Tenth International Conference on Computational Fluid Dynamics (ICCFD10), Barcelona, Spain, July 9-13, 2018

ICCFD10-219

Prismatic Mesh Generation Using Minimum Distance Fields

Beatrice Roget[1], Jay Sitaraman [2], Vinod Lakshminarayan[1], Andrew Wissink [3]
Corresponding Author : beatrice.f.roget.ctr@mail.mil

[1] Science & Technology Corporation, NASA Ames Research Center, Moffet Field, CA, USA
[2] Parallel Geometric Algorithms LLC, Sunnyvale, CA, USA
[3] U.S Army Aviation Development Directorate ADD (AMRDEC), Moffet Field, CA, USA

Abstract: Anisotropic prismatic/strand meshes are often used to capture viscous boundary layer effects in Reynolds Averaged Navier Stokes (RANS) simulations of high Reynolds number flows. This paper describes a new algorithm for generation of these prismatic meshes using the minimum distance field of the surface tessellation. The algorithm is based on initial point placement using both the closest point on this iso-surface, and the direction of best visibility. Initial point placement is followed by a smoothing operation based on an elastic spring analogy, which is constrained using the iso-surface of the distance field and the region of visibility for each node. Simulations are performed using a dual-mesh infrastructure, where the prismatic meshes transition to a Cartesian background mesh a short distance from the wall. This overset mesh system is then processed by a domain connectivity method to establish connections between self-intersecting strand meshes and strand/Cartesian mesh systems. Mesh and flow simulation results are presented for test cases of varying complexity.

Keywords: Computational Fluid Dynamics, Mesh Generation, Computational Geometry

1 Introduction

Unstructured meshes near wall boundaries typically utilize anisotropic prismatic meshes to capture the viscous boundary layer. In standard mixed element unstructured meshes, these prismatic layers transition into isotropic tetrahedra. In concave parts of the geometry, this transition occurs a very short distance from the wall. A robust approach to generate strand/prismatic meshes for complex geometries is still the subject of an active area of research. Previous works in this regard can be broadly categorized as advancing front methods and direct point placement methods. Recent research efforts in advancing front methods are by Pirzadeh [1], Lohner [2], Kallinderis [3], Marcum [4], Wang [5] and Alauzet [6, 7, 8]. Most these works build on the idea of following the boundary normal direction to march outward and create a prismatic mesh close to the surface. When the "fronts" self-intersect or intersect with a front from another part of the body, fixing and merging of fronts is performed, which may lead to generation of tetrahedra and pyramids in addition to prisms. A certain distance from the wall, the mesh transitions to nearly isotropic tetrahedra, creating a hybrid prismatic mesh. In general (with the exception of Lohner [2]), most of these works generate each layer of the prismatic mesh sequentially, checking intersections, validity and quality of cells generated in each layer and performing necessary algorithmic adjustments to preserve mesh validity. Owing to the sequential nature, the mesh generation process is in general serial and expensive because of the necessity to perform large numbers of intersection checks. Despite these limitations, advancing front methods remain the most robust grid generation strategies available for generation of anisotropic unstructured meshes on complex geometries. The progress in anisotropic mesh generation and adaptation in the last decade is documented in detail in a review article by Alauzet [9].

Direct point placement techniques have been explored on a more limited basis. Recent research efforts in this regard are by Tomac et al. [10], Garanzha et al. [11], and Haimes [12]. Tomac et al. explored generation of

a prismatic envelope of the whole boundary layer, similar to Lohner [2], with a complex algorithm consisting of feature extraction, classification and selective smoothing. Once the envelope was finalized, prismatic elements were filled in by simply subdividing the lines connecting the boundary surface and corresponding points on the envelope. The use of the single straight lines make this method essentially equivalent to the strand grid approach proposed by Meakin [13], where the surface tessellation, a single vector defined at each surface node, and a distribution of layers along the normal vector, are used to compactly represent the entire prismatic mesh. Garanzha et al. [11] treated the prismatic mesh generation as a hyper-elastic spring back problem and used PDE based techniques to obtain solutions to the non-linear elastic problem. Promising results were shown for a range of geometries and techniques for removing self-intersections were also developed. Solution to the PDE systems were reported to incur large costs making the method 4 to 5 times slower than traditional advancing front methods. Haimes [12] developed a method for generation of strand-type meshes using a constrained optimization technique. This approach attempts to minimize the area of the prismatic envelope created by following the local normals for a fixed distance, with a fixed-length constraint for the strands (lines connecting surface boundary to the prismatic envelope). The main idea is to use the area minimization to facilitate untangling and unwrapping of strands, since intersections and warp always lead to a larger area when using fixed length strands. In addition, Haimes [12] also included the ability to have multiple vectors at each surface vertex (a concept introduced by Loseille [14]) that greatly improves mesh quality near areas of low visibility, such as highly convex or concave-convex features in the geometry. The use of the fixed-length constraint for strands is central to the area minimization, because variable strand lengths can cause the envelope to collapse to the surface. However, the fixed-length constraint renders the prismatic envelope more concave than the original surface itself, making it harder to create additional prismatic layers.

The work presented here is motivated by the requirement to automatically generate volume meshes from just a discrete surface tessellation for flow computations. It is evident from the large body of previous work available in literature that a fully prismatic mesh that covers the entire computational domain (surface boundary to several body lengths away from the surface) cannot be generated for all but the simplest of geometries. More complex cases require the use of either a hybrid meshing paradigm or an overset dual-mesh paradigm where the prismatic layers transition to an adaptive Cartesian system. The latter approach is used in the $CREATEA/V^{TM}$ Helios framework, which serves as the test bed for all the development, implementation and testing presented in this work. In particular, the strand/Cartesian dual-mesh methodology is explored, where a near-body strand grid system is embedded within an adaptive Cartesian system that covers the entire compute domain (Figure 1(a)).

The original strand data structure defined by Meakin [13] was used extensively by Haimes [12] to create mesh generation methods and Katz [15, 16, 17] to construct flow solver methods. Several departures from the original strand definition are introduced in order to improve the versatility of the strand/Cartesian dual mesh paradigm. First, the strands are represented as poly-line curves (lines with more than two control points) on a selective basis, creating a continuum between pure strand grids and general prismatic grids. Grids represented this way are termed multi-level strand grids. Second, the fixed-length constraint on strands is removed, in order to facilitate incremental mesh generation on a level-by-level basis, such that the surface that serves as the basis for each level is smoother than the one at the previous level. Third, multiple vectors per surface node (following the work by Haimes [12]) are introduced. Finally, strand/prismatic grids are allowed to self-intersect and create invalid cells, which are subsequently removed during the overset grid assembly process. It is worth noting that the work presented here is inspired by the methodology of Haimes [12] and is a continuation of our previous work on mesh generation and flow solutions using the strand/Cartesian dual mesh approach [18, 19, 20].

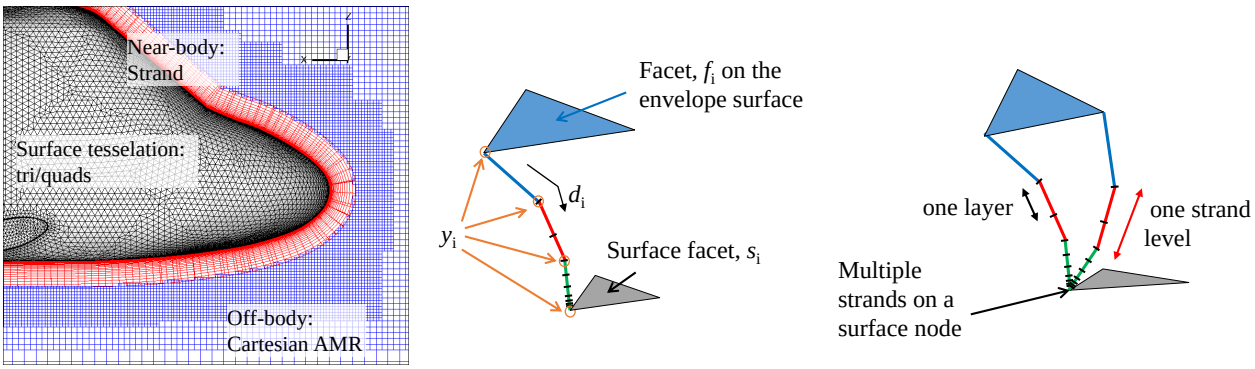
The objective of this paper is to develop an algorithm that can generate valid strand/prismatic grids that extend as large a distance as possible from a given surface grid, such that strand collisions can occur at a reasonable distance outside the area where viscous boundary layer effects are dominant. The primary challenge in strand mesh generation lies is devising a strategy for point placement in each layer of the prismatic mesh system. In this context, we use the iso-surface of minimum distance field as the guide for initial point placement and subsequent mesh smoothing. Several examples ranging from simplified test cases to realistic geometries will be presented to demonstrate the efficacy of the method.

2 Definitions

2.1 Multi-level strand/prismatic grids

The strand/prismatic grid system is fully defined in a compressed row storage format using the following data items.

1. Set of coordinate locations $y_i \in \mathbb{R}^3, i \in [1, \text{nmax}]$
2. Connectivity graph of the outer prismatic envelope, provided as facets, f_i, $i \in [1, \text{nfacets}]$, where each f_i is a triangle, with the vertex indices $\in [1, \text{nnodes}]$, where nnodes is the total number of nodes on the outer prismatic envelope.
3. Control point count, $m_i, i \in [1, \text{nnodes}]$, that specify the number of control points associated with each node on the prismatic envelope. Also note that, $\text{nmax} = \sum m_i$.
4. A single strand layer distribution function vector, $d_i, i \in [1, \text{nlayers}]$ that specifies the distribution of the actual nlayers prismatic layers. The same distribution function is used for all strands. The final nodal locations can be obtained by sub-dividing the poly-line curves, defined by y_i and m_i, into nlayers $-$ 1 segments, with each segment length proportional to $\Delta d = d_{i+1} - d_i$.



(a) Strand / Cartesian grid system (b) Multi-level strand grid description (c) Multiple strands per node

Figure 1: Description of the strand mesh system.

Figure 1(b) illustrates the strand storage format. The use of facets on the outer prismatic envelope supports multiple strands per node on the viscous wall surface: when there are multiple strands per node present, the outer envelope has a larger number of facets than the original wall surface and a surface node will connect to multiple outer nodes, as shown in Figure 1(c). The control point count, m_i, determines the number of levels in the strand mesh, with $\text{nlevels}_i = m_i - 1$. At the maximum limit, if $m_i = \text{nlayers}$, the mesh is a generalized prismatic grid, and at the minimum limit, if $m_i = 2$, the mesh is a pure strand grid as described by Meakin [13]. The compressed storage approach uses the fact that large regions of the geometry often have benign complexity and can be represented with just a single strand and distribution along that strand. However, there will be regions near convex/concave edges and corners that will require multiple levels. In a distributed computing system, compressed storage provides a considerable advantage for overset domain connectivity since the strand mesh system can be made available in each process. Since the entire dual mesh system (strand and adaptive Cartesian) is fully available in each process, all searching operations required for connectivity can be performed without any communication, leading to large benefits in efficiency and scalability (See Ref [21]). As a practical example, a 50 million node strand grid that was generated for the high-lift common research model (HL-CRM), used approximately 100MB or memory for storage with 5 control points per node. The same mesh, if stored in standard mixed element format, where the connectivity graph of each prism is stored, would take more than 4.2GB of memory and would be impossible to maintain in each process.

2.2 Inner and outer regions of the mesh

In general, for complex geometry where non-local intersections are possible, it is not viable to generate a fully closed prismatic mesh for the total desired distance away from the body. Therefore, we separate the prismatic mesh into two regions, inner and outer. The inner region is a fully closed mesh and may consist of a few levels. The outer region can have self-intersections, and is built by directly extruding and smoothing the envelope surface of the outermost inner level. It may also consists of a few levels, such that the total number of levels for both inner and outer regions is a small number, typically 10 or less. Most of the methodology presented in this paper is focused on producing a valid and closed inner region to the largest extent possible and improving its quality. For simpler geometries, with moderate concavities, it is possible to have a fully closed mesh at the desired distance and no outer region is necessary.

2.3 Required inputs

The strand/prismatic mesh generation technique uses the discrete tessellation of the body surface as input. The surface tessellation is composed of a set of nodes, $X = \{x_i \in \mathbb{R}^3, i \in [1, \text{nbnodes}]\}$ and their connectivity, $S = \{s_i, i \in [1, nbfacets], s_i = [a_1, a_2, a_3], a_k \in [1, \text{nbnodes}]\}$. The tessellation S should contain no hanging edge or node, and each s_i should have all triangles ordered such that the facet normal point the same way, i.e. for each edge of S, the associated facets have the edge extremities ordered the opposite way in their respective index list. In addition to the surface tessellation, other parametric inputs required are:

1. L, desired extent of the prismatic layers from the original surface to the prismatic envelope,
2. Δ_1, spacing required at the wall,
3. Δ_2, spacing at the outer envelope where the prismatic mesh interfaces with the Cartesian mesh,
4. nlayers, number of layers in the distribution function,
5. nmulti, maximum number of multi-strands from a node,
6. nlevels, maximum number of strand levels,
7. L_1, thickness of the initial level, and
8. s, stretch ratio for the thickness of each level

Most of the parameter inputs can be defaulted to empirically known values. For example, Δ_1 is auto-computable knowing the Reynolds number Re and $y+$ requirement, Δ_2 and L are auto-computable knowing the wake capturing requirement and nlayers can be determined using an acceptable stretch ratio. In most of the computational results presented later in the paper, the surface tessellation was the only input required and the volume mesh was constructed using the defaults for the operating condition. The default values used were nmulti $= 5$, nlevels $= 5$, and $s = 2.0$. Mesh generation proceeds on a level-by-level basis with the outer envelope of each level being used as the base surface for the next level. The thickness of the initial level, L_1, can also be computed automatically using geometric features of the surface tessellation. However, in the present work and this value was set manually. The thickness of remaining levels is computed automatically using the total desired extent and the stretch ratio. It is important to reiterate that the number of strand levels, nlevels is usually a small number (under 10), while the number of actual prismatic layers nlayers is in the order of 50-100.

