh containing nearly twice as many elements in the tetrahedral domain. These differences could easily be attributed to a lack of shock resolution. Therefore, the sumo-based mesh was re-generated with a tetrahedral quality criterion of 1.05, leading to a mesh with 17.6 million tetrahedral cells, compared with 15 million cells in the reference grid. The more restrictive tetrahedral shape requirements can only be obtained by allowing TetGen to split the triangulated envelope surface where necessary. In the end, the splits are carried through to the wall surface, thus leading to a total of 1.4 million additional nodes in the prismatic layer. Note that these results were obtained before the two-pass approach to tetrahedral mesh generation described in Section 3.3 was implemented.

A consequence of the different algorithms applied for the tetrahedral mesh generation is that this region of the mesh is not directly comparable in terms of density distribution and element size transition. Additionally, the sumo-generated mesh contains a full prismatic region with all 35 layers everywhere. In contrast, other programs such as TriTet can eliminate some of the layers where that is deemed advantageous. Mostly due to these two factors and a minor increase in tetrahedron density, the mesh generated by sumo features about 27% more nodes despite being based on the same surface mesh. With the Edge solver and current (June 2015) commercial pricing for processor core-hours, this growth corresponds to a cost increase of approximately US $7 per flow solution. The savings in mesh generation effort, on the other hand, amount to several hours of man-time and about one day delay due to mesh generator run-time. Detailed mesh generation performance is documented in Section 5.3.

Figure 11 shows the drag polar for the CRM model for mach numbers 0.7, 0.85 and 0.87, where the more finely resolved farfield mesh was used as explained above. A comparison at lift coefficient $C_L = 0.5$ indicates that the drag only differs by 0.04% between the sumo-generated and reference grids. This difference is most likely significantly less than the overall accuracy of the simulation. In Figure 12 the pitch moment $C_m$ as function of angle of attack is shown. Again good agreement between meshes is observed for moderate lift coefficients, while slightly larger differences occur at $\alpha = 4^\circ$.

Figure 13 and Figure 14 show the surface pressure and skin friction distribution of the CRM at Mach 0.85 and angle of attack $2^\circ$. The upper half of each image shows the solution based on the TriTet-generated grid while the lower side shows the solution based on the sumo-generated grid with 17.6 million tetrahedral elements. No significant discrepancies between the two solutions have been recognized.
5.2. F-16XL Flight Condition 70

Table 1 compares integrated forces and moments between the advancing-front mesh generated using TriTet and the hybrid prismatic/Delaunay grid produced using sumo and TetGen for flight test FC70 of the F-16XL case. While the discrepancies in lift force and pitching moment are minor, there is also a small difference in total drag and a larger difference in friction drag – about 6 drag counts. Figure 15 shows the surface pressure distribution, where the upper half pertains to the reference solution (TriTet-mesh) while the lower half presents the results from the simulation on the sumo grid. Only a slightly larger supersonic region over the canopy can be observed in the sumo grid solution; otherwise, the differences are small given the very complex flow topology.

Further studying the tetrahedral region (see Figure 16) in the near-field around the body, it is clear that the sumo produced grid (right) has a significantly larger growth ratio, however the near-field resolution is kept higher in the tetrahedral domain compared to the TriTet-mesh (left). Even in this case, the sumo-mesh was generated before the two-pass TetGen option was available.

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>TriTet</th>
<th>SUMO</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_L$</td>
<td>0.12010</td>
<td>0.12017</td>
<td>-0.06%</td>
</tr>
<tr>
<td>$C_D$</td>
<td>0.03760</td>
<td>0.03750</td>
<td>1.0 counts</td>
</tr>
<tr>
<td>$C_D_f$</td>
<td>0.00508</td>
<td>0.00449</td>
<td>5.9 counts</td>
</tr>
<tr>
<td>$C_m$</td>
<td>-0.12931</td>
<td>-0.12925</td>
<td>+0.05%</td>
</tr>
</tbody>
</table>

Table 1: F-16XL forces and moment coefficients at Mach 0.97, angle of attack 4.3° and chord Reynolds number 89 million.

5.3. Mesh generation performance

In Table 2, the size and computing time needed for the generation of the different hybrid meshes are shown. Note that, depending on the specific geometry in question, the achievable degree of software automation and user experience, some of the tools listed may require substantial manual effort and repeated attempts, which is not considered here. Use of SUMO entails only the setting of a small number of parameters, such as the desired height of the first cell and the number of layers.

While the method used in sumo has been outlined in Section 3, a short explanation is needed for the two other mesh generators mentioned in Table 2. The mesh generator TriTet [7] builds up the prismatic region
layer by layer, whereupon the tetrahedral domain is gradually filled by means of a serial advancing front method. For the CRM case, which is typical in this respect, about 80% of the computing time expended by TrTet is used by the tetrahedral mesh generation phase.

The commercial software icem-cfd incorporates multiple algorithms for hybrid mesh generation. An advancing front method conceptually similar to the approach implemented in TrTet requires somewhat longer computing times than TrTet for this particular case. The mesh listed in Table 2, however, was generated with another algorithm resembling the procedure employed by sumo, where the prismatic region envelope is generated once and then split into pentahedral cells, while the surrounding volume is filled by means of a Delaunay tetrahedralization. This process is substantially faster at just below 2 hours, but does not result in a mesh which passes the pre-processor of the Edge flow solver due to tangled or inverted cells. Additional investigations into the controlling parameters are therefore needed before a fully representative comparison can be established.

