A Mixed-Element 3D Advancing Layer Mesh Generator For Complex Geometries

Conference Paper · June 2017

DOI: 10.2514/6.2017-3453

2 authors:

Logan Numerow
University of British Columbia - Vancouver
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Carl Frederick Ollivier-Gooch
University of British Columbia - Vancouver
142 PUBLICATIONS **1,921** CITATIONS
SEE PROFILE

A Mixed-Element 3D Advancing Layer Mesh Generator For Complex Geometries

Logan Numerow*
Carl Ollivier-Gooch†

Advanced Numerical Simulation Laboratory
The University of British Columbia, Vancouver, BC, V6T 1Z4 Canada

With recent advances in 3D turbulent flow simulation come a growing need for high-quality automatic mesh generation. Advancing layer mesh generators are favored for their unmatched cell alignment, orthogonality and boundary layer resolution, yet full advancing layer meshing is almost nonexistant due to the challenges posed by complex geometry. This paper presents a fully automatic unstructured advancing layer mesh generation scheme suited for complex geometries in 3D, providing high-quality prism-dominant meshes throughout the domain. We focus on two critical but underdeveloped components of advancing layer mesh generation: a robust algorithm for processing front collisions, and a method for gradual simplification of the front in the transition from boundary layer to domain interior. The success of our method is evaluated for realistic flow configurations in 2D and in 3D, including a geometry used in a previous AIAA CFD Drag Prediction Workshop. We expect that the techniques developed here will be useful both independently and as improvements to modern advancing layer codes.

I. Introduction

In the early days of computational aerodynamics, structured meshes were the universal choice, and for good reasons. Computational resources limited the size and complexity of problems that were accessible, so geometries were simple enough for structured mesh generation to be relatively easy, including excellent resolution of boundary layers. Using these high-quality meshes (though coarse by modern standards), the pioneers of computational aerodynamics laid the foundations for much of today's solver technology.

As problem complexity — especially geometric complexity — grew, so too did the challenges in generating structured meshes. The 1980's saw the rise of unstructured mesh methods as an alternative to multiblock and overset structured meshes for complex geometries. Here, too, special measures were taken to ensure adequate boundary layer resolution with all-tet meshes. Notable among these efforts were the work of Pirzadeh[1] and of Marcum[2] in advancing layer mesh generation for high-aspect ratio cells.

Recent work in mesh generation for turbulent aerodynamic flows is returning full-circle, to generation of meshes that consist almost entirely of prisms or hexahedra within the boundary layer. Typically these schemes generate tetrahedra designed to be merged into non-simplicial cells and perform the merge as a post-processing step. Examples include the modern variants of Pirzadeh's and Marcum's mesh generators, as well as newer schemes.[3,4]

The motivations for the return to prism- and hex-dominant meshes are clear. Compared with tetrahedral meshes, prism/hex dominant meshes have lower cell count, better cell alignment and orthogonality, and

*Associate Professor, Mechanical and Aerospace Engineering, AIAA Associate Fellow.
†Professor, Mechanical Engineering, AIAA Associate Fellow.

elements that are more geometrically similar to each other. Taken together, these features improve solver performance, in terms of both computational resources and accuracy.

The previous work mentioned above focuses on maintaining prisms or hexahedra through the boundary layer. Marching typically terminates either after a fixed number of layers or when marching fails, which occurs when cells reach approximately unit aspect ratio or when marching fronts collide. This paper represents an effort to follow this trend to its logical conclusion: generation of prism-dominant meshes throughout the domain. We present a modified implementation and a three-dimensional extension of the two-dimensional scheme of Liu et al.[5] Our three-dimensional scheme is able to continue marching full layers of prisms even after some marching lines must be terminated (see section III). This allows us to generate meshes for external flows that are prism dominant all the way to the far field, retaining good mesh orthogonality, smoothness, and gradual changes in cell size and aspect ratio throughout. When meshing complex geometries, which often include multiple disconnected bodies, marching fronts frequently collide and overlap. In case of front collision, we re-mesh the overlapping neighborhood cleanly and produce a new front from which to continue marching (see section IV). In this paper, we provide results demonstrating the robustness of our method for meshing complex geometries, the ability to handle front collision cases, as well as the quality of meshes generated in realistic 2D and 3D flow configurations.

II. Main Procedure

As with other advancing front schemes, our method constructs the mesh in layers, marching from the boundaries into the domain interior. Meshes are quad-dominant with few triangles in 2D and prism-dominant with few pyramids and tetrahedra in 3D, such that faces are predominantly parallel to or othogonal to the boundary. The thickness of each layer can be selected to allow optimal resolution of flow physics; for viscous flows, we generate very thin boundary layers and high aspect ratio cells in order to accurately resolve high solution gradients normal to the domain boundaries. Typically, we select the thickness of the first layer t_0 and prescribe a per-layer growth ratio g, such that the thickness of the nth layer is as given by:

$$t_n = t_0 g^n \tag{1}$$

The use of a constant growth ratio between layers provides a mesh with smoothly varying cell size and aspect ratio throughout the domain, improving accuracy of numerical solutions.

Where anisotropic cells and/or increased resolution are desired in the domain interior, we define internal boundaries and march from them in both directions. We can, for example, better resolve turbulence in the wake of an airfoil if we march from an internal boundary along the wake line. This provides a framework that can be used to generate meshes with anisotropic features throughout the domain.

A. Surface Offsetting

The generation of each new front in the mesh can be framed as a surface offsetting problem,[6,7] but with an additional constraint. Since we wish to fill the space between layers with orthogonal quadrilaterals or prisms, the connectivity of the offset surface mesh must closely match that of the existing front, which rules out many existing methods. As such, we use a vertex-based surface offsetting routine that preserves mesh connectivity between layers.

Each vertex on the existing front (the *parent* node) generates a new vertex (the *child* node) offset from the parent by a specified distance in the surface normal direction. In 2D, we take the surface normal direction at a vertex to be simply the average of the normals of the two adjacent front edges. In 3D, we compute a weighted average of the adjacent face normals, where the weight of each face is the angle between the two edges shared by the face and the vertex. It is possible for this vertex normal not to be *visible* from one or more of the adjacent faces; that is, the dot product of the vertex normal and the face normal is negative. To avoid this, we use a normal computation by Aubry and Löhner[8] — later renamed the *most visible* normal — which guarantees a direction visible to all adjacent faces if one exists. This normal computation is much more expensive, and as such is used only where necessary to prevent an invalid mesh. The distance t_v between a parent node in layer n and its child node is given by equation 2. The geometry factor K_v, computed from relative neighboring vertex positions $\mathbf{d}_i$, increases smoothness of the new front by increasing marching distance from concave corners.