Mesh generation in each level is broadly categorized into three areas. They are (1) determination of multi-strand nodes and connectivity (first level only), (2) initial strand placement and finally (3) mesh smoothing.

2.4 Algorithm overview

An overview of the entire algorithm is presented in Figure 2. After reading the mesh and user inputs, the surface tessellation is pre-processed to create the required connectivity graphs (node-to-node, node-to-cell, and node-to-edge), and the edge dihedral angles are computed. Positive angles denote convex edges, while

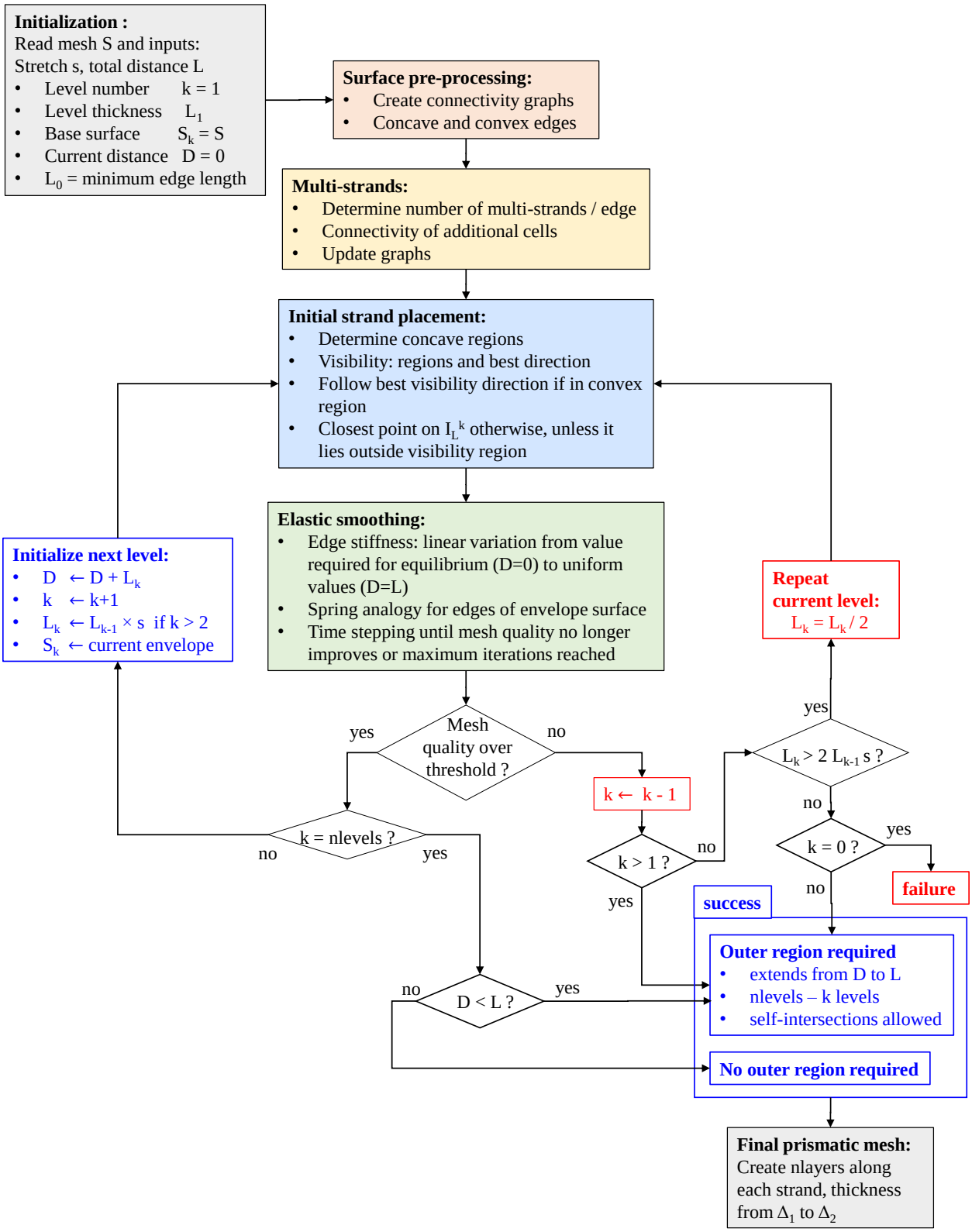


Figure 2: Algorithm overview.

negative angles denote concave edges. Edges whose angle is larger or smaller than a threshold value are tagged as convex or concave (+/- 45 deg was used in the present work). The additional multi-strand nodes and associated connectivity are then created, as described in the next section. Next, the algorithm attempts to build the prismatic inner region, level by level. Two operations are performed for this purpose: initial strand placement, followed by smoothing of the envelope tessellation based on an elastic spring analogy.

At the end of the smoothing iterations, if a valid mesh is generated with overall quality above a certain threshold, the next level is initialized, using the current envelope surface as the new base surface. Otherwise, for the first two levels, the current level is attempted again, with half the extent. This process is repeated until one of the following situations arises:

- a mesh of the full desired extent is created,
- the maximum number of levels is reached,
- no valid mesh can be created for a level above the first two,
- no valid mesh can be created for the second level, and the level thickness can not be reduced further (based on the stretch ratio), or
- no valid mesh can be created for the first level, and the level thickness can not be reduced further (based on the smallest edge of the surface tessellation).

In the first four cases, a final level needs to be created (outer region), which extends from the outermost prismatic envelope to the desired total extent and is composed of the remaining number of levels to be created. This is done by extruding the strands at each level and performing a Lagrangian vector smoothing on the strands in convex areas. The resulting self-intersections will be removed and handled by the domain connectivity during simulation. Finally, the last step is to subdivide each segmented strand into nlayers, with a thickness distribution matching the user specified (or default) wall and outer spacing. The next few sections describe in more details each algorithm component.

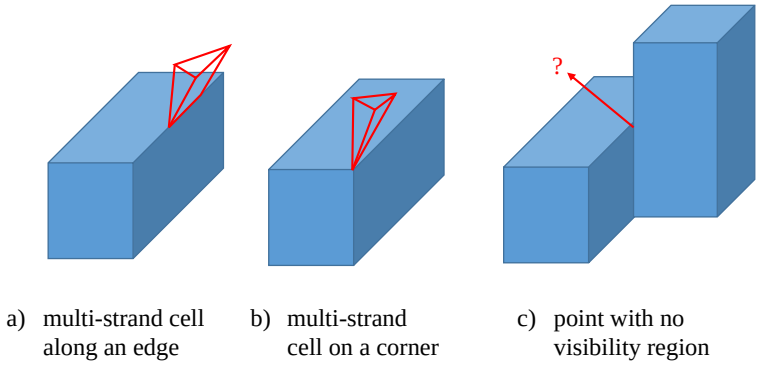


Figure 3: Example of multi-strand cells.

2.5 Multiple-strand determination

For the initial mesh level, in order to improve mesh quality, surface nodes which are located on convex edges can have multiple strands originating from them. This results in the creation of multi-strand cells with a triangular face on the envelope surface and either a segment or a point on the base surface. In the first case, the cell is termed a "wedge", illustrated in Figure 3(a), and in the second case, the cell is a tetrahedron, illustrated in Figure 3(b). Multi-strands are optional if a region of visibility exists for a node, but mandatory if there is no region of visibility, as in the case of Figure 3(c). Contrary to single strands, multiple strands are not constructed directly by point placement on the envelope surface; instead, the original surface mesh topology is first modified to include the multi-strand cells. Initially, multiple strands are assumed to

be coincident with the strand they are a duplicate of, i.e. the multi-strand cells have zero volume. The spring-analogy smoothing algorithm then ensures the multi-strand cells expand to fill the space around the the convex edges. The multi-strand topology is determined as follows: a number of multi-strands is first assigned to all convex edges, computed using the desired maximum number of multi-strands, the thickness of the initial level, and the local cell size. Convex edges are those with the angle between the two neighboring facets that share the edge exceeding a certain value, set to 45^o in the present work. Then, for each node, the number of regions delimited by convex edges is identified. Additional nodes are then created using one of the appropriate methods described below depending on the number of regions:

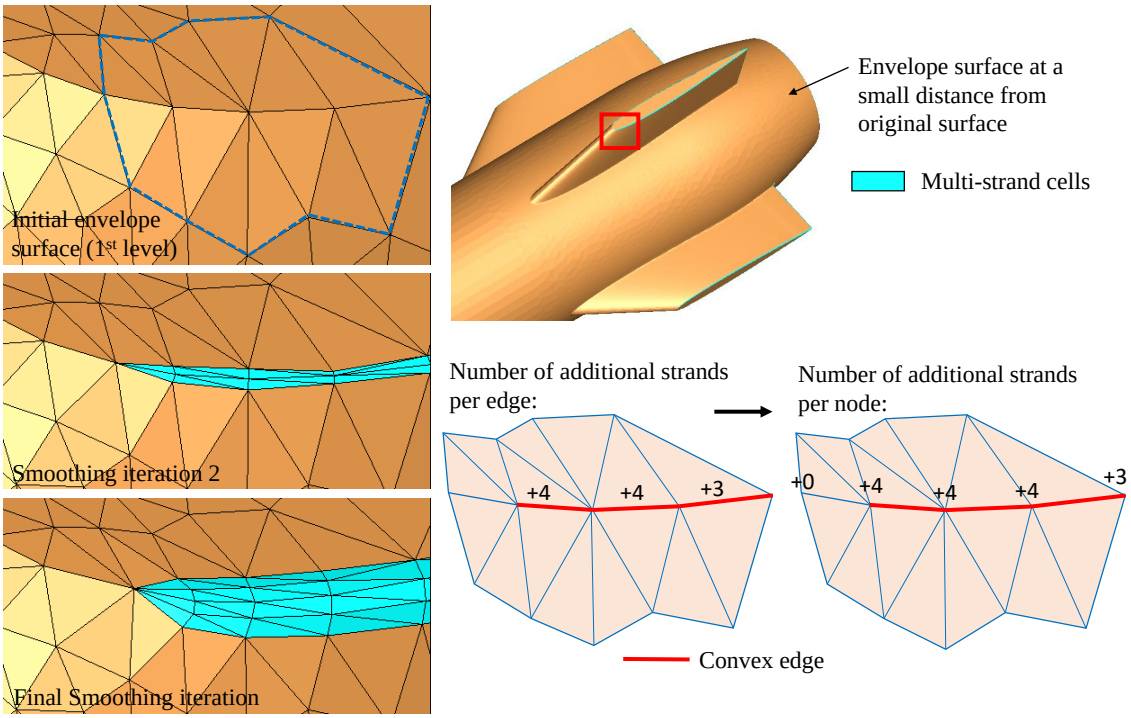


Figure 4: Multi-strand determination along convex edges.

- Case 1: for a node with two regions, the number of additional multi-strands created is set to the maximum value assigned to the two convex edges. Multi-Strand cells ("wedges") are created along convex edges by simply "stitching" the additional nodes created at each edge extremity, creating new facets on the envelope surface as illustrated in Figure 4.
- Case 2: for a node with three or more regions, a 2-D Delaunay tessellation is computed that fills a boundary consisting of a regular polygon with a number of sides equal to the number of convex edges around the node, as illustrated in Figure 5. The number of points on the boundary along each edge of the polygon is set to the multi-strand value of the edge. The total number of multi-strands created is then the number of nodes on the polygon boundary, plus the number of nodes created by the Delaunay tessellation, minus one. The additional multi-strand cells created correspond to the cells of the Delaunay tessellation (all tetrahedra).
- Case 3: for a node with a single region (single convex edge connected), no additional multi-strands are created, unless the node is also part of a concave edge (convex-concave vertex). In that case, all the connected edges that are not concave are treated as if they were convex, but assigned a single additional multi-strand. The vertex ending the convex edge is then surrounded by a number of regions equal to the number of non-concave edges around the node, and the multi-strand determination can proceed as in the second case. This case is illustrated in Figure 6.

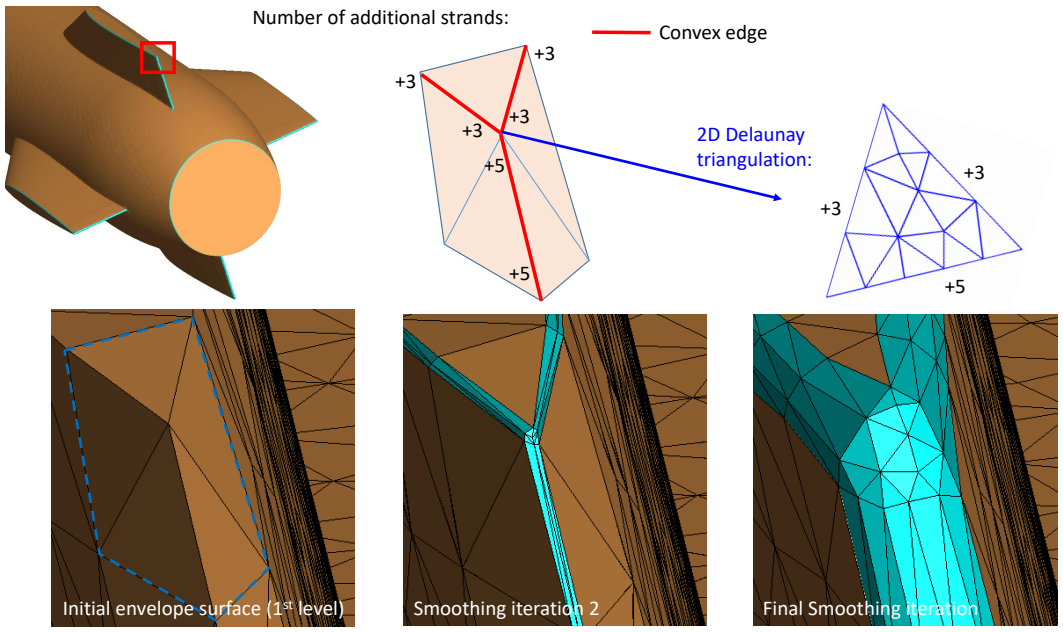


Figure 5: Multi-Strand determination around convex corner vertices.

