

MIT Open Access Articles

MOSS: multiple orthogonal strand system

The MIT Faculty has made this article openly available. **Please share** how this access benefits you. Your story matters.

Citation: Haimes, Robert. "MOSS: Multiple Orthogonal Strand System." *Engineering with Computers* 31.3 (2015): 453–463.

As Published: <http://dx.doi.org/10.1007/s00366-014-0375-9>

Publisher: Springer London

Persistent URL: <http://hdl.handle.net/1721.1/103796>

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Terms of Use: Article is made available in accordance with the publisher's policy and may be subject to US copyright law. Please refer to the publisher's site for terms of use.



MOSS: Multiple Orthogonal Strand System

Robert Haimes

Received: date / Accepted: date

Abstract: This paper describes an overset approach that is comprised of *virtual boundary layer*-like near-body grid coupled with an off-body Adaptive Mesh Refinement (AMR) far-field mesh for viscous fluids simulations. Unlike most *a priori* grid generation systems for the Reynolds-Averaged Navier-Stokes equations, the *strand* meshing paradigm is automatic, fast and requires little memory in order to provide boundary-layer coverage. In addition, the stacks of elements implied by the strands can be used to the simulation's advantage, where they naturally provide a line direction for semi-implicit solving.

Keywords: Boundary layer · Strand mesh · Automatic · RANS

1 Introduction

The use of Computational Fluids Dynamics (CFD) for complex 3D geometries has become commonplace in engineering analysis. This is done at various levels of modeling fidelity from panel methods, the use of the Euler Equations, Reynolds-Averaged Navier-Stokes equations (RANS), Large Eddy Simulations (LES) and even Direct Numerical Simulation (DNS) of the Navier-Stokes equations (without any turbulence modeling due to the ability to resolve the whole range of spatial and temporal scales). Each of these techniques has different meshing requirements in resolution and density and, in general, requires more and more resources (memory and CPU) as you climb the fidelity *ladder*.

The DoD program CREATE has as a central goal to put HPC simulation tools in the hands of acquisition engineers who may have domain knowledge but are not experts in mesh generation. This makes the use of automated tools that generate appropriate meshes important. To satisfy its mission CREATE intends to use a number of RANS solvers for time accurate simulations in design settings. This is an engineering compromise, in that the simulation times

are (close to) tractable and RANS can generate and convect many of the important fluid structures that are required to answer design questions. For example, in rotorcraft acoustics it is important to properly generate a boundary layer on the rotors that then naturally rolls up into vortical structures at the tip. It is important that these vortices don't prematurely dissipate. The interaction of these features with other objects (including the helicopter body) is the source for the typical acoustic signature.

Effective use of RANS solvers is a challenge. This is not because the solvers themselves are fragile but getting to the point of running the simulation can be the impediment. The requirement of a grid that is commensurate with the flow regime and the embedded geometry is the problem. A grid can be generated automatically for the solution of the Euler Equations with the use of Open Source software. The elements can be isotropic, because the Euler equations do not emit multi-scale body-based structures, but this is not true for RANS simulations. Boundary Layers are small features and tend to have strong variations close to walls (a single direction). This means that they are most effectively handled by anisotropic meshes. Conceptually, it makes sense to take structured collections of hexahedral elements, which sit on the geometry, to generate the grid system(s). This allows for the anisotropy to be naturally handled by the spacing of the grid planes in the structured blocks. These blocks can be placed side-by-side (abutting) or overlapping (as used by overset solvers) in 3 space. The grid generation task for either type of grid topology is far from automatic. It can take weeks (and maybe even months) of manual labor to successfully grid-up a single complex geometry. Clearly, this is a problem for simple parametric studies and it is out of the question to have (this much of) a human in the loop for design optimization.

An early interesting attempt at a different grid topology for solving the RANS equations can be seen in [1]. This work started from a triangulation of the body of interest and, in a sense, inflated this tessellation outward (maintaining the same surface topology). The number of inflation steps specified the number of prism layers found in the mesh. The outer exposure of the prism layers could be meshed in an isotropic manner with tetrahedra. This is an important advance in that the entire meshing process could be automated after the surface triangulation is produced.

Another significant automatic two-mesh approach is discussed in [2]. Here the off-body meshing system is AMR based. This idea is further refined into the concept of a *strand* mesh by [3] and [4], which is basically a recasting of the near-body mesh of [1] and the off-body mesh of [2].

Strands meshes are not prismatic meshes, though prism elements can be constructed from the strand mesh. The organization of a strand mesh is a response to analyzing the memory footprint of any grid used for RANS. A vast majority of the storage requirements for the mesh reside in the boundary-layer. This can be as high as 90%. Instead of requiring the storage for every 3D vertex in the mesh and then the indices into that storage to form the prisms, the first implementation of a strand mesh simply specified a single straight line for each surface vertex. The result is that there really is no near-body mesh (in