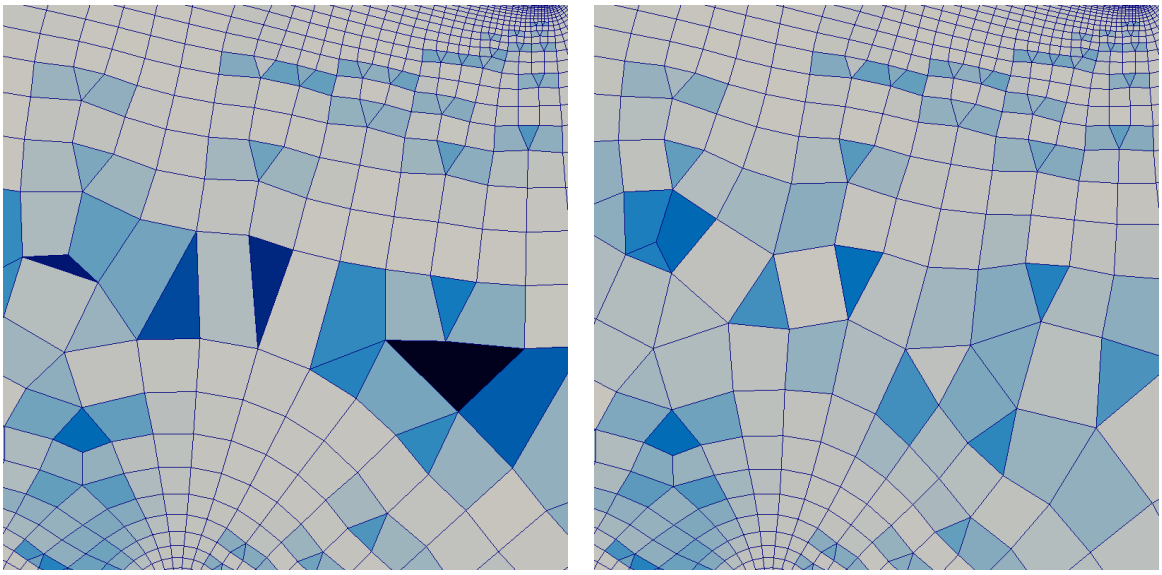
$$t_v = K_v t_n \tag{2}$$

$$K_v = 1 + \frac{1}{m}\sum_{i=1}^{m}(\mathbf{d}_i \cdot \hat{\mathbf{n}}_v)$$

Once the positions of nodes on the new front are determined, we add faces and edges with the same connectivity as on the parent surface to obtain an offset surface. In 2D, two adjacent parent vertices and their two children form a quadrilateral cell, which we can take to have been extruded from the edge between the two parent vertices. We can do the same in 3D, creating a quadrilateral face from each edge on the parent surface. A triangular prism is formed by the triangular parent and child faces and the three quadrilateral faces joining their edges. This procedure forms the basis of our advancing layer method, and is sufficient to generate a complete mesh for sufficiently simple geometries.

B. Smoothing

Mesh smoothing is essential for obtaining high-quality cells. Developing a general smoothing algorithm for highly anisotropic meshes is challenging because the desired mesh characteristics vary throughout the domain. Optimization-based methods[9] are applicable as long as appropriate quality metrics can be defined, but they are too expensive to be relied upon as extensively as needed here. Instead, we use a Laplacian-type smoothing algorithm based on the force model smoothing of Liu et al.[5]



(a) Before smoothing. (b) After smoothing.

Figure 1: Smoothing a front collision in 2D. Cells colored by scaled Jacobian quality metric.

The Algorithm iterates over mesh vertices, translating each along a net force vector computed from neighboring vertices and cells. The net force is the sum of several individual forces, each targeting a certain aspect of mesh quality, and as such this scheme can be easily modified to incorporate different quality metrics. The forces we use are as follows, where k are user-defined constants indicating the strength of each force.

1) **Adjacent force**. This moves a vertex toward the average of in-layer neighboring vertex positions, as in traditional Laplacian smoothing. We constrain this force, by projection, to act in the tangent plane at the current vertex location so as to prevent contraction of the front.

$$\mathrm{F}_{\mathrm{adj}} = \frac{k_{\mathrm{adj}}}{m}\sum_{i=1}^{m}[(\mathbf{v}_i - \mathbf{v}) - \hat{\mathbf{n}}(\hat{\mathbf{n}} \cdot (\mathbf{v}_i - \mathbf{v}))] \tag{3}$$

2) **Parent force**. Forces the edge length between the current vertex and its parent closer to the ideal spacing t_{n-1}.

$$\mathrm{F}_{\mathrm{par}} = -k_{\mathrm{par}}\hat{\mathbf{n}}_{\mathrm{par}}\frac{(|\mathbf{v}-\mathbf{p}|-t_{n-1})}{(|\mathbf{v}-\mathbf{p}|+t_{n-1})} \tag{4}$$