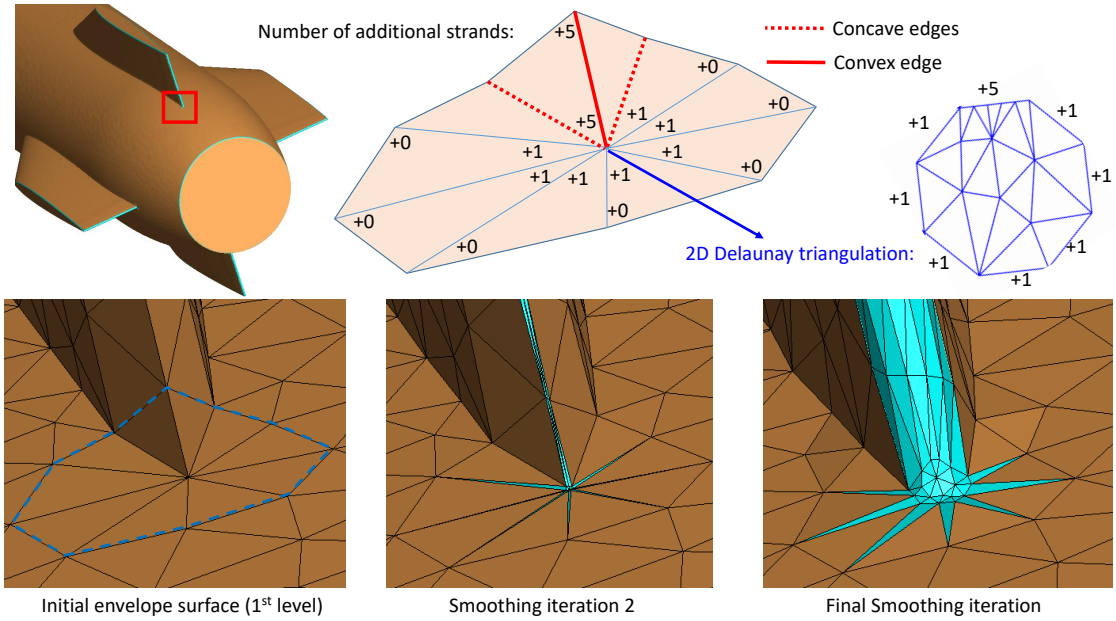


Figure 6: Multi-Strand determination around convex/concave vertices.

2.6 Initial strand placement

Initial strand placement involves finding the initial location of the end points of each strand. Two concepts are used for this purpose, the best visibility direction and the iso-surface of the distance field.

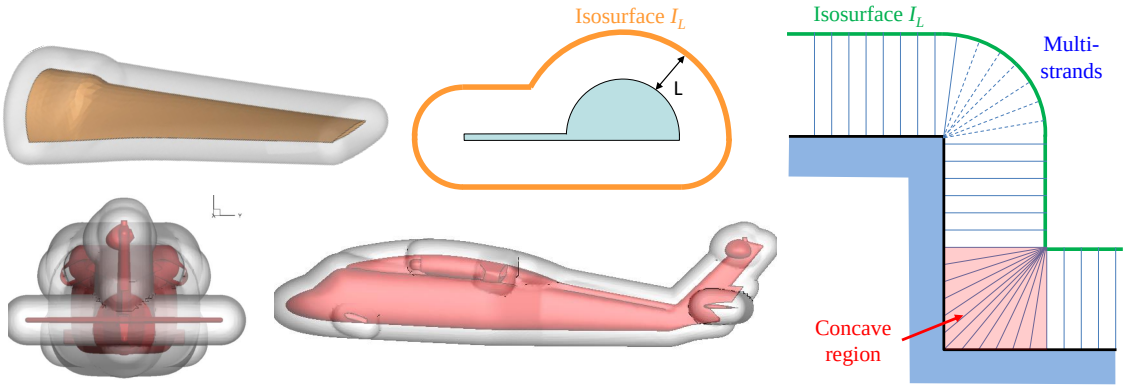
2.6.1 Best visibility direction

In most prismatic mesh generation approaches (advancing front or direct placement), the local normal direction is used for the extrusion of the nodes. The simplest way to compute a local normal direction at each node is by averaging the normals of all the facets that are associated with this node. The simple averaged normal, both unweighted and area weighted, often leads to issues because of the bias in averaging caused by the difference in the number of geometric regions and topological regions that enclose the given node. Aubry et al. [22] proposed a much more robust approach, using the concept of the most normal normal (`MNN`), i.e. the direction that maximizes the minimum angle between the surrounding faces as the optimal direction of choice. This concept was successfully applied by follow-on work by Aubry and Lohner [23] and Loseille and Lohner [14] for boundary layer meshing. The present work makes use of Aubry's method to compute this direction, termed as the best visibility direction.

For convex regions of the surface, an initial strand that follows the direction of best visibility for a distance equal to the current level thickness is appropriate (referred to as the best visibility strand). However, concave areas require a different strategy to avoid immediate local collisions and entanglement. Concave areas of the surface tessellation are identified as the set of all nodes for which the extremity of the best visibility strand is found to be at a smaller distance to the base surface when compared to its length. In general, techniques reported in advancing front literature utilize normal vector smoothing in concave areas followed by merging to form tetrahedra if smoothing fails. In the present work, concave areas use a different point placement technique based on the iso-surface of distance field.

2.6.2 Iso-surface of distance field

The iso-surface of minimum distance at I_L, is defined as the locus of points that are at a given fixed distance L from the discrete surface tessellation $\mathcal{S}$. Examples of minimum distance iso-surfaces are shown in Figure 7(a). The goal of the initial point placement algorithm is to compute point positions on I_L corresponding to each surface node on $\mathcal{S}$, such that the number of invalid elements is minimized.



a) Examples of minimum distance isosurfaces.

b) Strands pointing to closest point on I_L

Figure 7: Choosing a strand vector towards the closest point on isosurface of distance automatically results in desirable strand distribution in concave regions.

As shown in Figure 7(b), the closest vertex to each surface node on I_L is a good candidate for point placement, because it automatically creates a desirable bending of strands in regions of concavity. This point is abbreviated as `CLOVIS`, for **Clo**sest **V**ertex on the **Is**osurface, in the rest of the paper. The number of strands near the concave ridges/corners that bend is directly correlated to the iso-surface distance L, i.e. larger iso-surface distances would cause more strands in a larger region to bend away from the original best visibility direction.

However, the point distribution on the outer envelope surface is not ideal for generating the next mesh level, and a smoothing process needs to be applied to improve the quality of the envelope mesh.

Another problem associated with the CLOVIS solution is illustrated in Figure 8(a). For geometries that feature large variations in effective body thickness, the CLOVIS direction could lead to undesirable consequences such as penetration of the surface. Therefore, it is also important to ensure that the best visibility direction is used if the CLOVIS direction is outside the region of visibility of each surface node, as defined in the next section.

2.6.3 Limited region of Visibility

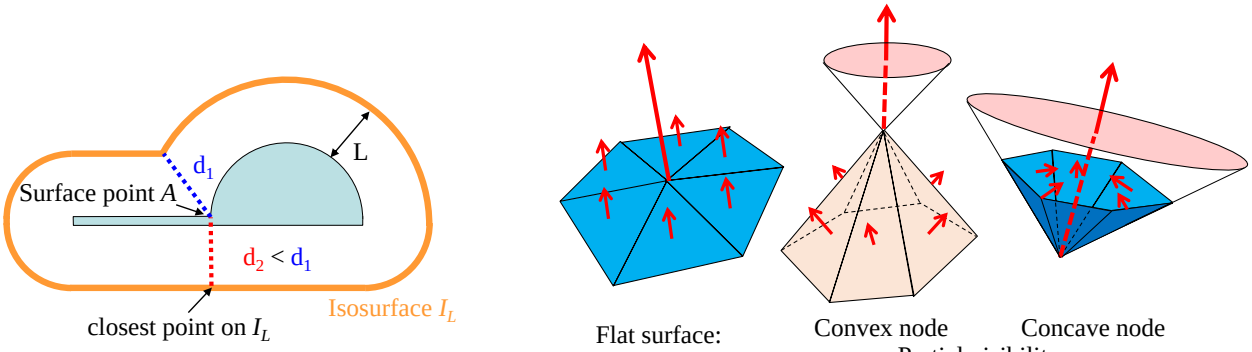
The region of visibility for a surface vertex is the region the strand vector can occupy such that all triangles connected to the surface vertex have non-zero areas when viewed along that vector towards the surface. A strand is in the (full) visibility region if all the dot products of itself with each of the neighbor face normals are positive. Since it is desirable that the strands do not get too close to the visibility boundary, a limited region of visibility constraint is enforced, using a maximum deviation input, typically set to $d_{max} = 80\%$, corresponding to the maximum allowed deviation from the direction of best visibility to the visibility boundary. A strand is in the limited visibility region if for all face neighbors, the dot product of the unit vector along the strand with the neighbor face normal is greater than a threshold value dp_{min}, defined as:

$$dp_{min} = \cos\left(\frac{\pi}{2} - (1 - d_{max})\alpha_{vis}\right) \tag{1}$$

where α_{vis} is the visibility angle, defined as the complementary angle to the maximum angle between the best visibility direction and any neighbor face normal:

$$\alpha_{vis} = \frac{\pi}{2} - \max_{k=1\ldots n} \arccos\left(\hat{b} \cdot \hat{n}_k\right) \tag{2}$$

where $\hat{b}$ is the direction of best visibility, and $\hat{n}_k$ is the face normal unit vector. For example, a vertex with 90 deg visibility is on a perfectly flat region, while a vertex with 0 visibility may be on a convex/concave corner with a visibility problem, requiring multi-strands to enable meshing of the initial level. The cone of visibility is the more restrictive region composed of all strands such that the dot product of the strand unit vector and the best visibility direction is more than the cosine of the visibility angle. Figure 9 illustrates these different concepts for an example mesh vertex located along a convex edge. During smoothing, the motion of a vertex on the envelope surface is constrained to remain within the limited region of visibility, which is the green area consisting of the intersection of all the local visibility cones for each neighboring face.



a) If the volume mesh features abrupt thickness variation, the closest point on the isosurface approach can fail.

b) Visibility regions that constrain the strand vector for creation of a valid strand cell.

Figure 8: The CLOVIS solution can lie outside the visibility region.

2.6.4 Initial strand placement

The algorithm for initial strand placement is summarized in pseudo code listing 1 in the Appendix. All nodes initially create the local best visibility strands. Next, the concave regions can be identified, by computing the distance to the base surface for all strand end points. For nodes in concave regions, the `CLOVIS` strand is also computed. The limited region of visibility check is then performed. If it fails, those strands revert back to the best visibility solution.

Application of these steps ensures that the initial point placement yields a mesh that is valid and of fair quality at most regions other than complex concave/convex corners and thin body intersections (e.g. trailing wing/fuselage interface). A constraint-driven smoothing algorithm (described in Section 2.7 is applied to correct these problems and improve the quality of the mesh.

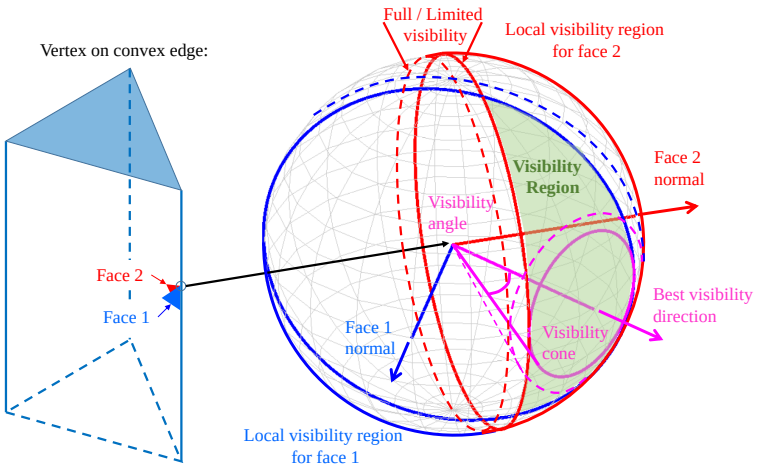


Figure 9: Visibility angle, cone, and regions for a vertex on a convex edge.