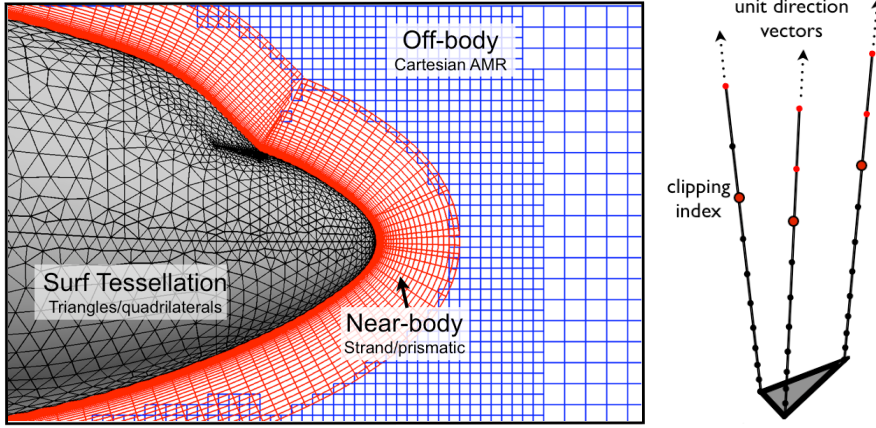


Fig. 1 Near-body / Off-body overset grid system (left) and strand definition (right). A single stack of prisms is depicted. From [4] and used with permission.

the traditional manner). The mesh is fully defined at the surface tessellation. Elements are internally constructed by the solver as needed. Strand meshing is a member of an emerging suite of *Virtual Boundary Layer Grid* systems that simply use a surface discretization to infer a volumetric mesh. Other members include [5] and [6] (which solves the 3D integral boundary layer equations).

Strands have a series of global settings which include: the strand length, the number of strand positions (the knots as seen on the right of Figure 1) and a vector of relative strand positions (numbers greater than 0 and finishing at 1.0 – in increasing order).

Individual strands (as can be seen in Figure 1) simply consist of a direction vector and a clipping index. This is a tiny requirement!

1.1 Issues with current strand implementation

Though the strand memory requirements are compelling, the implementation as discussed in [3] and [4] does have some problems:

- Spatial coverage. At corners, noses and Trailing Edges the total number of strands available to fill space is limited by the initial surface tessellation. To attempt to do a better job of filling space and maintaining consistent element sizes at the extent of the strands, extensive smoothing was applied throughout the strand mesh. This would significantly pivot the strands in the region of Trailing Edges. It is not clear what effect this would have in forming wakes and vortices (but is easy to surmise that it is not good).
- Orthogonality. After smoothing, the strands would be far from normal to the surface at places like Trailing Edges. This makes the accuracy of algebraic turbulence models suspect, but even worse is the effect on resolution.

The strand length should be specified to be the maximum size expected of the boundary layer found in the simulation (this is a global quantity). This ensures that the boundary layer is properly covered and therefore properly formed during simulation. If the strands are significantly pivoted, then much longer strands are required to cover the same distance off the wall. This is a problem in regions where the boundary layer is small or just forming. Larger strands have poor coverage where the entire length is not required.

- Premature cutoff. A single straight line emanating from the intersection of a sharp Trailing Edge and the Fuselage has no good direction that can be used to produce valid prism stacks for all of the surface triangles (see Figure 2). The right side of the Figure displays a blow-up of the flap’s Trailing Edge and the Fuselage that has the juncture low on the body (that gives a strong downward normal direction). Any strand direction emanating out of that juncture that is not parallel to the Trailing Edge itself will produce invalid stacks either above (if the strand points below the TE horizon) or below (if the strand points slightly above the TE horizon). The smoothing will force this strand downward. But even at a neutral position the stacks on both sides of the flap will be significantly skewed. In any case, inverted stacks cause the strand to have a low clipping index, which forces the off-body mesh to telescope deeply into the near-body mesh so that all volume can be covered. This is undesirable.

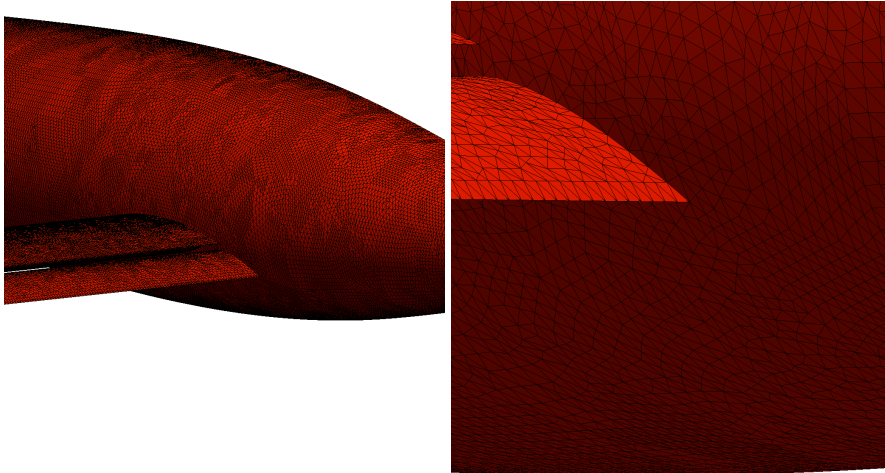


Fig. 2 Trailing Edge/Fuselage junction for multi-element wing (left) and detailed tessellation at the flap/body interface (right).

2 Design Goals

It is a luxury to be given a clean slate for a software project. To ensure a success, it is important to understand and explicitly state the desired functionality (i.e., software requirements). Automation is of paramount importance in the recasting of the strand mesh generator. The other specifications are to maintain the best parts of the current implementation and to obviously mitigate its issues. Specifically this includes:

- Memory footprint. Maintain the minimal memory requirements that roughly scales with the surface discretization of the body of interest.
- Spatial coverage. Provide a meshing scheme that can better fill volume in convex situations. The technique described in this paper *fans* multiple strands from certain surface vertices so that all volume can be consistently covered. This is inspired by tetrahedral meshes generated in [7] and the hybrid meshes of [8].