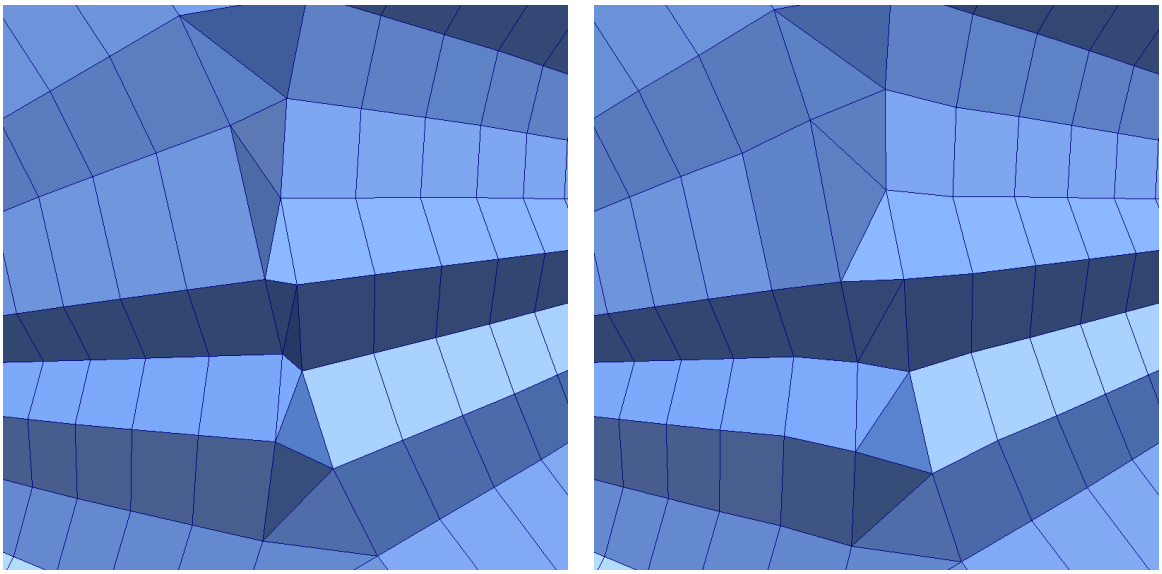
3) **Child force**. Fixes the edges between the current vertex and its child nodes. Here, we force the component of the edge length in the normal direction rather than the total edge length; combined with the parent force, this improves both the length and the orientation (orthogonality) of edges between layers.

   We use a broader definition of child node here, including any connected vertices in the next layer $n+1$ as well as vertices in the current layer n across a front merge. This is what causes the front spacing to vary smoothly in merge regions.

$$\mathrm{F}_{\mathrm{kid}} = \frac{k_{\mathrm{kid}}}{N_{\mathrm{kids}}}\sum_i \frac{(\mathbf{k}_i-\mathbf{v})}{|\mathbf{k}_i-\mathbf{v}|}\frac{(\hat{\mathbf{n}}\cdot(\mathbf{k}_i-\mathbf{v})-t_n)}{(\hat{\mathbf{n}}\cdot(\mathbf{k}_i-\mathbf{v})+t_n)} \tag{5}$$

4) **Restoring force**. Forces the node toward its original position. The strength of this force increases with distance from original position. This prevents other forces from distorting the mesh too much, allowing us to make them stronger in targeting their desired quality metrics. More important in anisotropic regions, this force can be relaxed toward the domain interior.

$$\mathrm{F}_{\mathrm{org}} = k_{\mathrm{org}}(\mathbf{v}_{\mathrm{org}}-\mathbf{v}) \tag{6}$$



(a) Before smoothing.

(b) After smoothing.

Figure 2: Smoothing a front collision in 3D.

We apply several iterations of smoothing after each new layer is placed, in order to obtain a clean front before generating the next layer. In practice, it is sufficient to iterate only over the last few layers of vertices. Unlike Liu et al,[5] we do not require convergence of the algorithm by force balance, as this tends to introduce unwanted artifacts while offering no additional guarantees about mesh quality. More effective and much less expensive is to smooth for only a few iterations after placing each layer. Figures 1 and 2 show the improvements made to mesh quality as a result of our smoothing routine, particularly in regions where advancing fronts collide and the structure of the mesh is disrupted. Cells in Figure 1 are colored by scaled

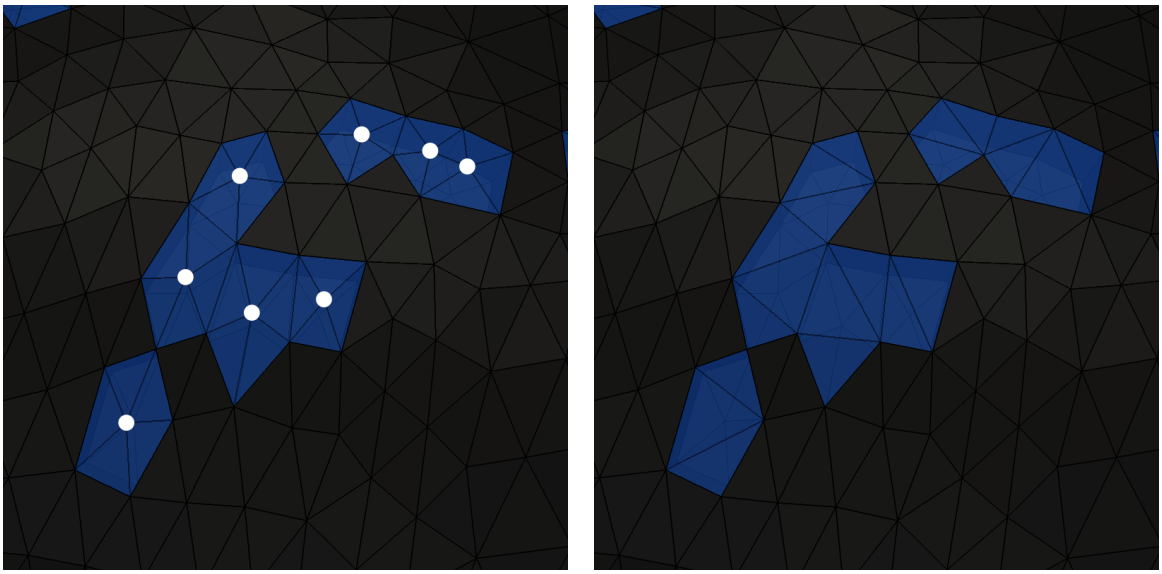
Jacobian in order to demonstrate the qualitative improvement in cell quality. Positive effects of smoothing on layer spacing and cell aspect ratio can be seen in both cases.

It should be noted that this smoothing method does not, explicitly or otherwise, necessarily avoid creating invalid mesh geometry. It is possible for vertices to move such that neighboring cells become inverted, though in practice the quality metrics targeted by the force model tend to avoid these configurations. More problematic, however, is the possibility for nearly-overlapping fronts to become overlapping after smoothing. Until opposing fronts collide (see section IV), vertices on opposite sides of the front are not topologically adjacent. Smoothing rules cannot account for close proximity of these vertices without a very expensive global analysis of the mesh. If smoothing pushes the fronts outward, it can cause them to overlap, creating an invalid mesh. To avoid this, we explicitly prevent smoothing from expanding the mesh into currently unmeshed regions.