2.6.5 Closest Vertex on the Iso-Surface (`CLOVIS`) algorithm

Given a tessellated surface, $\mathcal{S}$, the isosurface of the distance field at L is defined as:

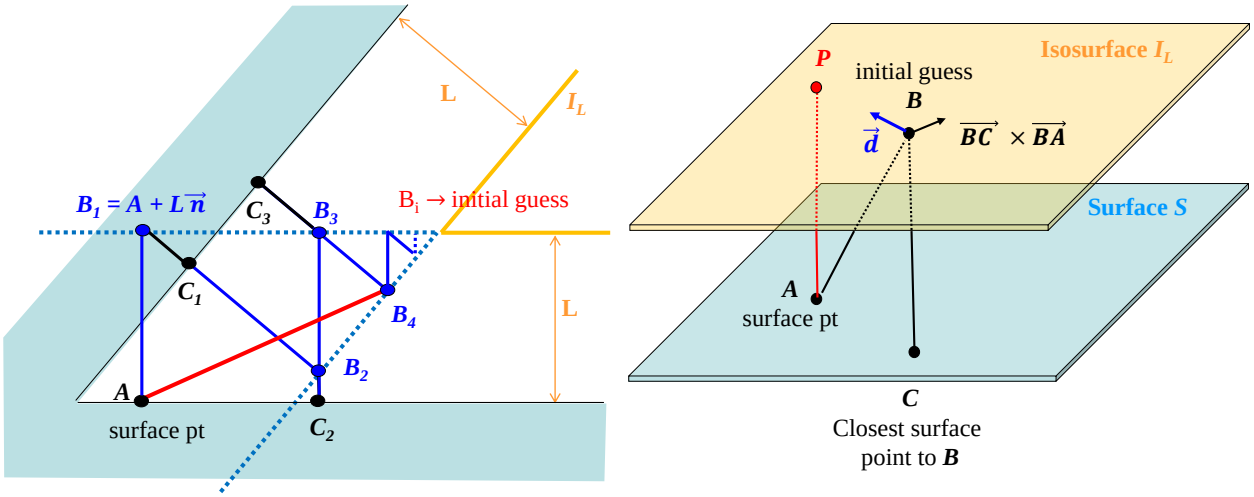
$$I_L = \{P \in \mathbb{R}^3 \mid \texttt{MinDist}(P, \mathcal{S}) = L\}, \text{ where } \texttt{MinDist}(P, \mathcal{S}) = \min_{A \in \mathcal{S}} \|\overrightarrow{AP}\| \tag{3}$$

For any surface vertex A, the closest point P on the isosurface at a distance L is any point that satisfies:

$$\begin{cases} \texttt{MinDist}(P, \mathcal{S}) = L \\ \|\overrightarrow{AP}\| = \min\limits_{MinDist(M,\mathcal{S})=L} \|\overrightarrow{AM}\| \end{cases} \tag{4}$$

The optimization process entails sliding the end point of the vector (P) on the iso-surface and locating it such that the segment AP has the shortest length. The fact that the isosurface of distance field is only known implicitly by its mathematical description makes the solution of Eq 4 challenging. A discrete solution to the continuous optimization problem can be obtained by constructing an approximate tessellated isosurface using a marching-cube method [24]. However, this approach was found to lack robustness and computational efficiency. Instead, an algorithm was designed to efficiently compute, for each surface vertex, the closest point on the actual analytical description of I_L shown in Eq 3. This method is referred to as the `CLOVIS` algorithm. The method makes intensive use of an efficient routine to compute $\texttt{MinDist}(P, \mathcal{S})$, which returns the shortest distance and the location of shortest distance to the surface $\mathcal{S}$ from a point P. Since surface $\mathcal{S}$ is known explicitly, this is an easier operation that can be efficiently accomplished using an Alternating Digital Tree (ADT) approach [25]. Note that this operation is identical to the computation of minimum wall distance fields for turbulence models, for which there are several well-established approaches documented in References [26, 27, 28].

The success of an optimization problem depends strongly on an initial guess that satisfies the solution constraint and is sufficiently close to the final solution. For surface nodes in concave regions, the end point of the best visibility strand lies below the iso-surface and hence does not satisfy the solution constraint. Extending the vector until it meets the iso-surface can fail if the initial vector direction is nearly tangent to the iso-surface I_L. Therefore, a geometrical construction algorithm listed in pseudo code listing 2 is used to march iteratively until an intersection with the iso-surface is obtained. The same algorithm is also illustrated in Figure 10(a).



a) Step 1: initial guess for closest point on isosurface of distance.

b) Step 2: marching towards the closest vertex on the isosurface of distance

Figure 10: `CLOVIS` algorithm : initial guess estimation and descent towards the optimum.

Once a initial guess B is obtained, the optimization is performed by walking on the iso-surface in a specific search direction. As shown in Figure 10(b), the search direction vector is chosen such that it is tangent to the isosurface and lies in a plane formed by the current value of B, the closest point C to B on $\mathcal{S}$, and the surface query point A, i.e. triangle ABC. The algorithm for `CLOVIS` is summarized in pseudo-code listing 3. The convergence of the `CLOVIS` algorithm is shown in Figure 11. In general, only 20 steps are required to achieve machine zero convergence of the optimization problem. It is also worth noting that the algorithm is embarrassingly parallel, since no neighbor information is required and each surface node can find its strand end point independent of each other. This feature makes this algorithm easily amenable to distributed and multi-threaded computing.

2.7 Mesh smoothing, constrained to the iso-surface, using spring analogy

To improve the mesh quality, a smoothing algorithm is applied, which is loosely based on a linear spring analogy with constraints enforced so that strand end points remain on the iso-surface of distance.

In this method, each edge on the envelope surface is treated similar to a linear spring, as illustrated in Figure 12(a). The spring rest length is zero and a dual-value stiffness is used, such that the springs are in equilibrium when the strand length tends to zero. This is because the initial layer may need to be very thin for complex bodies, in which case the envelope mesh formed by the strand extremities should be nearly identical to the surface mesh.

2.7.1 Edge stiffness determination

The stiffness value to satisfy force equilibrium at the surface is computed directly using the fact that for any 2D polygon, the sum of the vectors normal to each edge (pointing outwards) is zero when weighted by the edge length. The cells around each vertex are first projected onto a plane tangential to the isosurface

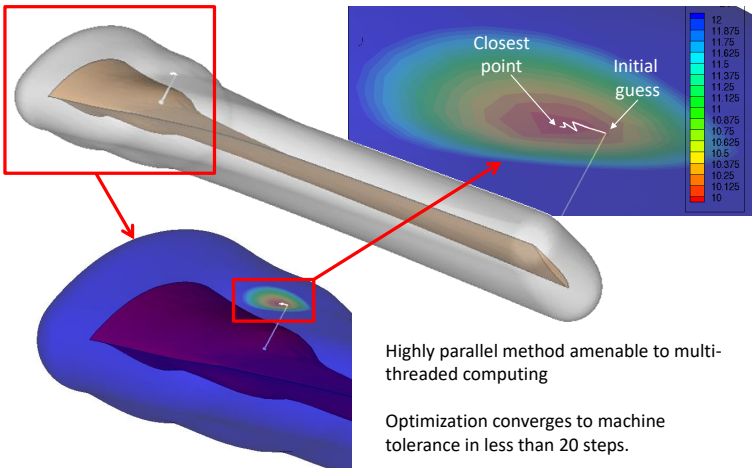


Figure 11: Convergence pattern of the `CLOVIS` algorithm.

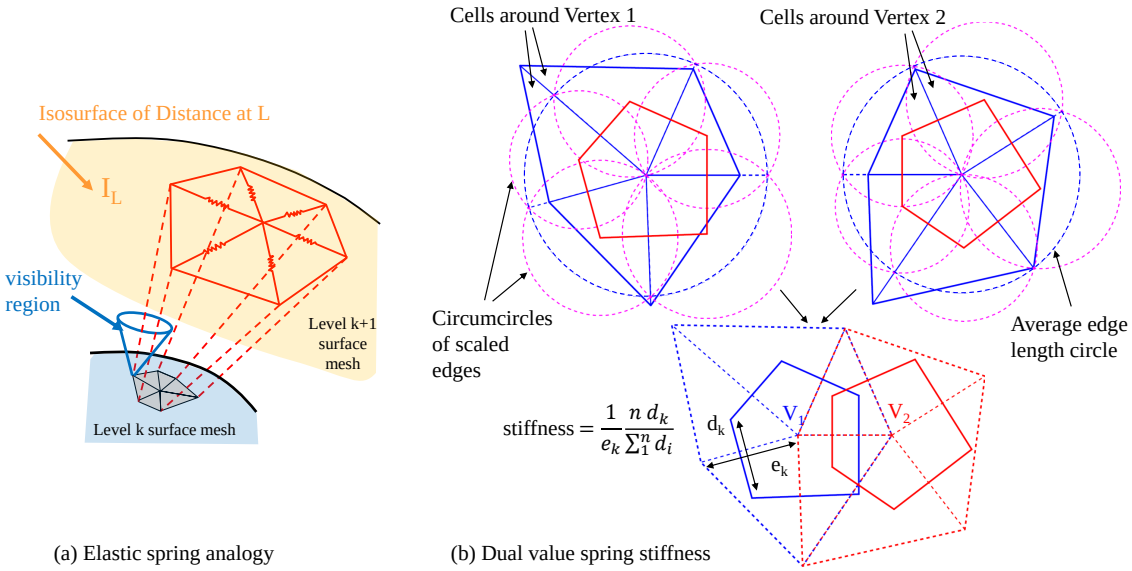


Figure 12: Envelope mesh smoothing using elastic spring analogy.

of distance. Note that the during the smoothing process, the direction of the total force is constrained to remain in this tangential plane. At the limit of zero distance to the surface, this plane can be approximated as the plane normal to the direction of best visibility, when that direction exists, or normal to the direction of the closest vertex on the isosurface otherwise. Then, a polygon is formed around the vertex by joining the centers of the circumcircles of each neighboring cell, whose edges are first scaled to the average edge length around the node. This is illustrated in Figure 12(b). In this way, the outward normals of the polygon edges are by design along the edges surrounding the nodes.

The force for a vertex surrounded by n edge vectors $\vec{e_k}$, k=1..n is computed as:

$$\vec{F} = \sum_{k=1}^{n} S_k \vec{e_k} \tag{5}$$

where S_k is the stiffness of the kth edge around the vertex. Force equilibrium is satisfied if we choose the

stiffness value:

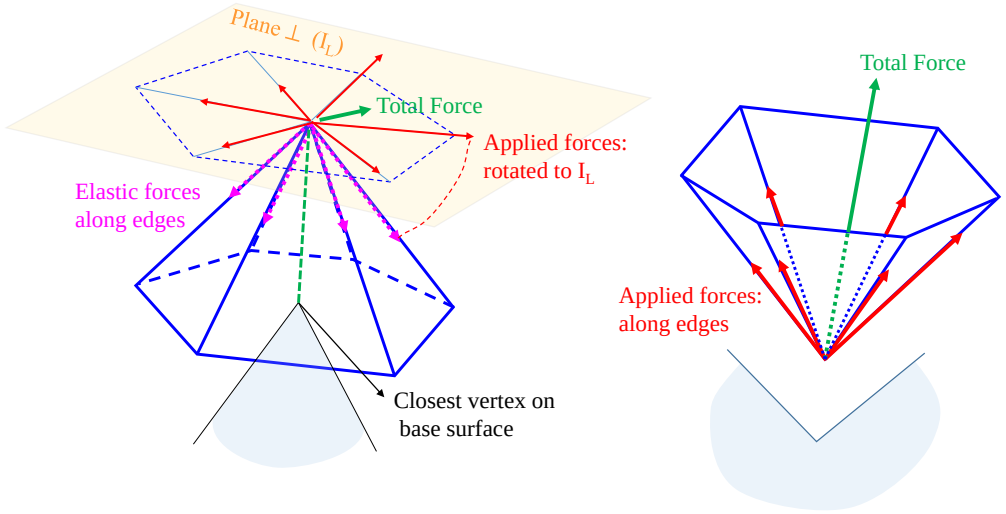
$$S_k = \frac{1}{||\vec{e_k}||} \frac{d_k}{\frac{1}{n}\sum_{i=1}^{n} d_i} \tag{6}$$

where d_k is the length between the centers of the two neighboring circumcircles. This method results in each edge being assigned two different stiffness values, one for each edge extremity. If the cells are equilateral triangles, the stiffness is identical for all edges and equal to the inverse of the edge length.

2.7.2 Elastic force computation

The elastic force on each strand end node is applied differently depending on the node position characteristics, which includes whether the node is in a convex region or concave region and located above or under the isosurface of distance.

- Case 1: if the node is in a convex region, the force magnitude along each edge is computed using equation 5, but it is applied in the edge direction projected on the isosurface (normal to the segment joining the node to its closest point on the base surface). This is illustrated in Figure 13(a).
- Case 2: if the node is in a concave region, the elastic force will tend to make the node move above the isosurface. Since this is desirable for reducing concavity for the next strand level, if the strand is above the isosurface, the force for each connected edge is applied directly in the direction of the edge, as illustrated in Figure 13(b). If the strand is at or below the isosurface, however, the force component normal to the isosurface and toward the surface is first removed.



(a) Elastic Forces on a Convex Vertex: Constrained to isosurface of distance

(b) Elastic Forces on a Concave Vertex: No constraint

Figure 13: Method for applying elastic forces for convex or concave vertices.

2.7.3 Constrained point motion

After the elastic force vectors are computed, the position of the strand end nodes are updated using a simple explicit time-marching scheme, with constraints applied to ensure all points remain on or above the isosurface of distance and within their respective region of visibility.

Walk length constraint

First, the walk length is computed using the maximum ratio of force magnitude and average edge length, r_{max}. The nominal walk (without constraints) between $\vec{p}^{\,i}$, the position vector at iteration i to $\vec{p}^{\,i+1}$, the

position vector at iteration $i+1$ is then set to:

$$\vec{p}^{\,i+1} = \vec{p}^{\,i} + \frac{0.2}{r_{max}} \vec{F}^{\,i} \tag{7}$$

so that no vertex walks more than 20% the average length of its surrounding edges. The walk length is also further limited so that it is less than 20% in any surrounding edge direction.

Visibility constraint

At the initial iteration, all strands are by design within the visibility region, except strands emanating from vertices with no visibility (requiring multiple strands), which follow the direction to the closest vertex on the isosurface.