Also, due to the overset nature of the meshing system there is a requirement based on the interpolation scheme performed between the near and off-body meshes. There is both less interpolation error and better conservation properties if the spacing seen in both meshes is consistent in the overlap regions. Therefore having isotropic spacing is desirable for the strand stacks at their extent (because the AMR mesh is naturally isotropic).

- Maintain surface normal / strand coherence. Simply stated: don't pivot (smooth) the strands where no smoothing is needed. This leaves the orthogonal nature of the strand direction alone unless in regions where there is collision of stacks.
- Keep the off-body mesh away from the surface. This is synonymous with maximizing the strand cutoff index. It ensures that the correct resolution is employed in order to generate and maintain the boundary layer.

This work introduces the concept of a *lifted surface*, which is the tessellation of the collected strand element stacks seen at the full strand length. This concept is used because the tessellation at the surface is no longer simply *inflated*, with the result being that the grid topology at the surface is not the same as seen from above. Said another way, there are stacks of elements produced that appear degenerate at the surface. The type of degeneracy depends on how the stacks are constructed.

3 Implementation

The description of the **MOSS** (Multiple Orthogonal Strand System) implementation is found below. It currently starts from creation in (or import into) a geometry kernel whose result is a *solid model*. It requires a watertight tessellation of the *solid* and an indication if the strands are to emanate outward (external) or into the solid (for internal flows). The rest of the steps follow from

the information associated with the surface tessellation (i.e., what geometric entity or entities own the surface vertex) in a completely automated manner.

Besides the *solid model*, the only other inputs into this meshing system are:

- The strand length (in the model’s units).
- The number of strand positions (the number of knots as seen on the right of Figure 1).
- The relative strand positions (a vector of knot distances – numbers greater than 0 and finishing at 1.0 representing the full strand length).
- The number of smoothing iterations (see Section 3.10).

3.1 Initial import from Geometry Kernel

MOSS has been implemented on top of EGADS [9] (which is built upon OpenCASCADE [10]), CAPRI [11] and Capstone (from CREATE-MG) but could be ported to any *solid modeling* geometry kernel that has the following capabilities:

- Supports a Boundary Representation (BRep) that refers to the hierarchy and connectivity of topological objects (Faces, Edges and Nodes). These topological entities have underlying geometric objects that are surfaces, curves and 3D positions in space, respectively.
- Supply a manifold check function that ensures that the geometry to be meshed is, in fact, a *solid*.
- Provides evaluation functions (i.e. given a Face/surface and $[u, v]$ coordinates return the 3D position $[x, y, z]$).
- Computation of surface normals given a Face and $[u, v]$.
- The ability to support the construction of (or provide) a watertight tessellation of the *solid* (see Section 3.2).

The starting point of a *solid model* is not a firm requirement (and will be relaxed in the future), but is crucial for automation. It allows for knowing how to treat the bounds of Faces and Edges, which will become obvious as the discussion of the implementation progresses.

3.2 Tessellation consistent with the Geometry Kernel

Since the strands emanate from a tessellation of the body of interest, it is important that whatever scheme is used to provide that discrete view of the geometry either comes from, or can be reassociated with, the BRep as held by the geometry kernel. The tessellation must have the following characteristics:

- Be watertight and manifold.
- Be curvature driven.
- Can contain triangles and/or quadrilaterals.

- Edge vertices must be able to be traced through the tessellation to recover a discretized representation of the curve. The Edge discretization must begin and end at vertices that are the location for Nodes.
- All vertices are marked with the owning geometry (Faces, Edges and Nodes).
- A list of Faces and $[u, v]$ coordinates for each Face is required at any vertex in the tessellation. The list has a single entity for a vertex interior to a Face, usually 2 for an Edge and there are 2 or more entries in the list for Nodes.

The list of Faces is used to connect the vertices back to the geometry and to directly compute the surface normals that are required at each vertex. These normals will represent the starting directions for the strands. Those vertices associated with Edges and Nodes will initially have multiple normal specifications (one for each Face touching the entity).

3.3 Classification of vertices associated with Edges/Nodes

The classification of vertices in the tessellation is used to determine how to treat specific situations found in the geometry. It indicates whether the normals can be merged into a single strand or if adding additional strands at an Edge/Node vertex is required.

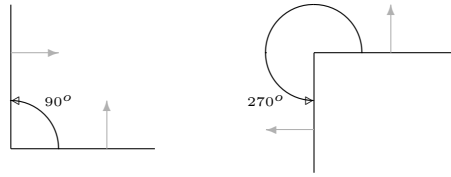


Fig. 3 *Winding Angle* – In the plane is generated by the cross-product of the Face Normals (gray vectors).

The classification is driven by the *winding angle* found between pairs of Faces and the normals associated with each Face at these tessellation vertices as seen in Figure 3. The *winding angle* is simply the angle traversed starting at one surface, pivoting at the vertex and ending at the other (in the plane generated by the cross product of the 2 normals). If the angle is less than 180° then the pair is *concave*, if greater then it is considered *convex*.

- Convex Edge Vertex. This is where the pair of Faces is locally convex, which may require adding additional strands by fanning (see Sections 3.5 and 3.6) unless the vertex can be also classified as *Same Normal*.
- Concave Edge Vertex. If the Edge vertex displays a Face pair as concave then it is marked for merging, which is applied in Section 3.4.
- Convex Node. This vertex has all combinations of pairs of Faces (from the Face list) indicating convex. It is treated in Section 3.7.