III. Simplifying the Front

As we generate increasingly thick layers of cells, the aspect ratio of the cells decreases. Once aspect ratio reaches unity, we wish to maintain unit aspect ratio throughout the rest of the domain. To accomplish this, we remove vertices too close to neighboring vertices relative to the spacing between layers. This results in simplification from layer to layer that gradually reduces the density of vertices on the front. Without doing this, cells would become elongated in the surface normal direction as in meshes created by hyperbolic mesh generators. Solution gradients are typically larger in the surface normal direction, so this increased resolution along the streamlines adds cost to computations without improving their accuracy.

As per the method of Liu et al,[5] we can manage aspect ratio in two dimensions by collapsing edges on the front. We merge two adjacent points, removing the edge between them and collapsing the quadrilateral containing this edge into a triangle. This method, unfortunately, does not generalize to three dimensions. Collapsing a single edge on the top of a prism is not possible; the resulting cell has two quadrilateral faces that share three vertices, and as such has a degenerate corner. We may instead try collapsing an entire face, such that its prism cell becomes a tetrahedron; unfortunately, this collapses only a single edge of the neighboring prisms, so the problem remains. We recommend the removal of vertices as a less restrictive approach to front simplification that generalizes more naturally to higher dimensions. We describe first the approach to modifying the front surface, and proceed to illustrate how the volume mesh can be adapted consistently.



(a) Before decimation. Vertices to be decimated are marked with white circles.

(b) After Decimation.

Figure 3: Simplifying the front by vertex decimation.

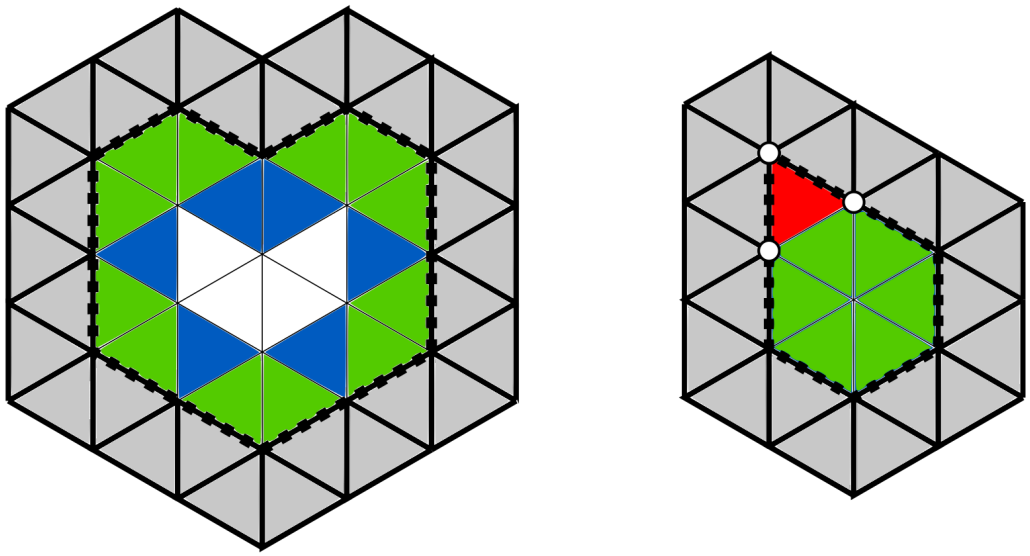
Removing a vertex and its adjacent faces from a surface mesh leaves a polygonal hole which must be trian-

gulated. Since we may wish to remove many nearby vertices in any given layer, these holes can be arbitrarily complex and can have one or many interior islands. Optimal triangulation of such holes is unrealistically expensive,[10] and we will not attempt this. Rather, to obviate this complexity, we employ a greedy algorithm using incremental vertex decimation.[11] One vertex is removed at a time, and its neighborhood — a simple polygon — is triangulated. We use the triangulation that minimizes total area, which tends to produce a smooth surface and avoid thin or degenerate triangles. This was found to perform qualitatively better than minimizing the maximum dihedral angle or the average dihedral angle. Though the optimal triangulation of a simple polygon has cubic time complexity[10] in the number of vertices, the number of vertices here is small (average 6) and does not increase with the number of vertices decimated, so we can consider the decimation of a single vertex to be a constant-time operation. The outcome of decimation of several vertices in a region is presented in Figure 3, with the affected faces displayed in blue. We will also refer to the simpler two-dimensional analog of decimation, which constitutes the removal of a vertex and the placement of a new edge between its two neighbors on the front. A vertex is selected for decimation if it satisfies one of two stop conditions:

1) The vertex is too close to an adjacent vertex, or to an edge connecting two of its adjacent vertices, relative to the local spacing between layers. Specifically, the distance to a neighboring vertex or edge, projected onto the tangent plane, is less than a given fraction of the layer spacing. This is the stop condition that works to maintain unit aspect ratio as layer spacing increases.

2) The edge connecting the vertex to its parent intersects a face on the front. Equivalently and more easily tested for, the direction from the vertex to its parent points outward from the vertex. This inverted geometry leads to an invalid mesh if not dealt with.

It is possible for the front to intersect its parent layer after decimation; for example, consider the decimation of a vertex at a sharp convex corner (unlikely, since such a vertex would not satisfy any of the stop conditions). This is extremely uncommon under normal circumstances, except in the case of tunnel closure (see section C) which requires special treatment. A simple improvement could be made to the robustness of the scheme; we could verify that each decimation is valid by checking that the new triangulation does not intersect underlying faces on the previous layer, and choose not to decimate the vertex if intersection occurs.



(a) Place pyramids on green faces and tetrahedra on blue ones.

(b) No valid way to close quadrilateral faces. Decimate any of the three marked vertices to fix.

Figure 4: Covering quadrilateral faces before Delaunay tetrahedralization.