For single strands, the visibility constraint is imposed using the visibility region, which is the intersection of the limited visibility cones for each neighbor face (section 2.6.3). For each face, if the walk exits the local visibility cone, the walk vector is first modified by projecting it to the plane tangent to that cone. If the walk still exits the visibility region, the walk vector is shortened until the new point location is on the (limited) visibility boundary.

For multi-strands, a visibility constraint is also applied, but following a different approach. Multi-strands are allowed to move outside the region of visibility of their original node. In fact, the smaller the visibility angle is, the larger the allowed deviation is from the initial strand direction. Multi-strands are constrained to remain within a cone centered on the initial strand direction, and with an angle equal to the complementary angle of the visibility angle, if it exists (for vertices with no visibility, no visibility constraint is applied).

Isosurface of distance constraint

Finally, the constraint that all vertices on the envelope surface remain on or above the isosurface of distance is enforced. If the corresponding base surface point is in a convex region, this is done by moving the point along the local normal to the isosurface (vector joining the point to its closest point on the surface) until it is located on the isosurface.

For base points in a concave region, this operation is performed only if the strand end point is located below the isosurface and the distance to the surface is decreasing. Otherwise, the point is allowed to move above the isosurface, but the strand length is limited so that the distance to the surface does not exceed a limit L_{max} set to:

$$L_{max} = L_k \left(1 + \frac{1}{2}(1 - f^2)\right) \tag{8}$$

where L_k is the thickness of the current strand level and f is the ratio of L_k by the initial strand length (less than one for concave regions).

2.7.4 Smoothing termination based on mesh quality

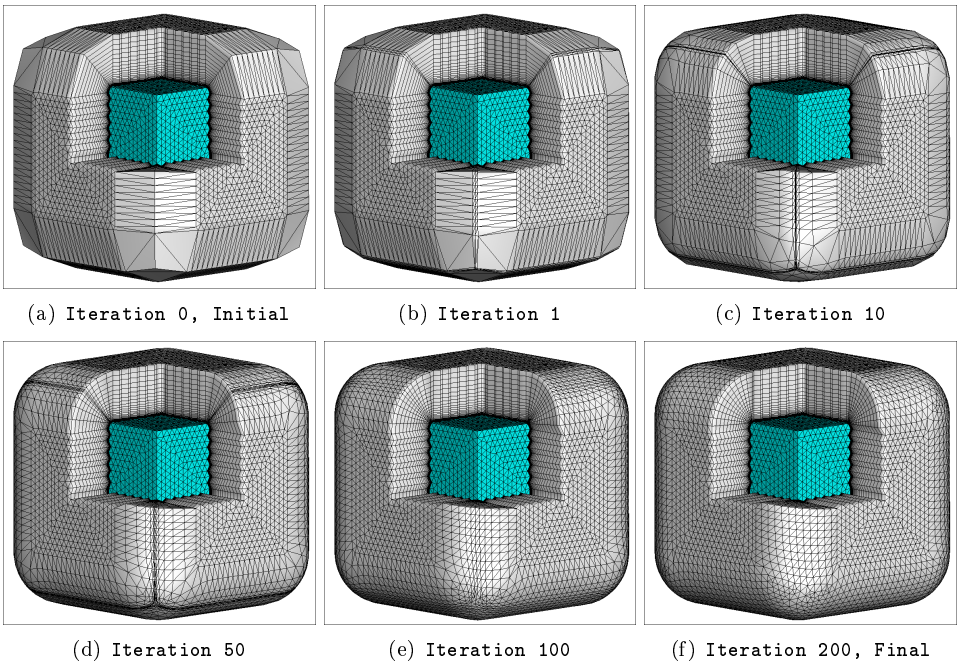
The smoothing iterations are terminated if one of the following three situation arises:

- the maximum walk length for the entire mesh is below a threshold value,
- the maximum number of iterations is reached (set to 200 in this work), or
- the envelope quality could not be improved in the last 20 iterations, and the mesh is valid.

The mesh is considered valid if all prisms have a quality metric above a certain threshold, set to 0.2 in the present work. The prism quality is defined as the minimum dot product of any strand vector with the bottom (base surface) or top (envelope surface) normal vector. The envelope quality is defined as the maximum ratio of cell areas around any node on the envelope surface tessellation.

After termination, the valid mesh with the largest envelope quality is retained for use in the next strand level. Figure 14 shows the evolution of strands and the prismatic envelope around a simple cube geometry.

Note that the multi-strands emanating from a node are initially co-incident. The strands spread and converge to their final location with smoothing iterations.



(a) `Iteration 0, Initial` (b) `Iteration 1` (c) `Iteration 10`

(d) `Iteration 50` (e) `Iteration 100` (f) `Iteration 200, Final`

Figure 14: Evolution of multi-strand mesh system with smoothing iterations.

3 Description of Flow Solvers Used

All of the strand/Cartesian simulations are performed with the Helios [29] framework with `mStrand` as the near-body solver and `SAMCart` as the off-body solver. Traditional unstructured grids are also generated for comparison purposes. In this case, Helios uses either FUN3D [30] or NSU3D [31] as the near-body solver to model the near-wall flows.

3.1 Multi-strand near-body solver (mStrand)

The Reynolds-averaged Navier-Stokes (RANS) equations in a general moving coordinate system in three dimensions are solved in `mStrand`. Turbulence closure is accomplished with the negative Spalart-Allmaras (SA) model [32, 33]. A fully parallel implementation of `mStrand` is obtained by partitioning the envelope surface mesh (the surface that forms the outer-boundary of the strand mesh) into contiguous blocks on the basis of surface elements using `METIS` [34]. The strand grid spatial discretization is based on a vertex-centered finite volume approach, where a dual-cell is constructed around each grid point. The strand solver accommodates both quadrilateral and triangular prisms depending on the surface topology and can handle general prismatic mesh in the normal (strand) direction. More details of `mStrand` can be found in Lakshminarayan et al. [35].

3.2 Cartesian Off-body Solver (SAMCart)

A structured adaptive solver SAMCart is used for the Cartesian off-body grid. The parallel mesh adaptive capability is provided by the SAMRAI [36] library and the solution in each block is obtained using a solver called Cart. The Cart solver uses a high-order central differencing scheme - 6th order with 5th order dissipation for the inviscid terms and 4th order for the viscous terms. Cart implements the Spalart-Allmaras turbulence model and can also enable Detached Eddy Simulation (DES) capability. The solver includes an implicit second order BDF2 time integration scheme with Lower-Upper Symmetric Gauss-Seidel (LU-SGS)

and diagonalized Alternating Direction Implicit (ADI) operators. Further description of SAMCart can be found in Wissink et al. [29].

4 Results and Discussion

4.1 Geometries with moderate complexity

Two generic geometries, a business jet aircraft and a missile, are considered for evaluation of mesh and solution quality and comparison with traditional unstructured mixed element mesh generation methods. The business jet geometry is shown Figure 15. There are multiple concavities and concave/ convex corners in this geometry. Meshes are generated using the present methodology (CLOVIS + smoothing), a mesh generator from Mississippi State University–Advancing Front and Local Reconnection method (AFLR3 [37]) and the commercial software Pointwise, which uses its T-Rex algorithm to construct prism layers from anisotropic tetrahedra [38]. The prismatic envelope of the mesh is shown for each case. Both CLOVIS and AFLR3 produce a regular surface, while Pointwise, owing to its tetrahedral merging technology, produces an irregular surface with differing number of prism layers around the geometry. Both AFLR3 and Pointwise show some amount of prism layer collapse near the concave regions. The use of minimum distance iso-surface does yield meshes with increased smoothness, that do not collapse in the concave corners. It is important to note that we did not adjust any of the standard parameters in AFLR3 and Pointwise and it may be possible to obtain improvements to the prismatic envelope using another set of parameters. Figure 16 shows the flow solution(s) obtained using each of these grid systems. Unstructured grid solutions are obtained using the NASA FUN3D [30] code and strand/Cartesian solutions are obtained using the Helios framework. There is good overall agreement among the computed solutions. Detailed evaluation shows that there are observable differences in the concave corner areas at the rear of the pylon attachment. The pylon attachment has a sharp trailing-edge and a small amount of camber and hence behaves very similar to a wing. Both the strand and AFLR3 grids capture of the rear stagnation point, which is absent in the solution using Pointwise grids. Solution residuals converged by 6 orders of magnitude for all of these grids. Therefore, it is more likely that the differences observed here are because of the differences in grids rather than lack of convergence of the flow solution. Figure 17 show similar grids generated for the missile geometry. The missile has four fins with sharp, zero-thickness trailing-edges. There are concavities present at the base of the fins, and also concave/convex corners present at the intersection of the fin trailing-edge and the body. The fins also have a sharp rectangular tips, with only one triangle straddling the tip towards the trailing-edge. The prismatic envelope shows significant differences between CLOVIS, AFLR3 and Pointwise methodologies. First, the use of multi-strands provides much improved coverage and preserves orthogonality of prism layers compared to the single vector approach used by AFLR3 and Pointwise. Second, the prismatic envelope shows irregularities in both AFLR3 and Pointwise methodologies, while CLOVIS because of its use of the minimum distance iso-surface, shows a regular and smooth envelope. Finally, while prismatic envelope collapse can be noted with AFLR3, it is not present with Pointwise or CLOVIS. Figure 18 shows pressure contours around the missile geometry. As in the business jet case, good overall agreement can be noted between all three mesh systems and associated calculations. Detailed evaluation again shows observable differences. The AFLR3 mesh system shows substantial pressure oscillations towards the trailing-edge of fin, which are absent in the CLOVIS mesh system and present to a smaller extent in the Pointwise mesh system. Note that both CLOVIS and Pointwise generate meshes that are smoother towards the fin tip when compared with AFLR3 and the pressure oscillations are attributable to the lack of smoothness in the AFLR3 mesh.

4.2 Full aeroelastic calculations for the UH-60A aircraft

This test cases encompasses all the complexities that are encountered in a realistic simulation of rotorcraft forward flight. All the strand volume meshes used in this calculation were auto-generated using the prismatic mesh generation technique. The mesh system, shown in Figure 19 has boundary surfaces with strong convex, concave and concave-convex features. Self-intersecting meshes are trimmed to create optimal overlap through an efficient domain connectivity procedure [18]. As shown in the grid details in Figure 19(b-d), concave features are adequately resolved using prismatic meshes and the self-intersections at tight corners are appropriately removed to create an optimal overlap suitable for overset gridding.

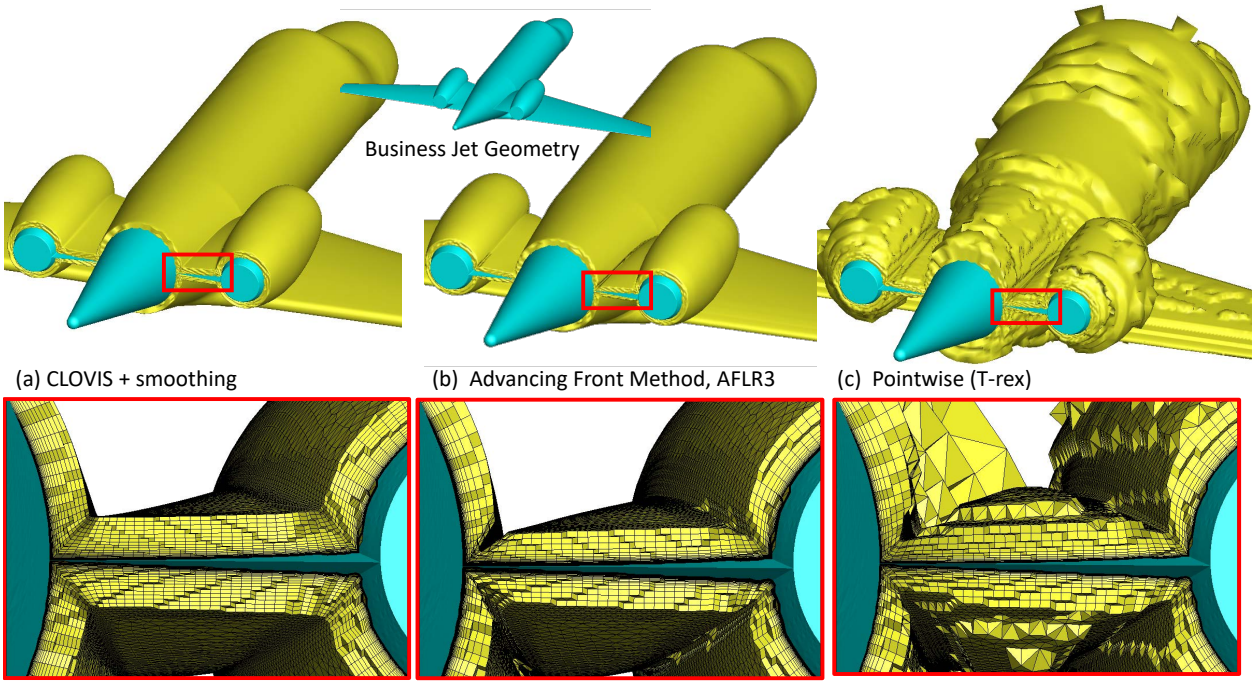


Figure 15: Grid details for the business jet geometry.