- Concave Node. This vertex has at least a single Face pair flagged as concave. The vertex is treaded in the same manner as a *Concave Edge* (see Section 3.4) unless it can also be classified as an *Opposite-Normals Node*.
- Opposite-Normals Node (ONN). This classification is for situations like that seen in Figure 2 where there is a Face pair that displays a *dot* of the normals that is less than -0.95 . This occurs at sharp Trailing-Edge/Fuselage intersections and must be handled in a special manner as described in Section 3.8.
- Same Normal. When the difference between the normals for a Face pair is less than 3° , the vertex is marked as having the same normals. This occurs when there is a smooth transition between Faces or the vertex is from a periodic-like seam separating a single closed surface.

3.4 Merging of normals for a *Concave Edge/Node* strand

Marking a vertex as being either a *Concave Edge* or a *Concave Node* indicates that the multiple normals will be coalesced to a single strand. The direction of the strand is the average of the directions of all of the normals in the Face list. Any set of normals that is marked as *Same Normal* will only be included once in the sum. The new strand direction is renormalized.

This is usually just the starting strand direction. Being *concave* indicates that there will probably be overlapping element stacks and this strand will be a candidate for smoothing (see Section 3.10).

At this point we can examine the *lifted surface* because all of Edges and Nodes that will not have expanded treatments have been handled. A contrived *solid* geometry can be seen on left of Figure 4 with its tessellation displayed. At this point the *lifted surface* is the tessellation supported by the strands at the strand length, which can be seen as the yellow surfaces on the right of Figure 4. Note that the locations marked as *convex* are open and will be filled in next.

3.5 Specifying Edge strand fanning numbers

In order to compute a spacing for vertices marked as *Convex Edges*, the tessellation is examined and the distance perpendicular to the Edge itself (on the Face) is computed from the surface triangles/quadrilaterals. The spacing on both Faces is averaged and by knowing the *winding angle* and the strand length, it is simple to compute the number of subdivisions of the *winding angle* that are needed to meet this averaged spacing requirement. This integer is stored away with the vertex.

After all of the Edge vertices are handled, each complete discretized Edge is examined by traversing from start to end Node. Any abrupt changes in subdivision numbering is smoothed. Note that this will taper the subdivision numbers of any *Convex Edge* vertex adjacent to a *Concave Node* that obviously

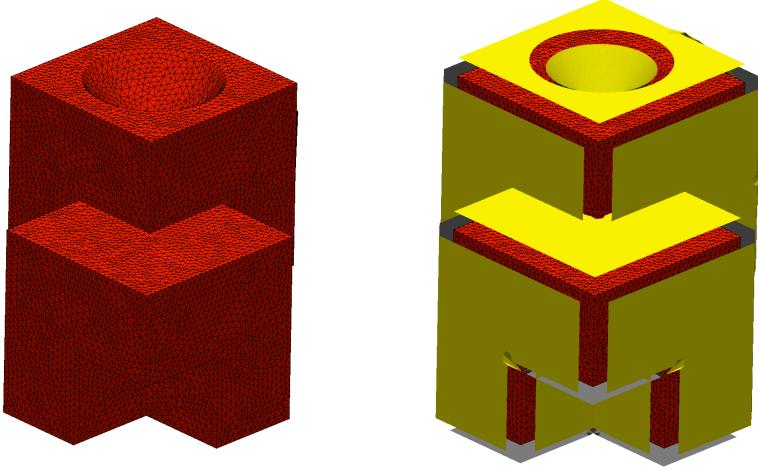


Fig. 4 Original BRep displayed with watertight triangulation (in red) and *lifted surface* with separation at *convex* Edges and Nodes (yellow).

needs to collapse to the single strand. Any *Convex Edge* vertex touching a *Convex Node* or a *Opposite-Normals Node* is not adjusted. The closure of a *Concave Edge* vertex adjacent to an *ONN* is reopened in preparation for the treatment seen in Section 3.8.

Nodes with only two coincident Edges also merit special attention. If the Node is additionally marked as *Same Normal* then the fanning number at the ends of the Edges are set to the average of the values for both.

3.6 Creation of Edge fanned strands

For each vertex marked as a *Convex Edge*, new strands are created that fan from one of the Face normals to the other. These are the locations that will be connected by sets of triangles constructed for each discretized Edge segment. This is done in a manner consistent with what is seen in [7], which fills the segment with a polytriangle strip. If the fans on both side of the strip have the same number of subdivisions, quadrilaterals could be used but are not. For simplicity, all added elements on the *lifted surface* are triangles.

The result of the construction of the fanned Edges from Figure 4 can be seen on the left of Figure 5. The lifted tessellation and constructed triangles are highlighted. If quadrilaterals were used, the element at the base would be a prism (the collapsing of the base quad into a line – the Edge segment). But with triangles, the element stacks produced by this procedure terminate with a *wedge* (a 3D cell with 5 vertices, built with 2 quadrilateral and 2 triangular sides). This can be thought of as the degenerate quadrilateral-based prism cut in half along the diagonal of the top quad side. The quadrilateral sides of the wedge laterally match up with the quads on the neighboring stacks. The triangle at the top starts the prismatic stack and the base itself is the line corresponding to the Edge segment.