Following decimation, we have obtained as desired a simplified surface, but we have portions of the front whose connectivity does not match that of the previous layer. Therefore, we cannot place orthogonal prisms

between corresponding faces in these regions. Let us first place each prism cell whose top triangular face still exists, leaving us only with gaps underneath pieces of the front where vertices were decimated. Similarly, in 2D, we place each quadrilateral cell whose top edge still exists. In 2D, the gaps are polygonal and are bounded by a set of edges, and they can be easily filled by triangulating the domain within these edges. Delaunay triangulation is recommended for the best initial cell quality, which will be further improved by smoothing. This approach for filling gaps is useful in part because it extends to higher dimensions; as a domain bounded by edges in 2D can be triangulated, a domain bounded by triangular faces in 3D can be tetrahedralized. The gaps in the 3D mesh are not bounded by only triangular faces, since their sides are formed by the quadrilateral sides of adjacent prisms, so we must first close off the quadrilateral faces before we can apply a tetrahedralization routine to fill the gap. This is done by placing a pyramid cell joining each quadrilateral face to the opposite vertex on the triangle underneath (see figure 4). Note the invalid configuration in4b, which can arise after front collision repair (section IV), where all three vertices of the top face are intact but the face itself has been removed. This must be corrected by decimating one of the three marked vertices. It is important to note that certain gap geometries in 3D will require the addition of one or more Steiner points during tetrahedralization. Fortunately, Steiner points can always be placed in the interior of the gap and thus do not affect existing mesh connectivity.

A. Degenerate Cases

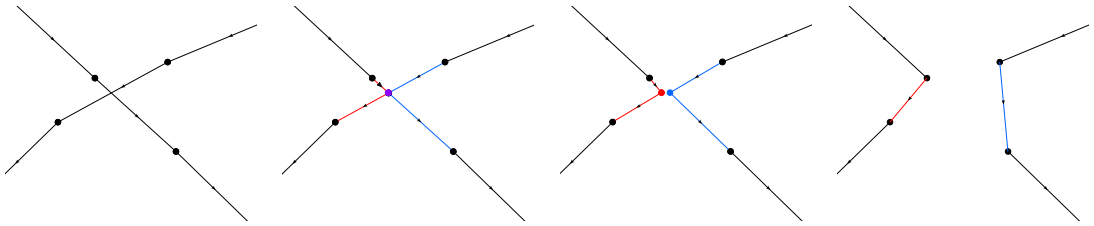
Most often when an inward-marching front component has nearly closed, we obtain a front that cannot be decimated any further without collapsing it. In 2D, the minimal front is a triangle of three vertices, while in 3D the minimal front is a tetrahedron of four. If a vertex on such a front must be decimated, we instead remove the front and close the gap completely. It is not fundamentally problematic to have a layer of only one or two vertices; an inward-marching front could reasonably converge to a point, for example. We avoid these degenerate fronts because our front collision repair algorithm requires that fronts are closed and oriented curves or surfaces. Fortunately, mesh quality in regions where we remove a degenerate front is not degraded; this arguably improves mesh quality by avoiding unwanted artifacts, such as excessively high-valence vertices.

IV. Front Collision

The advancing layer method has seen limited use for nontrivial geometries due to the difficulty of handling front collision. Especially when input geometries consist of multiple disconnected components, we inevitably encounter overlap or self-intersection in the front surface. To maintain a valid mesh, we must detect these collisions, repair the front to obtain manifold and non-intersecting surfaces, and mesh the space between the old and new valid fronts in the same manner as in section III.

A. Two Dimensions

The first step of our process, after the generation and simplification of the front, is to detect intersections between front edges and clip them at points of intersection, resulting in a front surface whose edges do not intersect but which contains non-manifold vertices (points with more than two incident front edges). A new vertex is added at each point of intersection, and the two intersecting edges are split.



(a) Intersection of two front edges.

(b) Edges split at point of intersection.

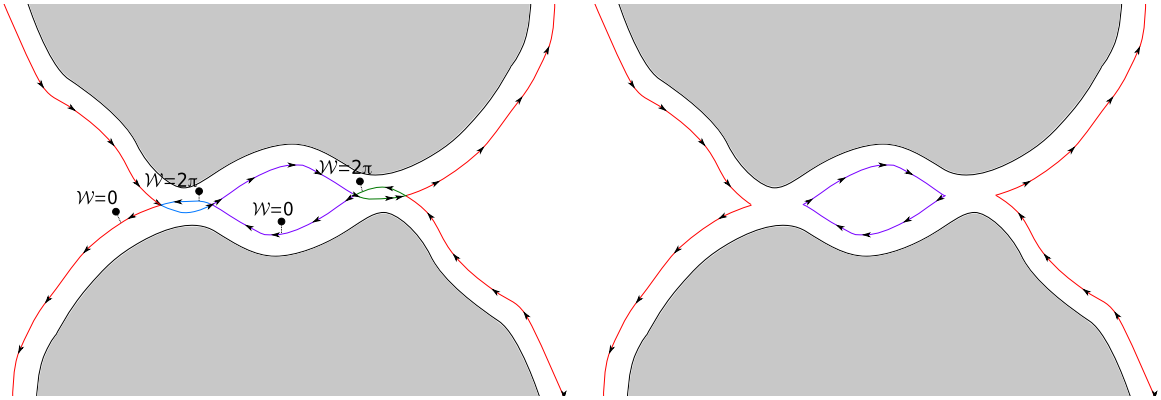
(c) Non-manifold vertex split into two.

(d) Decimation of vertices added in clipping.

Figure 5: 2D edge clipping.

To explain the remaining steps, we must establish the notion of edge orientation. We orient the front edges counterclockwise around the input geometry, such that the meshed region is to the left of the front edge and the unmeshed region is to the right. This information allows us to repair the front in a valid and consistent manner after encountering a front collision.

After clipping at points of intersection, we recover manifoldness of the front by disconnecting sections of the front joined by non-manifold vertices. At each of these points, we have two incoming edges (edges oriented towards the vertex) and two outgoing edges. We can form two incoming-outgoing edge pairs, connecting each incoming edge to the outgoing edge that was not part of the same original edge prior to splitting. The vertex is then split into two manifold vertices, with one connected to each edge pair.



(a) Winding number of zero indicates a valid component.

(b) Invalid components removed.

Figure 6: Identifying valid front components using winding numbers.