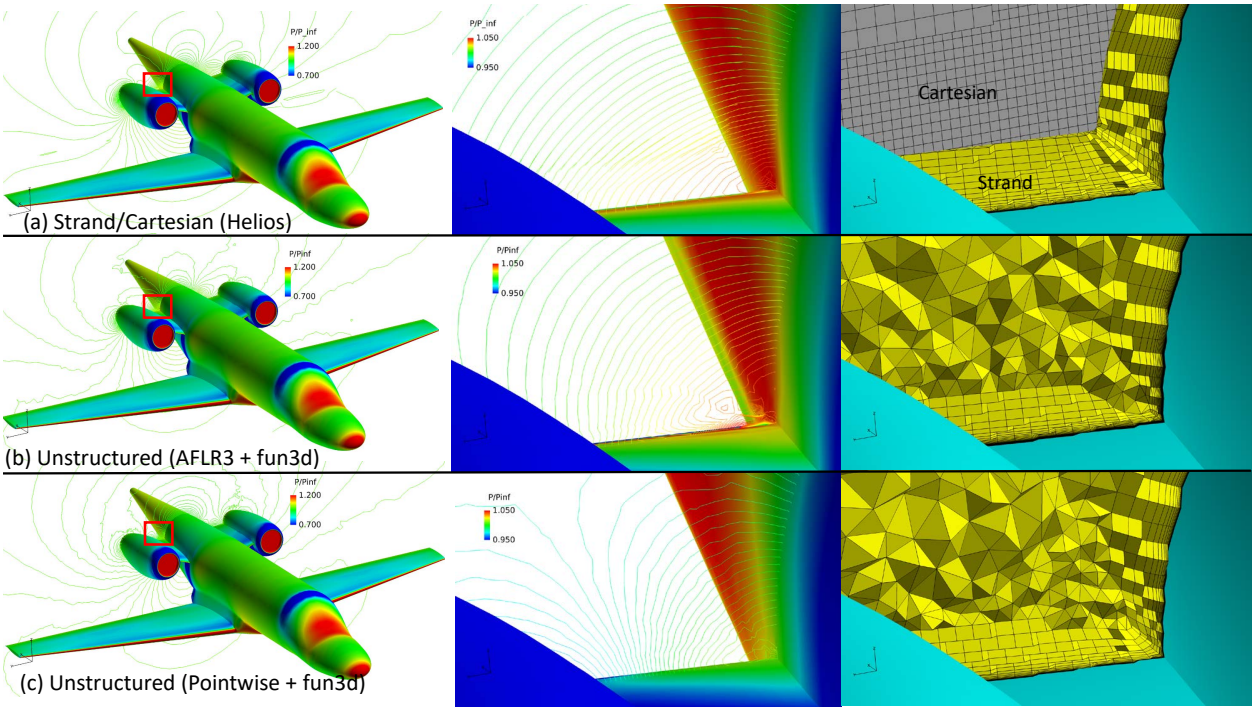


Figure 16: Flow solutions for the business jet geometry, comparing strand/Cartesian and unstructured mixed element grids.

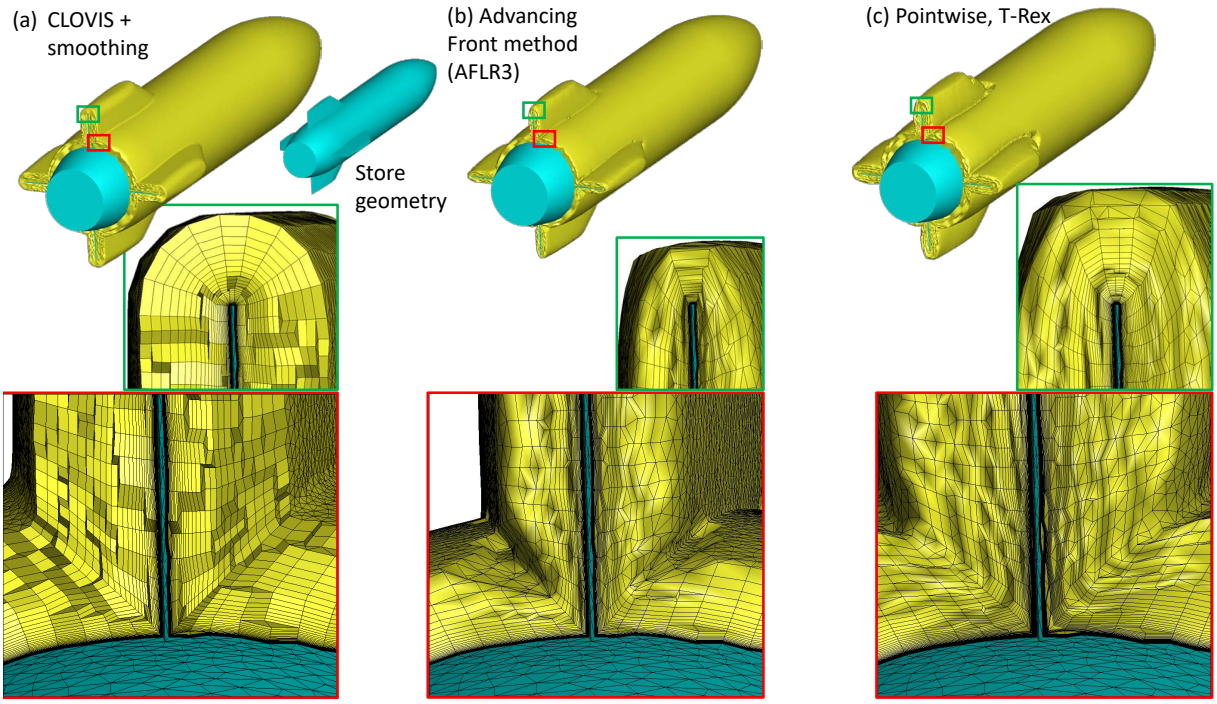


Figure 17: Grid details for missile geometry.

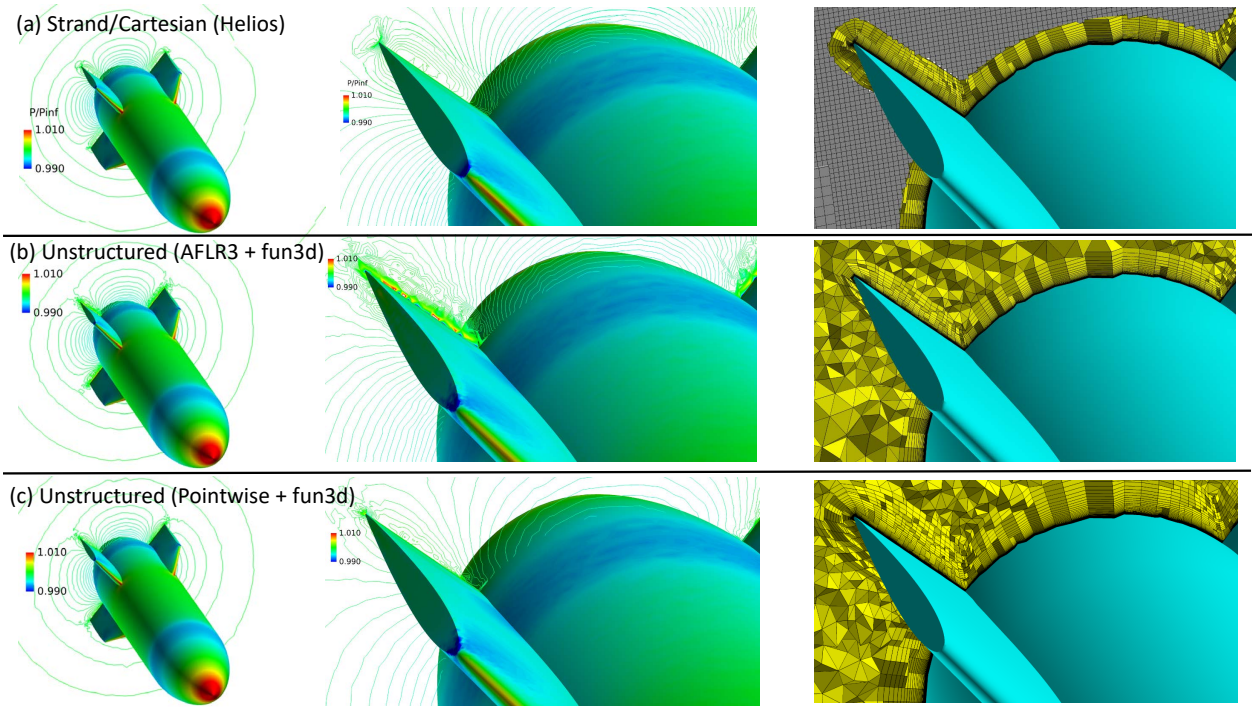
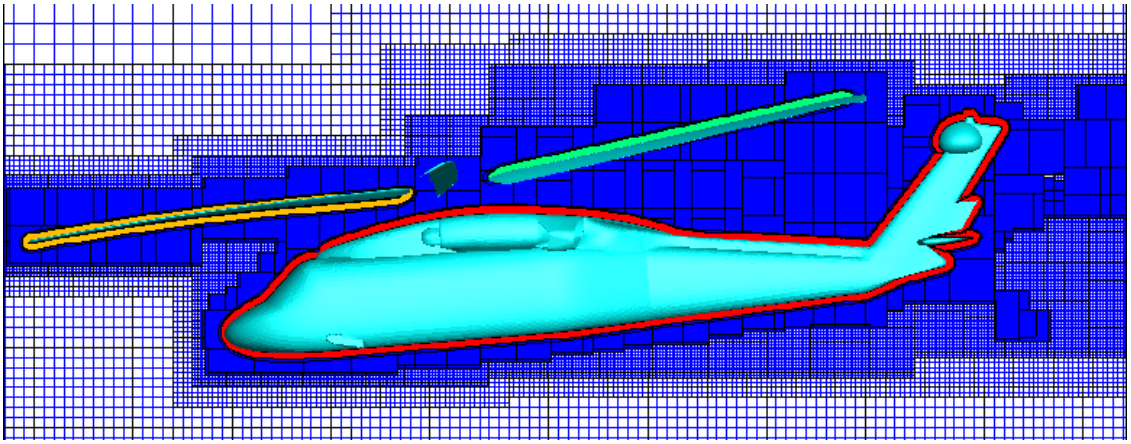
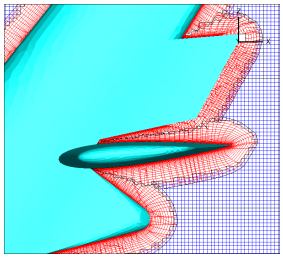


Figure 18: Flow solutions for missile geometry, comparing strand/Cartesian and unstructured mixed element grids.

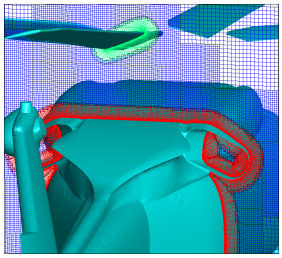
The calculation is performed for the high speed forward flight condition of the UH-60A aircraft. The rotor blades dynamically deform based on the aeroelastic response of the underlying blade structural model. The Rotor Comprehensive Analysis System (RCAS [39]) code framework performs the computational structural dynamics (CSD) and rotorcraft trim calculations. Exchange of loads and displacements between the CFD and CSD and mesh deformations according to these displacements are facilitated through the conservative MELODI [40] interface. Figure 20 shows comparisons between computations using strand/Cartesian grid systems and traditional unstructured/Cartesian grids system. Good agreement can be noted in both the surface C_p and Mach number field contours. Figure 21 compares the predicted aerodynamic loading



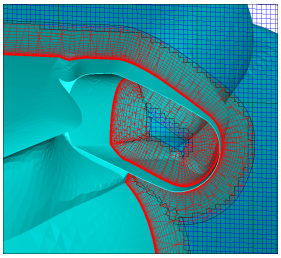
(a) Overall multi-mesh system



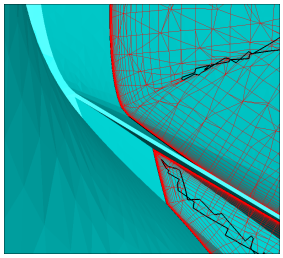
(b) Horizontal tail detail



(c) Rear view with section along inlet



(d) Inlet detail



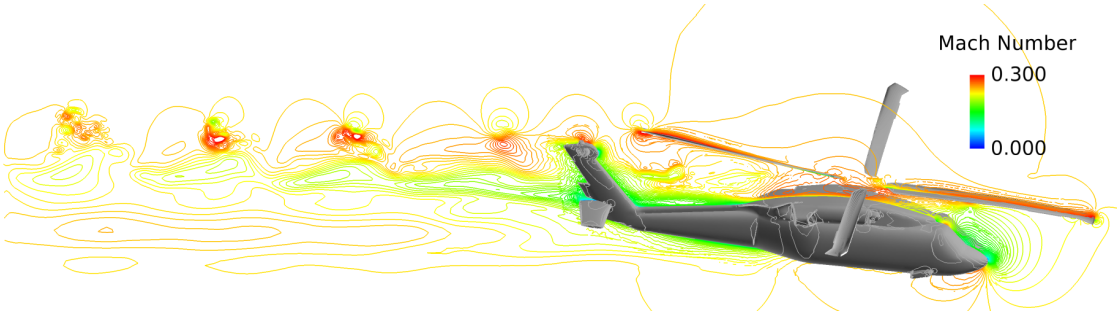
(e) Inlet Corner grids

Figure 19: Strand and Cartesian meshes generated for UH-60A blackhawk helicopter forward flight test case.

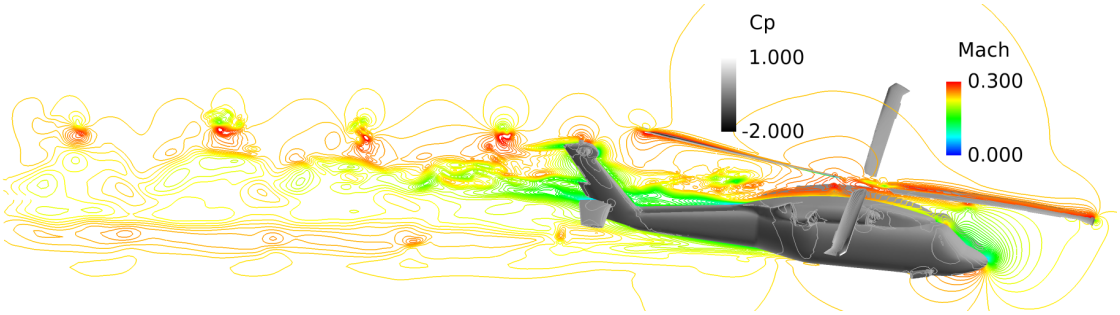
with corresponding measurements obtained from the UH-60A flight test campaign [41]. Good agreement can be noted in both the phase and magnitude of the sectional normal force, chord force and pitching moment variations when compared with the flight test measurements. Furthermore, the strand grid based predictions are seen to be in excellent agreement with corresponding predictions that used fully unstructured mixed element meshes for the near-body.