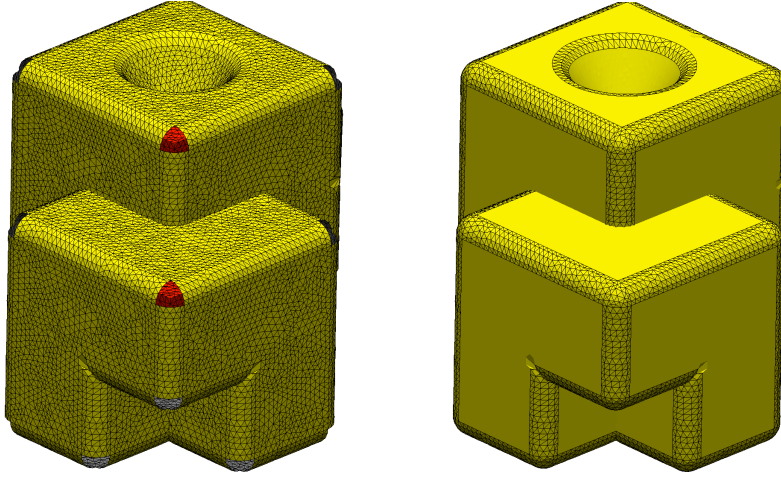


Fig. 5 Fans completed on Edges (left) and full unsmoothed lifted surface (right).

3.7 Filling in of *Convex Nodes*

For the situation where two Edges hit a Node marked as *Same Normal* then the fans match up by construction. This situation is then simply closed where all other *Convex Nodes* are open.

Voids in the *lifted surface* can be seen on the left of Figure 5 where *Convex Edges* come together at a *Convex Node*. The number of sides of this convex open polygonal region is determined by the number of Edges meeting at the Node. A new spacing requirement is set for each void as the average of the exposed segment distances. The void is closed by the following procedure:

1. Creation of center strand. This is done by averaging the direction of all of the strands that outline the convex region to be filled.
2. Close up the exposed Edge segments by creating triangles that have 2 positions on the exposed Edge opening and connect to the new center strand.
3. Insert a new strand where the spacing is too large by splitting the interior triangle side, which creates 4 triangles from the original 2.
4. Use a MINMAX angle criteria to drive swapping of interior triangles in order to maintain a good quality tessellation.
5. Iterate on 3 and 4 above until the spacing requirement is satisfied.

The resultant *lifted surface* after filling the *Convex Node* vertices can be seen on the right of Figure 5 (where only the constructed triangles are outlined). Since this geometry has no *ONN* vertices, the entire object is now closed by the complete tessellation of the original object and the constructed triangles. All element stacks from the original tessellation are a simple reflection of the surface and could produce series of prisms for triangles and stacks of hexahedra for quadrilaterals. All constructed triangles (at the *lifted surface*) do not exist in the original tessellation and collapse to a single surface vertex

(associated with a BRep Node) and a line segment for a BRep Edge. The stack is primarily prismatic except at the base (the surface vertex) where it degenerates to a tetrahedron for the Node and a wedge for an Edge segment.

3.8 Filling in of *ONN* – Terminating a Fanned Trailing Edge

The goal here is to provide a technique that allows for the construction of a valid mesh under the condition of no clear strand visibility for all of the stacks touching a *Opposite-Normals Node* (if simply merged).

The operation is like the procedure found in Section 3.7 except that, because the situation is actually closed (like for a *Concave Node*), the construction is done producing triangles (in the *lifted surface*) in the opposite orientation. The first step is to generate a strand from the node that is aligned with the Trailing Edge (the Face pair creating the *ONN*). This is used as part of the reopening of a *Concave Edge* vertex adjacent to this Node as described in Section 3.5. And in the case of Figure 2 it is the first Trailing Edge segment that touches the body. This new strand is marked as *frozen* for the smoothing operation described in Section 3.10. Then these steps are used:

1. Creation of center strand. This is done by averaging the direction of all of the strands that outline the region to be filled. This strand is also marked as *frozen*.
2. Close up the exposed Edge segments by creating triangles that have 2 positions on the exposed Edge opening and connect to the new center strand.

It should be noted that the prismatic stacks inferred by these triangles are also of the opposite orientation compared to the rest of the elements (the *lifted surface* orientation is opposite). This construction is only used to close the lifted surface tessellation and these stacks should not be passed on to the solver. The two created strands and the constructed triangles are all marked as *inverted*. See the discussion in Section 3.11.

The left portion of Figure 8 depicts these stacks as they close off the Trailing Edge fans. It should be noted that some of the volume represented by these stacks is initially inside the Fuselage (which has been removed from the figure for clarity).

3.9 Verify the closed *lifted surface*

The following criteria are used in order to check the validity of the *lifted surface*:

- Do all lifted elements have neighbors? If any elements in the tessellation that makes up the *lifted surface* does not have a neighbor, then the discretization is not manifold and something has gone wrong in the construction.

- Is the first volume element in all stacks valid? If this is not the case there is something wrong with the orientation of one or more strands and the smoothing phase may not be able to fix the problem.