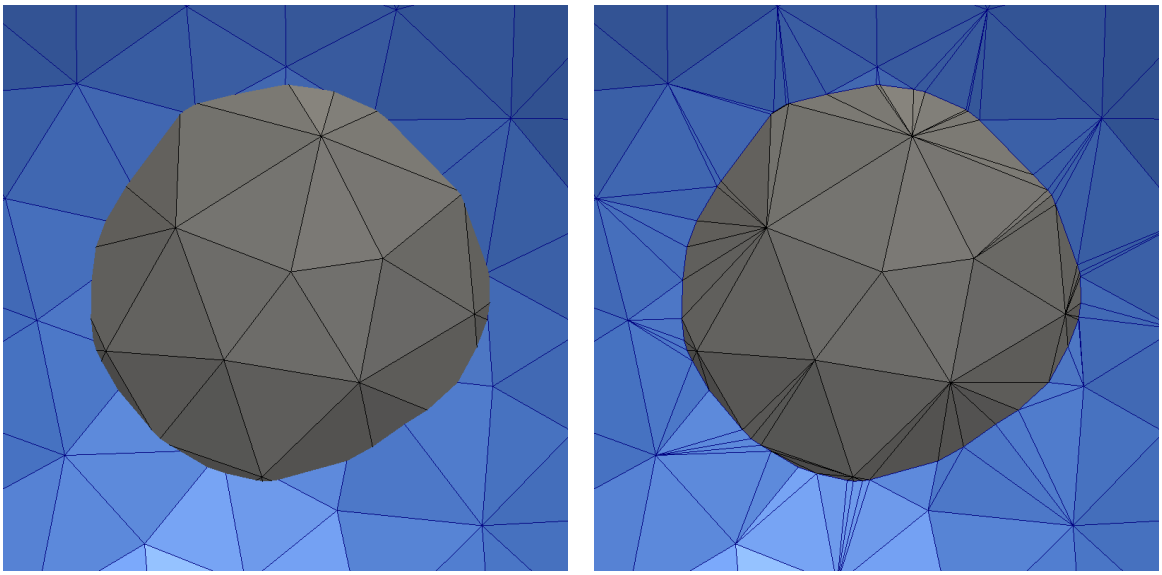
Once this reconnection step is applied at each non-manifold vertex, the front consists only of manifold, closed and non-intersecting surface components. Some components are valid new front surfaces, while others come from originally overlapping regions and must be removed. We distinguish between valid and invalid components using winding numbers,[12] and remove invalid components to obtain a valid front. For each disconnected component of the front, we select a point a small distance ϵ to the right of the midpoint of an arbitrary edge. Note that for the valid outer front component, this point lies in the unmeshed region outside of all front components. We compute the winding number of the entire front (the union of all valid and invalid components) around this point. If the winding number is zero, the component is valid; otherwise, the component is inverted and must be removed. This is demonstrated in Figure 6 for a simple case; the point inside the purple component has a winding number of zero because it is circled clockwise by the purple component and counterclockwise by the red one.

Finally, we decimate (see section III) the vertices added in clipping in order to avoid vertex clustering in the repaired regions. This step also serves to maintain the advancing front mesh topology; the vertices removed here do not have defined parent nodes, and as such they would require special smoothing rules.

B. Three Dimensions

The three-dimensional front collision repair, like the 2D method, begins with clipping the front surface at self-intersections. In 3D, the front is comprised of triangular faces. In general, the intersection of two faces is a line, and the intersection of an edge with a face is a point. To clip the fronts along curves of intersection, we insert new vertices at edge-face intersections and new edges along face-face intersections. Analogous to splitting intersecting edges in 2D, intersecting faces in 3D must be divided into smaller triangles, such that the triangulation respects the newly added clipping vertices and edges.

In 2D, the clipping process can be accomplished using only local operations, and edge intersections can be handled one-by-one. Unfortunately, to achieve this in 3D is more difficult. An attempt was made with Algorithm 1, and while the algorithm was successful in some cases, it failed to terminate in others. Each new edge introduced has the potential to intersect other faces or edges, such that clipping one intersection can create another, and we encountered intersecting configurations that could not be eliminated this way.



(a) Before clipping. (b) After clipping.

Figure 7: Clipping of overlapping fronts.

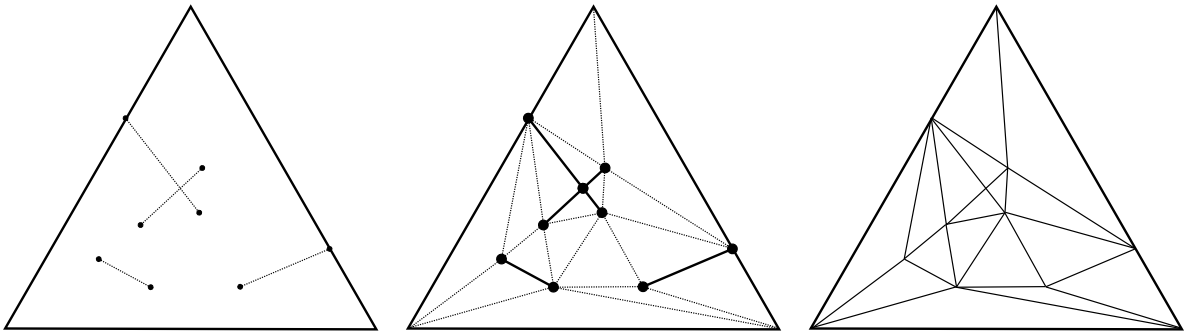
Algorithm 1 Localized 3D intersection clipping.

while at least one self intersection exists:
  if an edge intersects a face:
    Insert a vertex at the point of intersection.
    Bisect both faces adjacent to the intersecting edge.
    Split the intersecting face into three by adding an edge from each of its vertices to the new vertex.
  if an edge intersects an edge:
    Insert a vertex at the point of intersection.
    Bisect both faces adjacent to each intersecting edge.