4.3 NASA High Lift Common Research Model (HL-CRM)

The NASA HL-CRM is a wing-body high lift system in nominal landing configuration, with slat and flaps deployed at 30^o and 37^o respectively, without nacelle, pylon, tail and support brackets. This configuration has various geometric complexities that include highly convex thin trailing-edges that intersect the body to form concave-convex features, small gaps between different bodies and socket type features on the main body that house the undeployed flaps and slat. The auto-generated strand/Cartesian mesh system for this geometry is shown in Figure 22, where the overall configuration and insets of mesh system near features such as the slat/body gap, flap/body gap, and flap/flap gap are portrayed. Multi-strand meshes, as shown in the blunt trailing-edge inset, are generated for all bodies and provide better mesh resolution and orthogonality



(a) Unstructured/Cartesian mesh based solution



(b) Strand/Cartesian mesh based solution

Figure 20: Flow solutions for the UH-60A high speed forward flight (advance ratio, $\mu = 0.368$ and blade loading, $\frac{C_w}{\sigma} = 0.0783$) problem obtained by the Helios [29] framework using Unstructured/Cartesian and Strand/Cartesian mesh systems.

that improves the accuracy of capturing the shear-layers that are transported from one body to the other in the high lift flow scenario. Figure 23 shows the velocity magnitude and pressure coefficient contours on the high lift wing. Contours show smooth transition between the various mesh system and can be observed to be fairly continuous across the overset interfaces, indicating good accuracy of the intra-mesh (self-intersections) and inter-mesh (between bodies and the off-body mesh) overset connectivity operations. Figure 24 shows the pressure coefficient variation at 5 chosen span stations from the strand/Cartesian computations and their comparison with peer computations using unstructured/Cartesian mesh system with both NSU3D and FUN3D used as the near-body solvers on the unstructured volume mesh. Good agreement can be noted with all sets of computations, underscoring the ability of auto-generated strand/Cartesian mesh system to capture results with the same level of accuracy as an unstructured mesh system.

5 Concluding Observations

Anisotropic prismatic mesh generation is required for simulation of high Reynolds number viscous flows. Automation of prismatic mesh represents a large stride towards improved usability and productivity of CFD simulations and the work presented here contributes directly to achieving this goal. Specifically, we designed, developed, implemented and tested a new methodology that uses the minimum distance field as a guide for generating a prismatic envelope of a given surface. A set of algorithms were designed for topology changes to accommodate multiple vectors per node (multi-strand), point placement in concave areas and constrained smoothing using an elastic analogy. The point placement and constrained smoothing algorithms are iterative in nature and amenable to highly parallel execution. Results obtained using the automatically generated prismatic meshes show that: (1) Prismatic envelopes generated are similar or better in quality to those obtained using widely available advancing front/tetrahedral merging methods and (2) Good quality aerodynamic predictions are obtained when using the auto-generated strand/Cartesian mesh system. Predictions show good agreement with measured data and excellent agreement with comparable

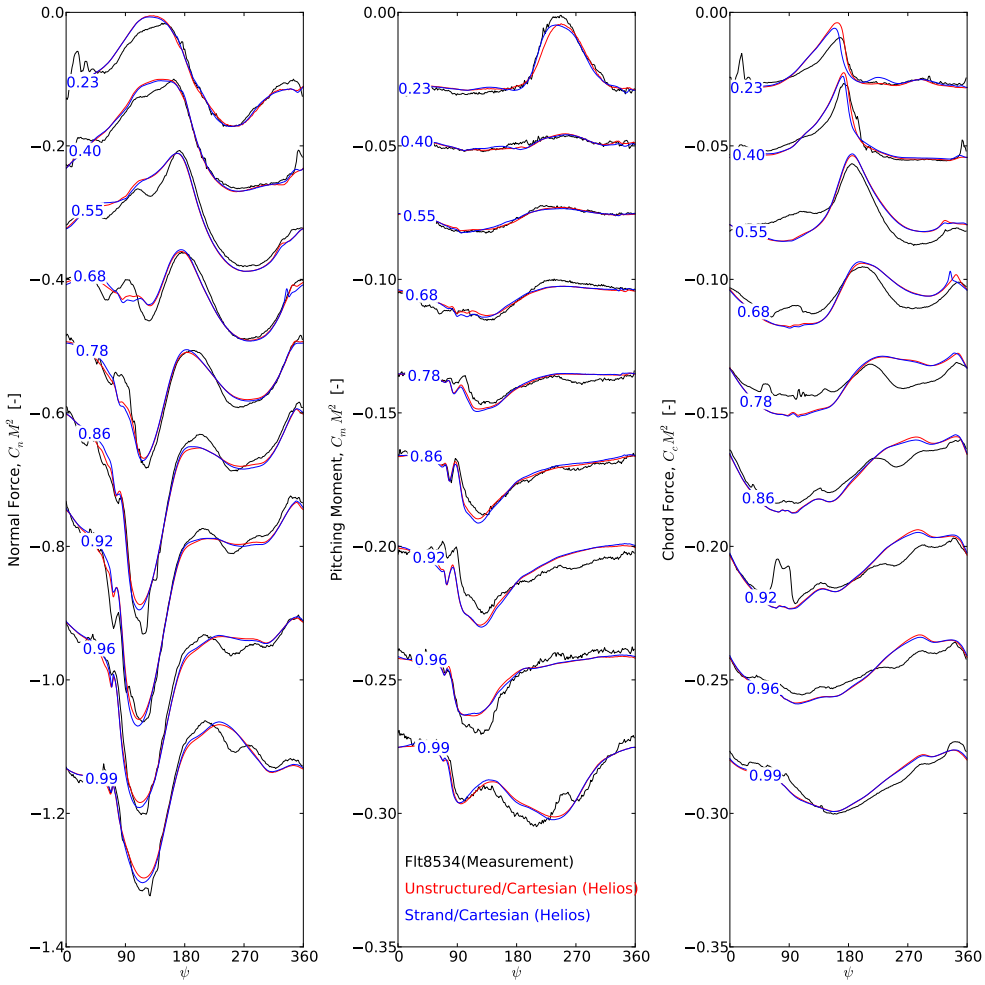


Figure 21: Comparison of computed sectional aerodynamic loading on the rotor blades with measured experimental data. Red lines are strand/Cartesian results and blue Lines are unstructured/Cartesian results. Aerodynamic loading at different sections of the blade are shown in a spread plot format. Loading is non-dimensional and in the deformed airfoil frame. Normal Force : normal to the local chord line and towards the suction side of the airfoil. Chord Force: aligned with the chord line and towards the leading edge. Pitching moment : around the local quarter chord and positive nose-up.

calculations using fully unstructured volume meshes. Several challenges and open problems still remain in this area, they are : (1) improvement of local mesh quality by inclusion of mesh quality gradient into the spring smoothing technique (2) merging of topology to avoid very small elements and further increase the prismatic level thickness (before self-intersections become necessary) and finally (3) improved heuristic for automatic determination of the number of levels and level thickness to obtain the prismatic envelope at a desired distance for a given geometry.

Acknowledgments

Material presented in this paper is a product of the CREATE-AV Element of the Computational Research and Engineering for Acquisition Tools and Environments (CREATE) Program sponsored by the U.S. Department of Defense HPC Modernization Program Office. Authors are also grateful to Dr. Robert Haimes, for his guidance and extensive discussions on various aspects of the meshing algorithms.

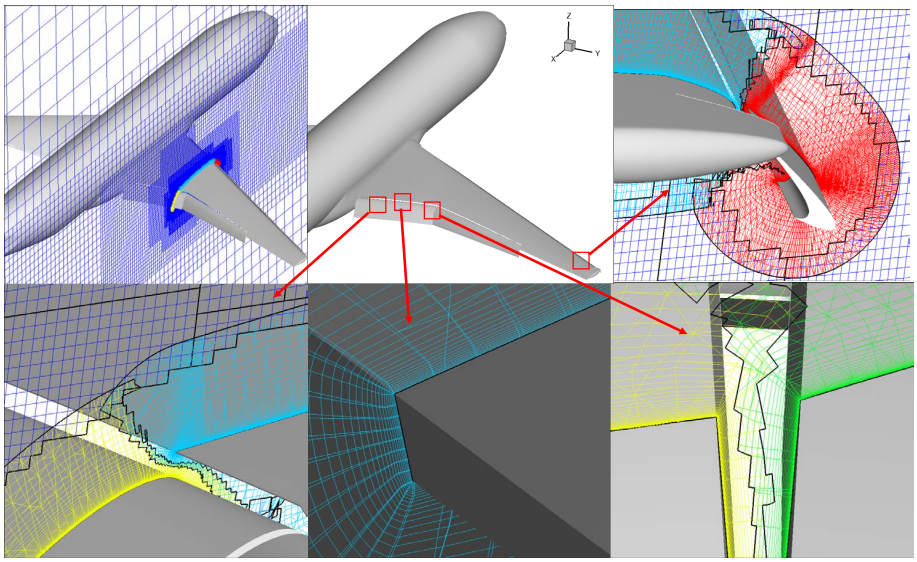


Figure 22: Mesh system used to model the High-lift Common Research Model (HLCRM)

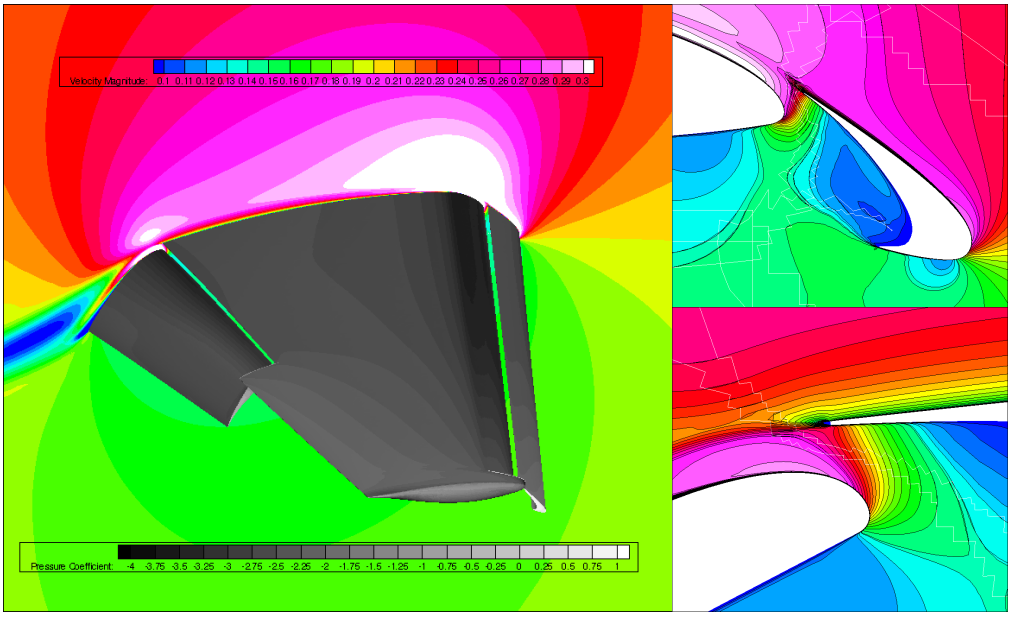


Figure 23: Flow solution obtained for the High-lift Common Research Model ($\alpha = 8^o$, $M = 0.2$, $Re = 3.26 \times 10^6$)

References

[1] Shahyar Pirzadeh. Unstructured viscous grid generation by the advancing-layers method. *AIAA journal*, 32(8):1735–1737, 1994.

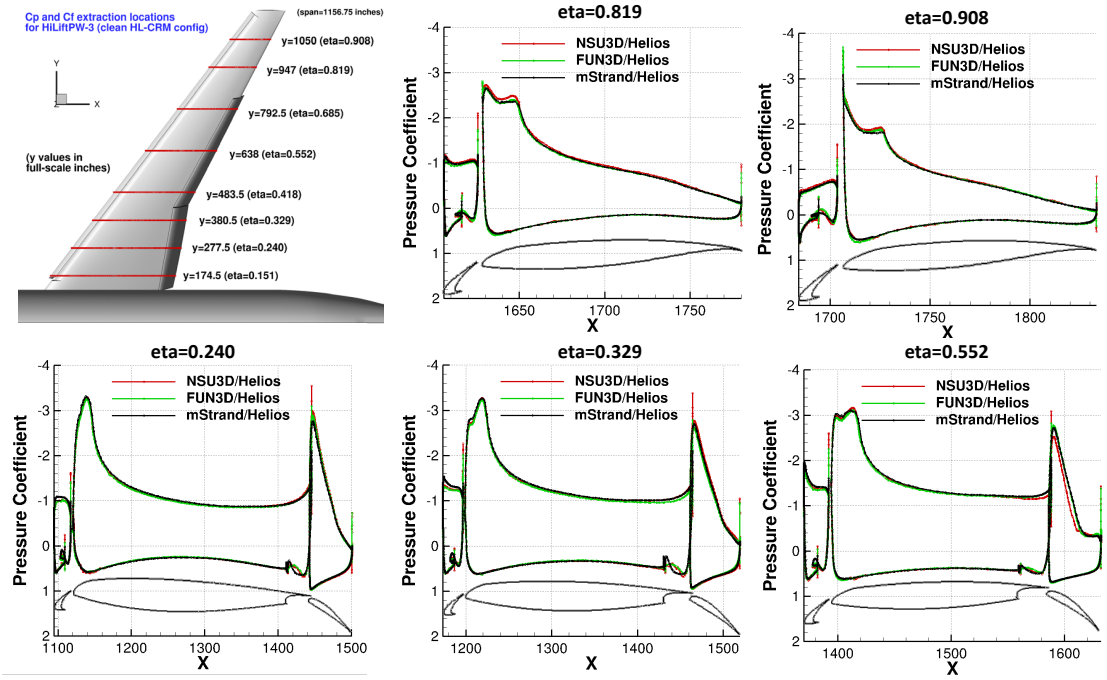


Figure 24: Sectional pressure distributions at several span stations from strand/Cartesian computations and their comparison to unstructured/Cartesian computations ($\alpha = 8^o$, $M = 0.2$, $Re = 3.26 \times 10^6$).