The following procedure is used to test for stack validity, drive the smoothing (Section 3.10) and set the *clipping index* (Section 3.11), which starts at the first strand index off the surface:

1. Generate the triangle/quadrilateral at the specified strand height (or index)
2. Produce the triangle/quadrilateral normal
3. Dot the normal with each strand direction supporting the triangle / quadrilateral
4. Stop if any is less than or equal to zero
5. Increment the strand index
6. Goto 1 until the full strand length is reached

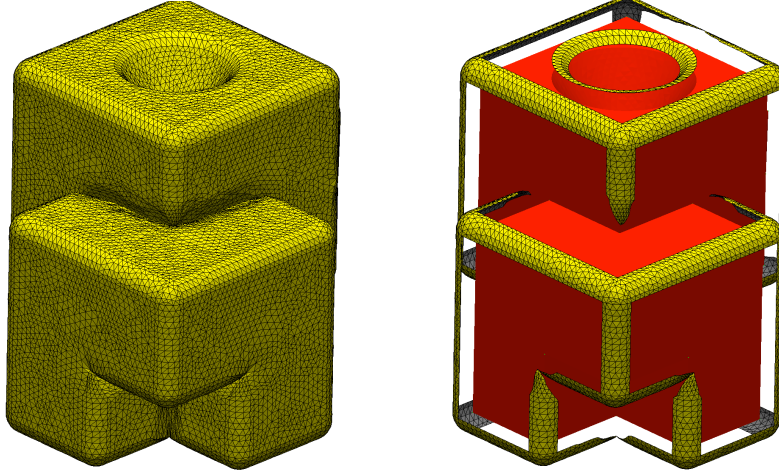


Fig. 6 Smoothed *lifted surface* (left) and examination of smoothed fans (right).

3.10 Smoothing (strand pivoting)

The purpose of the smoothing is to adjust strands that locally collide with the goal that all strands provide valid stacks up to the specified strand length. A seductively simple Laplacian smoother is used in a manner similar to that described in [1]. The procedure is as follows:

- Mark all strands where the stack validity check indicates that the directions are converging. This is a tighter criteria than inversion where the minimum dot product of the strands supporting the stack and the lifted surface facet normal is at 24° or less. Initially there should be no candidates from *Concave Edges/Nodes* due to their cylindrical and radial-like construction.

- Flood the *lifted surface* neighbors up to a specified depth away from marked strands (currently this is set to 4 neighbors). This allows for pivoting into a larger region.
- Update the touched strands by performing the Laplacian smoother (averaging neighboring strand directions and renormalizing) unless the strand is marked as *frozen*.

Generally the strand smoothing is done in 2 phases, each is terminated by the convergence of strand directions or by reaching a maximum number of iterations. Each phase is performed a user specified number of times (5 by default):

1. Edge/Node phase. This only adjusts strands emanating from either Edge or Node vertices.
2. Interior phase. Only smooths strands that can be found interior to Faces.

On the left of Figure 6 the completed smoothed *lifted surface* can be seen. Most of the tessellation is left undisturbed; only where there was the collision of stacks have the strands been adjusted. The picture on the right of Figure 6 displays the constructed tessellation from *Convex Edges* and *Convex Nodes*. Note that the smoothing occurred primarily where the fans have tapered-off towards *Concave Nodes*.

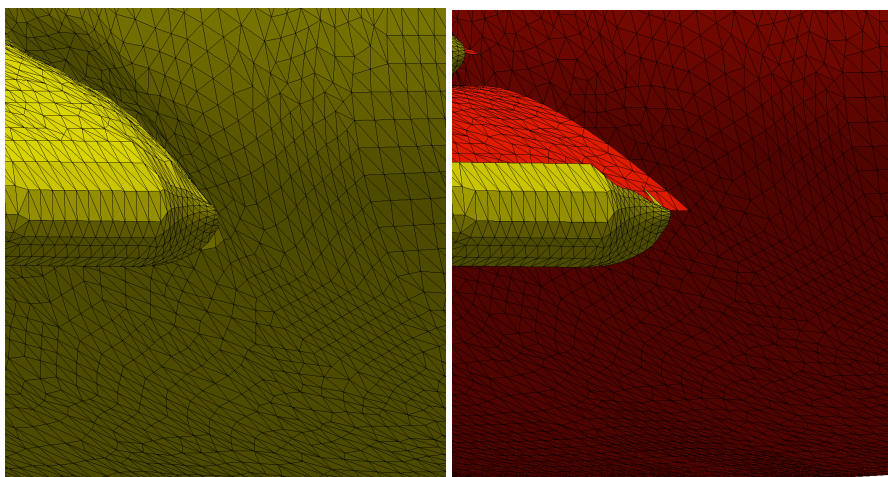


Fig. 7 Smoothed lifted-surface for flap Trailing Edge/Fuselage junction (as seen in Figure 2) for multi-element wing (left) and a display of only the fans from the complete lifted-surface (right).

The left picture of Figure 7 displays a blow up of the completed smoothed *lifted surface* at the flap/body junction as seen initially in Figure 2. The right part of Figure 7 shows the smoothed fanned *lifted surface* covering the Trailing Edge. It can be noted that the smoothed *ONN* treatment shows a similar structure to the tapered fans that can be seen in Figure 6, which is partially do to

the strands pivoting away from the juncture. In addition, the fanned triangles have moved from intersecting the Fuselage to providing a gap (which is filled by neighboring stacks) of approximately the strand length. This movement was along the direction of the Trailing Edge even though the predominate strand direction in this region has a strong downward component (due to the Fuselage).

The control of the strand smoothing is accomplished by the construction of inverted stacks as described in Section 3.8. The right hand side of Figure 8 shows the apex of the conical region defined by these stacks that start at the *ONN* and extend to the base of fanned trail-edge treatment (originally at the Fuselage). This ensures that the movement is limited by the strand length and also allows for the *frozen* strands to guide that movement. The limited movement is accomplished because all inverted stacks include at least one of the strands that are static and aligned with the appropriate direction.