To guarantee termination, we use instead a global method that detects all intersecting geometry prior to clipping. For each face, a list of intersection lines (indicated by their endpoints) is constructed. Note that each intersection line will appear on two lists, as an intersection line represents the intersection of two faces. Vertices are placed at endpoints of intersection lines. The insertion of new vertices in the mesh data structure is challenging here, because each new vertex will be an endpoint of several intersection lines; it is important to create each vertex once and only once. Also, if two intersection lines on a face are crossing, a new vertex must be inserted at the crossing point, which is actually a point where three faces intersect. Finally, each intersecting face is split into a triangulation with edges along all the intersection lines on the face (Figure 8). Since the faces are planar, we can safely project them into two dimensions and use a Delaunay triangulation routine. This global clipping method is challenging to implement and is less efficient than the localized algorithm, so it would be beneficial to find an improved localized algorithm that could guarantee termination. An example of clipping of overlapping fronts, viewed from inside one of the front surfaces, is shown in Figure 7.

The remainder of the 3D front collision repair process is a direct extension of the 2D scheme. The notion of face orientation is more natural here; the top of each front face is exposed to the unmeshed region while the bottom points toward the boundary. The orientation of a face is established by the ordering of the three associated vertices; the vertices are ordered counterclockwise around the top of the face, by convention. Note that at a manifold edge between vertices v_1 and v_2, the two adjacent faces will have opposite orderings for these vertices. That is, if one face has vertices ordered $v_1v_2v_3$, the other will have $v_2v_1v_4$. We say that two faces have *consistent orientation* at a non-manifold edge if they have opposite orderings for the edge vertices.

We next recover manifoldness by disconnecting the front into separate components, accomplishing this



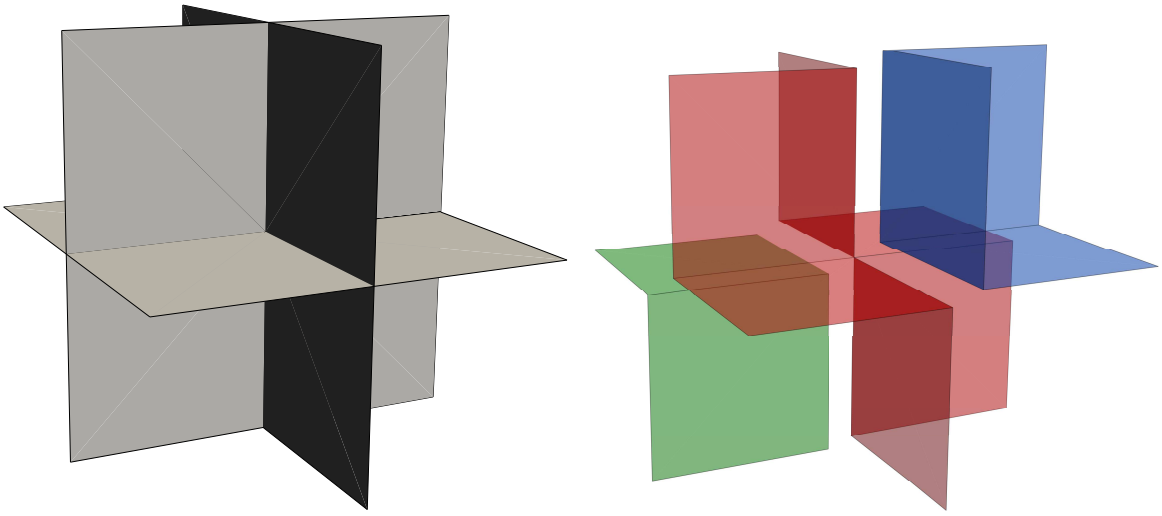
(a) Intersection lines found on triangular face.

(b) Additional vertex added at the crossing point of two intersection lines.

(c) Final triangulation of the face.

Figure 8: Subdivision of a face with multiple intersection lines.

by applying a local operation at each non-manifold vertex. In 3D, a manifold vertex has an adjacent *ring* of faces which make up a patch of the surface. At a non-manifold vertex, we can separate the adjacent faces into multiple rings, just as in 2D we separate into pairs the edges incident on a non-manifold vertex. We start at an arbitrary face adjacent to the vertex, and move across an edge shared by the face and the vertex to the next (opposite) face. We traverse a ring of faces by repeatedly moving to the next face until we return to the starting face. A non-manifold edge has not two but four adjacent faces, so we have three to choose from when moving to the next face across a non-manifold edge. The correct next face is the one that has consistent orientation with the current face but was not part of the same original face prior to clipping. Note that clipped faces which were part of the same original face before clipping will be perfectly parallel, and can be identified that way; it is not useful to store information about which faces were originally together. Once we have traversed a ring of faces, we make a copy of the vertex and reconnect these faces to it, separating the ring of faces from the non-manifold vertex. We continue to split off rings of faces until the non-manifold vertex has no more adjacent faces, at which point the non-manifold vertex has been replaced by several manifold vertices (in the same location) and can be deleted. An example is shown in Figure 9, in which a non-manifold vertex at the intersection of three planes is split into three separate neighborhoods.



(a) Non-manifold vertex at the intersection of three planes. Coloring indicates the orientation of the face.

(b) Faces partitioned into three neighborhoods, or rings, distinguished by color.

Figure 9: Splitting of a non-manifold vertex neighborhood in 3D.

All components of the front now are closed, manifold surfaces. Some components are valid new fronts, and others come from originally overlapping regions and must be removed. Existing methods, to identify

valid components, rely on heuristics that are not applicable to complex geometry.[13] However, we find that as in 2D the invalid components can be filtered out using winding numbers. The generalization of the winding number to three dimensions is established by Jacobson et al[14] and is calculated using the signed solid angle subtended by a surface around a point. For each disconnected component of the front, we select a point a small distance ϵ above the centroid of an arbitrary face, and compute the winding number of the entire front around this point. If the winding number is non-zero, the component is inverted and should be removed.

Finally, we decimate the vertices added in clipping. This is essential in 3D, because clipping introduces a high density of vertices along lines of intersection and creates high aspect ratio faces which should be removed to benefit mesh quality. These unwanted artifacts of clipping can be seen clearly in Figure 7.