[2] Rainald Löhner. Progress in grid generation via the advancing front technique. *Engineering with computers*, 12(3-4):186–210, 1996.

[3] Yannis Kallinderis, Aly Khawaja, and Harlan McMorris. Hybrid prismatic/tetrahedral grid generation for viscous flows around complex geometries. *AIAA journal*, 34(2):291–298, 1996.

[4] David Marcum and J Gaither. Mixed element type unstructured grid generation for viscous flow applications, 1999.

[5] Zhi Wang, Jaime Quintanal, and Roque Corral. Accelerating advancing layer viscous mesh generation for 3D complex configurations. *Procedia Engineering*, 203:128–140, 2017.

[6] Frédéric Alauzet, Xiangrong Li, E Seegyoung Seol, and Mark S Shephard. Parallel anisotropic 3D mesh adaptation by mesh modification. *Engineering with Computers*, 21(3):247–258, 2006.

[7] Frederic Alauzet and Dave Marcum. Metric-aligned and metric-orthogonal strategies in AFLR. In *23rd AIAA Computational Fluid Dynamics Conference*, page 3108, 2017.

[8] Frédéric Alauzet and David Marcum. A closed advancing-layer method with changing topology mesh movement for viscous mesh generation. In *Proceedings of the 22nd International Meshing Roundtable*, pages 241–261. Springer, 2014.

[9] Frédéric Alauzet and Adrien Loseille. A decade of progress on anisotropic mesh adaptation for computational fluid dynamics. *Computer-Aided Design*, 72:13–39, 2016.

[10] Maximilian Tomac and David Eller. Towards automated hybrid-prismatic mesh generation. *Procedia Engineering*, (82):377–389, 2014.

[11] VA Garanzha and LN Kudryavtseva. Hyperelastic springback technique for construction of prismatic mesh layers. *Procedia Engineering*, 203:401–413, 2017.

[12] Robert Haimes. MOSS: multiple orthogonal strand system. *Engineering with Computers*, 31(3):453–463, 2015.

[13] Robert Meakin, Andrew Wissink, William Chan, and Shishir Pandya. On strand grids for complex flows. In *18th AIAA Computational Fluid Dynamics Conference*, page 3834, 2007.

[14] Adrien Loseille and Rainald Löhner. Robust boundary layer mesh generation. In *Proceedings of the 21st International Meshing Roundtable*, pages 493–511. Springer, 2013.

[15] Aaron Katz, Andrew M Wissink, Venkateswaran Sankaran, Robert L Meakin, and William M Chan. Application of strand meshes to complex aerodynamic flow fields. *Journal of Computational Physics*, 230(17):6512–6530, 2011.

[16] Aaron Katz and Dalon Work. High-order flux correction/finite difference schemes for strand grids. *Journal of computational physics*, 282:360–380, 2015.

[17] Oisin Tong, Aaron Katz, Yushi Yanagita, Alex Casey, and Robert Schaap. High-order methods for turbulent flows on three-dimensional strand grids. *Journal of Scientific Computing*, 67(1):84–102, 2016.

[18] Jayanarayanan Sitaraman, Vinod K Lakshminarayan, Beatrice Roget, and Andrew M Wissink. Progress in strand mesh generation and domain connectivity for dual-mesh CFD simulations. In *55th AIAA Aerospace Sciences Meeting*, page 0288, 2017.

[19] Vinod Lakshminarayan, Jayanarayanan Sitaraman, and Andrew Wissink. Sensitivity of rotorcraft hover predictions to mesh resolution in strand grid framework. *AIAA Journal*, pages 1–12, 2017.

[20] Vinod K Lakshminarayan, Jayanarayanan Sitaraman, Beatrice Roget, and Andrew M Wissink. Simulation of complex geometries using automatically generated strand meshes. In *2018 AIAA Aerospace Sciences Meeting*, page 0028, 2018.

[21] Jay Sitaraman and Beatrice Roget. OSCAR - an overset grid assembler for overlapping strand/cartesian mesh systems. In *11th Symposium on Overset Composite Grids and Solution Technology, Dayton, Ohio, October 15-18*, 2012.

[22] Romain Aubry and Rainald Löhner. On the most normal normal. *International Journal for Numerical Methods in Biomedical Engineering*, 24(12):1641–1652, 2008.

[23] Romain Aubry and Rainald Löhner. Generation of viscous grids at ridges and corners. *International Journal for Numerical Methods in Engineering*, 77(9):1247–1289, 2009.

[24] M Wissink, J Sitaraman, A Katz, and B Roget. Application of 3D strand mesh technology to rotorcraft hover. In *53rd AIAA Aerospace Sciences Meeting*, pages 2015–0044, 2015.

[25] Javier Bonet and Jaime Peraire. An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems. *International Journal for Numerical Methods in Engineering*, 31(1):1–17, 1991.

[26] Beatrice Roget and Jayanarayanan Sitaraman. Wall distance search algorithm using voxelized marching spheres. *Journal of Computational Physics*, 241:76–94, 2013.

[27] PG Tucker. Differential equation-based wall distance computation for DES and RANS. *Journal of computational physics*, 190(1):229–248, 2003.

[28] Derek Jung and Kamal K Gupta. Octree-based hierarchical distance maps for collision detection. *Journal of Robotic Systems*, 14(11):789–806, 1997.

[29] Andrew M Wissink, Jayanarayanan Sitaraman, Buvaneswari Jayaraman, Beatrice Roget, Vinod K Lakshminarayan, Mark A Potsdam, Rohit Jain, Andrew Bauer, and Roger Strawn. Recent advancements in the Helios rotorcraft simulation code. In *54th AIAA Aerospace Sciences Meeting*, page 0563, 2016.

[30] Robert T Biedron, Jan-Renee Carlson, Joseph M Derlaga, Peter A Gnoffo, Dana P Hammond, William T Jones, William L Kleb, Elizabeth M Lee-Rausch, Eric J Nielsen, Michael A Park, et al. FUN3D manual: 13.2. 2017.

[31] Dimitri J Mavriplis and Karthik Mani. Unstructured mesh solution techniques using the NSU3D solver. In *52nd Aerospace Sciences Meeting*, page 0081, 2014.

[32] P. Spalart and S. Allmaras. A one-equation turbulence model for aerodynamic flows. In *30th aerospace sciences meeting and exhibit*, page 439, 1992.

[33] Steven R Allmaras and Forrester T Johnson. Modifications and clarifications for the implementation of the Spalart-Allmaras turbulence model. In *Seventh international conference on computational fluid dynamics (ICCFD7)*, pages 1–11, 2012.

[34] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[35] Vinod K Lakshminarayan, Jayanarayanan Sitaraman, Beatrice Roget, and Andrew M Wissink. Development and validation of a multi-strand solver for complex aerodynamic flows. *Computers & Fluids*, 147:41–62, 2017.

[36] Andrew M Wissink, Richard D Hornung, Scott R Kohn, Steve S Smith, and Noah Elliott. Large scale parallel structured AMR calculations using the SAMRAI framework. In *Supercomputing, ACM/IEEE 2001 Conference*, pages 22–22. IEEE, 2001.

[37] David L Marcum. Advancing-front/local-reconnection (AFLR) unstructured grid generation. In *Computational Fluid Dynamics Review 1998: (In 2 Volumes)*, pages 140–157. World Scientific, 1998.

[38] John P Steinbrenner. Construction of prism and hex layers from anisotropic tetrahedra. In *22nd AIAA Computational Fluid Dynamics Conference*, page 2296, 2015.

[39] H Saberi, M Hasbun, JY Hong, H Yeo, and RA Ormiston. Overview of RCAS capabilities, validations, and rotorcraft applications. In *Proceedings of the American Helicopter Society 71st Annual Forum*, pages 5–7, 2015.

[40] Beatrice Roget, Jay Sitaraman, and Andrew M Wissink. Maneuvering rotorcraft simulations using CREATE A/V Helios. In *54th AIAA Aerospace Sciences Meeting*, page 1057, 2016.

[41] R Kufeld, Dwight L Balough, Jeffrey L Cross, and Karen F Studebaker. Flight testing the UH-60A airloads aircraft. In *ANNUAL FORUM PROCEEDINGS-AMERICAN HELICOPTER SOCIETY*, volume 5, pages 557–557. American Helicopter Society, 1994.

Appendix

Algorithm 1 Initial strand placement

X: set of surface grid node coordinates (x_i)
S: set of tuples describing facets
L: desired distance from the surface to place points
Y: (output) set of strand end node coordinates (y_i)
procedure InitStrands(X,S,L)
    $\mathcal{S} \leftarrow (X, S)$ ▷ discrete surface tessellation
    for each $s_i \in S$ **do**
        $n_i \leftarrow$ facetNormal(s_i)
    end for
    for each $x_i \in X$ **do**
        $V \leftarrow \{n_j\}$ ▷ set of normals of all facets that have x_i as a corner
        $n_i \leftarrow$ `MNN`(x_i, V) ▷ best visibility direction (`MNN` algorithm [22])
        $m \leftarrow x_i + Ln_i$
        $d \leftarrow$ `MinDist` $(m, \mathcal{S})$ ▷ minimum distance to the surface (X, S)
        if $d < L$ **then** ▷ x_i is in a concave region w.r.t to I_L
            $c \leftarrow$ `CLOVIS`$(x_i, n_i, \mathcal{S}, L)$ ▷ See section 2.6.5
            if c outside visibility region **then**
                $y_i \leftarrow m$
            else
                $y_i \leftarrow c$
            end if
        else
            $y_i \leftarrow m$
        end if
    end for
end procedure

Algorithm 2 Initial Guess for `CLOVIS`

A: surface node location
$\vec{n}$: initial normal (best visibility direction if defined, average normal otherwise)
$\mathcal{S}$: set of tuples describing facets
L: desired distance from the surface to place points
B: (output) end point of strand vector on minimum distance iso-surface, I_L
procedure InitCLOVIS(A,$\vec{n}$,$\mathcal{S}$,L) ▷ Find a good initial guess that lies on the I_L
    $i \leftarrow 0$
    $B \leftarrow A + L\vec{n}$
    $[d, C] \leftarrow$ `MinDist` $(B, \mathcal{S})$ ▷ Distance and location of closest point on the surface $\mathcal{S}$
    while $|d - L| > \epsilon$ **do**
        $i \leftarrow i + 1$
        $[d, C] \leftarrow$ `MinDist` $(C + \frac{L}{d}\overrightarrow{CB}, \mathcal{S})$
    end while
    return B
end procedure

Algorithm 3 `CLOVIS`

A: surface node location
$\vec{n}$: Initial normal, usually `MNN`
$\mathcal{S}$: surface tessellation
L : Iso-surface distance desired
B : Closest point on the iso-surface (output)
procedure `CLOVIS`(A,$\vec{n}$,$\mathcal{S}$,L) ▷ Find the closest vertex on the minimum distance iso-surface
    $i \leftarrow 1$
    $B_1 \leftarrow$ `InitCLOVIS`$(A, \vec{n}, \mathcal{S}, L)$
    $[d_i, C_i] \leftarrow$ `MinDist` $(B_i, \mathcal{S})$
    $\alpha \leftarrow 0.02 \times AC_i$ ▷ Initial guess for line search parameter
    repeat
        $\vec{s_i} \leftarrow \left(\overrightarrow{B_iC_i} \times \overrightarrow{B_iA}\right) \times \overrightarrow{B_iC_i}$
        repeat ▷ find α that will ensure $\|AB_{i+1}\| < \|AB_i\|$
            $\alpha \leftarrow 0.5 \times \alpha$
            $B_t \leftarrow B_i + \alpha \frac{\vec{s_i}}{\|\vec{s_i}\|}$
            $B_t \leftarrow A + \frac{AB_t}{\|AB_t\|}\|AB_i\|$ ▷ Move B_t such that $\|AB_t\| = \|AB_i\|$
            $[d_t, c_t] \leftarrow$ `MinDist` $(B_t, \mathcal{S})$
        until $d_t \leq L$ and $\alpha > \epsilon$ ▷ $d_t \leq L \implies \|AB_{i+1}\| > \|AB_i\|$
        if $\alpha \leq \epsilon$ **then** ▷ No α can reduce distance, minimum distance point is found
            $B \leftarrow B_i$
            return B
        else
            $B_{i+1} \leftarrow$ `flushToIso`(A, B_t, L) ▷ Shorten AB_t such that B_{i+1} lies on L
            $\alpha \leftarrow 4.0 \times \alpha$ ▷ Increase line search parameter for the next iterate
            $i \leftarrow i + 1$
            $[d_i, C_i] \leftarrow$ `MinDist` $(B_i, \mathcal{S})$
        end if
    until $\|B_{i+1} - B_i\| > \epsilon$
    $B \leftarrow B_i$
    return B
end procedure