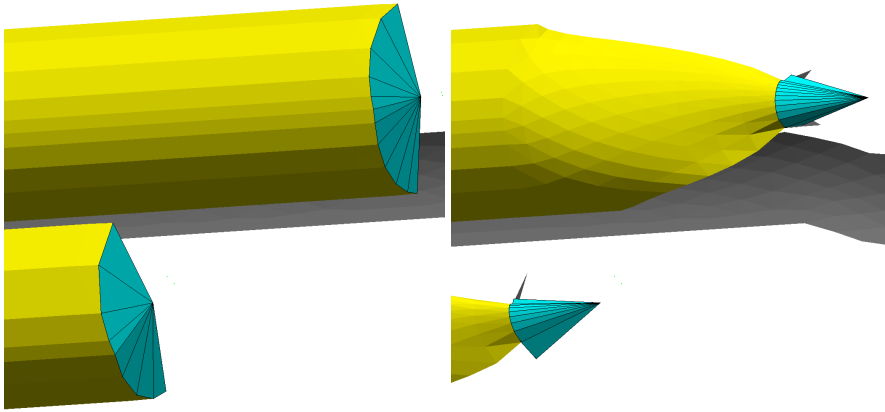


Fig. 8 *Opposite-Normals Node* treatment for both wing and flap Trailing Edge/Fuselage junction before smoothing (left) and after smoothing (right). Cyan indicates the sides of the inverted stacks. The Fuselage and *lifted surface* (except for fans) has been turned off so not to obscure the view.

3.11 Finish by setting the clipping index

The strands and inverted stacks constructed to deal with the *Opposing Normals Node* treatment can be removed. The region these stacks occupy is completely overlapped by other stacks and they should not be exposed to the solver (their internal orientation generates negative volumes). This leaves a *tear* in the *lifted surface* but in a covered concave region.

The algorithm described in Section 3.9 is used to set the clipping index associated with each strand. The index used is the minimum found by all *lifted surface* elements touching the strand.

Non-local interference (when a parts of a body come closer to itself than the strand length) or body/body interference can now be computed and the strand clipping index adjusted down where appropriate. This is not done as part of **MOSS** but is performed by PICASSO [12] for the CREATE-AV suite of solvers.

4 Discussion & Status

This paper presents a straight-forward approach to strand grid generation. Coupled with an off-body AMR mesh, this multiple mesh scheme removes the grid generation expert and manual procedures from the process of getting results from CFD RANS solvers. The fairly simple meshing technique outlined in this paper meets the lofty goals articulated in Section 2. Specifically:

- Fast and automatic. This is due to the fact that the formation of the *lifted surface* requires simple construction and bookkeeping. Most of the CPU time consumed in the overall process (after the initial tessellation) is in the strand smoothing. And, this is a trivial iterative process of applying a local Laplacian operation.
- Strand pivoting (smoothing). This is where there may be a requirement for user intervention. The pivoting, infrequently, does not produce the desired result. There is nothing in the smoothing operator that will limit or bound the strand direction. The end result is that the local Laplacian smoother can diverge away from maximizing the strand cutoff index. When this happens the user is left with the only option of reducing the number of outer smoothing iterations, which may have the negative effect of possibly leaving other regions unfinished.

This can clearly be improved upon at the expense of CPU time. Limits could be placed on the pivoting of the strands, which would allow maintaining the local nature of the operator. Global operators could be applied such as an optimization that explicitly maximizes the strand cutoff index or equalizes the areas (or spacings) at the top of the element stacks. The latter is consistent with the desire to be isotropic for the mesh-to-mesh interpolation.

- Solver Requirements. What is needed is a solver that can deal with the fact that the elements within **MOSS**' stacks are not explicitly defined (if you wish to take advantage of the small memory footprint). At a minimum, the solver needs to be able to effectively handle prisms and more importantly occasional tetrahedra and wedges that reflect the collapsing of one or more vertices on the surface. Also, if the initial tessellation contains quadrilaterals, then the solver will see stacks of hexahedra.

Initial testing of traditional CFD solvers based on finite-volume discretizations has not shown promising results [13]. This class of solver is sensitive to abrupt changes in spacings, triangle/tetrahedron/wedge elements types and high valence numbers in the mesh. **MOSS** grids display all of these

characteristics in one region or another. More recent findings using finite-element CFD solvers display none of these problems in 2D [14] with no reason to believe that these results would not be seen in 3D as well. This has prompted an unusual set of circumstances where the ability to mesh has driven choices in solver technology (instead of visa-versa). Work is currently underway in the CREATE-AV program to write a 3D finite-element CFD solver that can natively deal with the near-body mesh that **MOSS** can produce.

- Adaptation at the *lifted surface*. Though not explicitly part of the initial release of **MOSS**, adaptation can be supported. By specifying triangle/quadrilateral element at the *lifted surface* as well as the barycentric coordinates in that element, the strand mesh can be adapted. If the triangle is constructed from a *Convex Node*, then there is no explicit surface involvement, a new strand emanates from the single surface vertex so that the barycentric coordinates in the element at the *lifted surface* is pierced. If internal to the element, then 3 triangles are produced from the one. If the position is along a triangle side, then the neighboring element is also involved. If the neighbor is a triangle, then the 2 triangles are broken up along the shared side and 4 triangles result. If the neighbor is a quadrilateral, the triangle is broken into two and the quadrilateral is slit into 3 triangles.