For a resolution comparable to the TrTet mesh, i.e. 25.8 million nodes, sumo required a total computation time of slightly more than 10 minutes. A mesh for the F-16XL configuration with 8.3 million nodes was created in less than 5 minutes total time. Timings were produced on a Linux workstation with two 2.8 GHz Intel Xeon (Nehalem) processors.

### 5.4. Mesh Quality

Figure 17 shows the distribution of element skewness in the prismatic layer. The average skew angle displayed here is similar to objective \( f_1 \) given in (8) for the envelope optimization and defined as

\[
\beta = \frac{1}{6} \sum_{k=1}^{3} \cos^{-1} \left( \frac{\vec{n}_1 \cdot \Delta s_k}{||\vec{n}_1|| \Delta s_k} \right) + \cos^{-1} \left( \frac{\vec{n}_2 \cdot \Delta s_k}{||\vec{n}_2|| \Delta s_k} \right),
\]

Figure 15: Surface pressure comparison for F-16XL.

Table 2: Element counts and wall-clock generation times of the computational grids.

<table>
<thead>
<tr>
<th>Grid</th>
<th>Generation time</th>
<th>Surface triangles</th>
<th>Prism cells</th>
<th>Tet cells</th>
<th>Total cells</th>
<th>Total nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRM TrTet</td>
<td>1:32:45</td>
<td>7:20:32</td>
<td>1212k</td>
<td>34.3M</td>
<td>15.1M</td>
<td>50.1M</td>
</tr>
<tr>
<td>CRM icem-cfd</td>
<td>1:18:56 (1:51:23)</td>
<td>1212k</td>
<td>42.4M</td>
<td>14.2M</td>
<td>57.9M</td>
<td>23.7M</td>
</tr>
<tr>
<td>CRM sumo</td>
<td>0:02:01</td>
<td>0:10:17</td>
<td>1295k</td>
<td>45.4M</td>
<td>17.6M</td>
<td>62.9M</td>
</tr>
<tr>
<td>F-16XL TrTet</td>
<td>0:17:30</td>
<td>4:50:21</td>
<td>316k</td>
<td>12.2M</td>
<td>14.8M</td>
<td>27.2M</td>
</tr>
<tr>
<td>F-16XL sumo</td>
<td>0:01:01</td>
<td>0:04:38</td>
<td>316k</td>
<td>11.3M</td>
<td>16.3M</td>
<td>27.6M</td>
</tr>
</tbody>
</table>

16
where \( \mathbf{n}_1 \) and \( \mathbf{n}_2 \) are the normal vectors of the two triangular faces and \( \Delta s_k \) are the vectors connecting opposing vertices between these faces.

Figure 16: Grid comparison of near-field tetrahedral resolution for F-16XL case

\[
\begin{align*}
\text{mean skew angle } & \beta \\
\text{element fraction } & [\%]
\end{align*}
\]

Optimized – \( c_t = 0.1 \)
Smoothed – \( c_t = 0 \)

Figure 17: Pentahedron quality improvement by curvature.

Filled bars in 17 correspond to the use of envelope optimization followed by curved extrusion according to (10) with \( c_t = 0.1 \) in (12). The element skew angle distribution shown by the empty bars, on the other hand, is obtained with envelope optimization disabled and \( c_t = 0 \). Envelope optimization increases the computational cost of the entire mesh generation process by 12% in this case.

With curved extrusion, about 40% of all pentahedral elements have mean skew angles below 6°, and 77% conform to \( \beta < 18^\circ \). Without this feature, only 11% fall into the best quality category and 64% of elements have mean skew angles above 18°.

Use of curved prism extrusion \( (c_t > 0) \), and not envelope optimization, is the main contribution to the significantly improved element shape quality, because the optimization stage only acts on macro-elements, not the final pentahedra. In many cases, however, the global optimization stage improves envelope quality to such a degree that curved extrusion can be employed without introducing unrecoverable tangled elements.
6. Conclusions

When comparing the mesh generation timings, an interesting observation can be made. For the common situation where a parallel CFD solver is run on a dedicated compute cluster of moderate size, the analyst may be evaluating post-processed results of a steady-flow simulation based on a SUMO-generated mesh before an advancing front mesh has even been completed. This is a substantial advantage of the presented open-source method.

Obviously, this does not mean that there is no need for high-quality advancing-front mesh generation tools. A substantial proportion of relevant geometries and flight conditions likely require more detailed control over mesh generation parameters than what is available in the present hybrid Delaunay implementation. However, for routine solutions where serial mesh generation time is a severe bottleneck, SUMO or the underlying libraries can be used to accelerate the turnaround time considerably.

While the present implementation can be used in an almost fully automated way, the level of robustness initially aimed for has not quite been achieved yet. A large variety of different geometric configurations have been successfully meshed; nevertheless, some users still report that the quality of the prismatic layer generated for some particular cases is insufficient for the flow solver in use. Such problems are sometimes related to a failure of the node classification stage for peculiar geometric conditions; another remaining difficulty is the presence of extremely low-quality triangles in the surface mesh.

Future efforts will therefore be dedicated to improve existing surface mesh generation methods such as to avoid badly shaped triangles which presently may be created near intricate surface intersections and in situations such as near wing-tip trailing edges of wings with very thin airfoil sections. Furthermore, the generation of a C-type mesh topology for the prismatic layer near wing trailing edges will be attempted in order to improve the shape of control volumes and ameliorate wake resolution.

7. Acknowledgements

This work was partially financed by the fifth Swedish National Aeronautics Research Program (NFFP-5). Contributions from FMV project 356973-LB833400 monitored by Curt Eidefeldt are gratefully acknowledged.

References

URL http://www.ansys.com