For element stacks that map to surface elements, the barycentric coordinates described at the *lifted surface* are used on the corresponding surface element to query the geometry kernel for the actual normal (at that point). The new strand is created and if the barycentric coordinates indicate an internal position, the stack is split into 3 for triangles or into 4 triangles (for a quadrilateral). If the position is on a side, then like above, the neighboring element is also involved where quadrilaterals are broken into 3 triangles. For the situation where there is no neighbor (the *ONN* treatment) an error should be raised. At this point the new stacks need to be checked for collisions and smoothing applied when needed.

If the triangle comes from a *Convex Edge* segment and the barycentric coordinates reflects a position on the triangle-triangle side in the wedge, then the new strand emanates from the vertex below and the wedge is broken into 2 (with the neighbor appropriately split). If the barycentric coordinates indicate any other position on the lifted triangle then the Edge segment needs to be split. Under this circumstance the modification to the mesh is not entirely local and the entire stack (and neighboring stacks) may need to be rebuilt.

A final comment needs to be made about the absolute minimum memory usage shown by the strand mesh implementations; this is an engineering compromise that could be relaxed (at the expense of a larger solver memory footprint). For example, the global strand length could be made local (an additional floating-point word per strand) so that boundary layer coverage could

be better managed. One could even imagine using a 3D integral boundary layer code [6] to provide that individual strand length.

The strands could be curved. A quadratic (instead of linear) strand would cost an additional 3 floating-point words, in addition bending could be cubic at the expensive of 6 additional floating-point words per strand. Obviously a more complicated smoothing scheme would be needed that would also bend the strands when required.

Acknowledgements: This work was supported through NASA Award #NNX10AJ98G (“Geometric Control for Design Through Analysis”) where Michael J. Aftosmis acted as the technical monitor. Bob Meakin (CREATE-AV) provided the inspiration. William Chan (Nasa Ames) and Andrew Wissink (Army Rotorcraft) provided guidance. Romain Aubry (CREATE-MG) assisted in improving this paper.

References

1. Kallinderis, Y., and Ward, S., “Prismatic Grid Generation for Three-Dimensional Complex Geometries”. AIAA Journal Vol. 31, No. 10, October 1993.
2. Delanaye, M., Aftosmis, M.J., Berger, M.J., Liu, Y., and Pulliam, T.H., “Automatic Hybrid-Cartesian Grid Generation for High Reynolds Number Flows around Complex Geometries”. AIAA Paper 99-0777, 37th AIAA Aerospace Sciences Meeting and Exhibit, Reno NV, January 1999.
3. Meakin, R.L., Wissink, A.M., Chan W.M., Pandya S.A., and Sitaraman J., “On Strand Grids for Complex Flows”. AIAA Paper 2007-3834, 18th AIAA Computational Fluid Dynamics Conference, Miami FL, June 2007.
4. Wissink, A.M., Katz, A.J., Chan, W. M., and Meakin, R.L., “Validation of the Strand Grid Approach”. AIAA Paper 2009-3792, 19th AIAA Computational Fluid Dynamics Conference, San Antonio TX, June 2009.
5. Moro, D., Nguyen, C., Peraire, J., and Drela, M., “Advances in the Development of a High Order, Viscous-Inviscid Interaction Solver”. AIAA Paper 2013-2943, 21st AIAA Computational Fluid Dynamics Conference, San Diego, CA, June 2013.
6. Drela, M., “Three-Dimensional Integral Boundary Layer Formulation for General Configurations”. AIAA Paper 2013-2437, 21st AIAA Computational Fluid Dynamics Conference, San Diego, CA, June 2013.
7. Aubry, R., and Lohner, R., “Generation of viscous grids at ridges and corners”. Int. J. Numer. Meth. Engng; 77:1247-1289, 2009.
8. Ito, Y., Shih, A.M., Soni, B.K., and Nakahashi, K., “An Approach to Generate High Quality Unstructured Hybrid Meshes”. AIAA Paper 2006-0530, 44th AIAA Aerospace Sciences Meeting and Exhibit, Reno NV, January 2006.
9. Haimes, R., and Drela, M., “On the Construction of Aircraft Conceptual Geometry for High Fidelity Analysis and Design”. AIAA Paper 2012-0683, 50th AIAA Aerospace Sciences Meeting and Exhibit, Nashville, TN, January 2012.
10. “OpenCASCADE”, <http://www.opencascade.org>.
11. Haimes, R. and Follen, G., “Computational Analysis PRogramming Interface”. Proceedings of the 6th International Conference on Numerical Grid Generation in Computational Field Simulations, July 1998.
12. Wissink, A.M., Katz, A.J., and Sitaraman J., “PICASSO: A Meshing Infrastructure for Strand-Cartesian CFD Solvers”. AIAA Paper 2012-2916, 30th AIAA Applied Aerodynamics Conference, New Orleans, LA, June 2012.
13. Katz, A.J., and Wissink, A.M., “Efficient Solution Methods for Strand Grid Applications”. AIAA Paper 2012-2779, 30th AIAA Applied Aerodynamics Conference, New Orleans, LA, June 2012.
14. Wissink, A.M., Katz, A.J., Sitaraman J., Burgess, N., and Haimes, R., “Progress in Automatic Viscous Meshing from CAD Using Strand/Cartesian Meshes”. AIAA Paper 2013-3075, 21st AIAA Computational Fluid Dynamics Conference, San Diego, CA, June 2013.