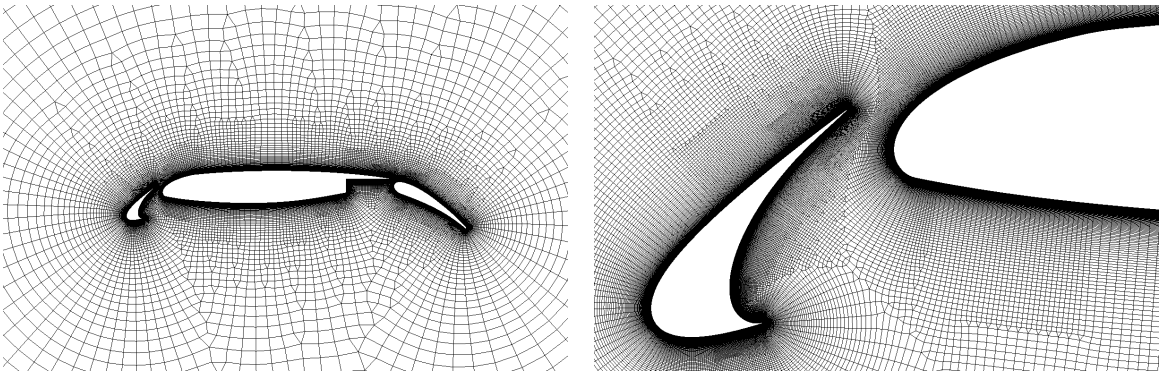
C. Tunnels

A particularly difficult case of front collision in three dimensions occurs when closing a topological tunnel, such as the hole of a torus. The difficulty comes not due to failure of our front collision algorithm, but because decimation (see section III) sometimes removes too many front vertices and causes the front to intersect the previous layer before the tunnel is completely closed. Due to this inconsistency, we recommend that tunnels be detected and closed using other methods. Dey et al[15] provide a method that identifies topological tunnels and computes the shortest path of vertices around them, including a software implementation. A loop of vertices around the tunnel is a polygon that can be triangulated[10] with a surface to close the tunnel. We can continue marching from both sides of a single surface closing the tunnel, or we can select two adjacent loops around the tunnel to triangulate in order to obtain a new front on each side.

V. Results

A. 30P30N Airfoil

Our 2D scheme was used to generate meshes for high-order turbulent flow solutions on the 30P30N multi-element airfoil. Three meshes of increasing resolution were used to assess the accuracy of the flow solver, with boundary layer thickness, growth ratio and number of layers chosen to simulate uniform mesh refinement in the medium and high resolution meshes. Figure 10 depicts the medium resolution 30P30N mesh, along with fine grade mesh detail.



(a) Medium resolution mesh overview.

(b) High resolution mesh detail.

Figure 10: 30P30N airfoil meshes.

Mesh quality statistics and mesh generation times are provided in Table 1. These results, and all subsequent results presented in this paper, are obtained using an Intel Core i7-860 processor at 2.80 GHz.

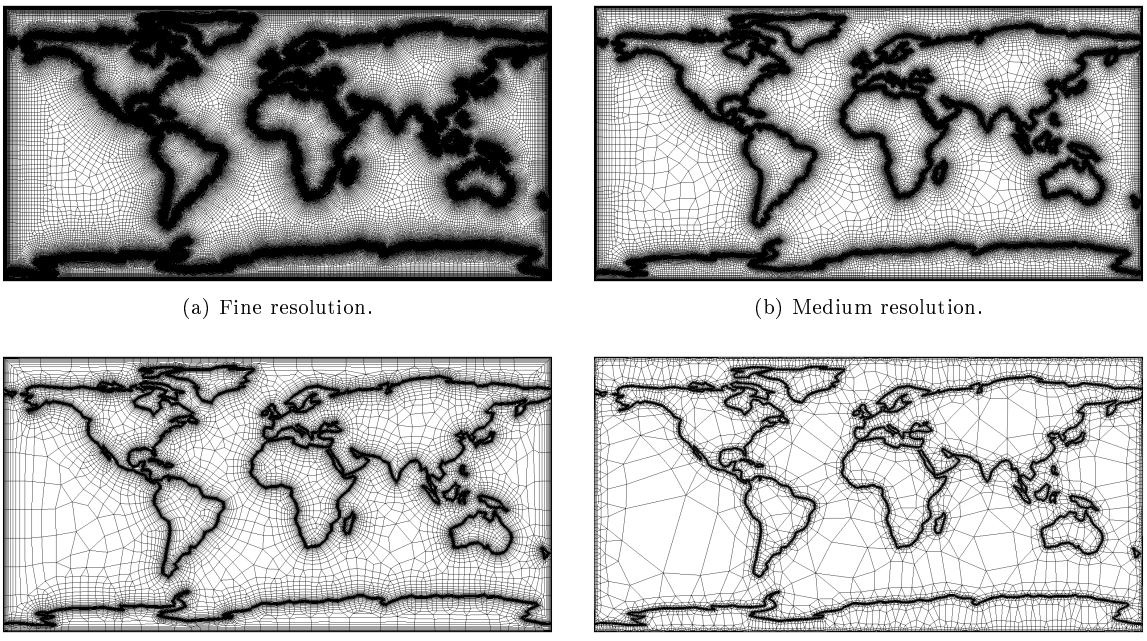
	Boundary Edges	Total Cells	Percentage Quads	t_0	g	Layers	Time (seconds)
Coarse	507	12413	97.0	0.00012	1.2000	44	-
Medium	986	47442	98.4	0.000057	1.0954	88	-
Fine	1950	184819	99.0	0.000028	1.0466	176	-

	Minimum Angle	$<40^o$	$<45^o$	$<50^o$	$<55^o$	Minimum Jacobian	< 0.6	< 0.7	< 0.8	< 0.9
Coarse	37.5	3	31	91	317	0.566	3	12	56	363
Medium	36.1	4	43	197	699	0.617	0	15	79	532
Fine	35.2	14	78	350	1425	0.579	1	32	147	978

Table 1: 30P30N mesh statistics.

B. Coastlines

To demonstrate the robustness of our method for complex 2D geometries, we use a meshing domain constructed from a map of the world's coastlines[a]. This geometry has a multitude of jagged internal boundaries, many of which have very small angles. Parallel boundary segments exist very close together — for instance, Central America, the Bosphorus Strait, and the Red Sea — and as such many front collisions must be resolved in each advancing front layer. Moreover, node spacing on the opposing fronts is often quite different, causing difficult merges when the fronts collide. The results in Figure 11 show that our method is capable of consistently handling all of these challenges and generating meshes with very high cell counts.



(a) Fine resolution.

(b) Medium resolution.

(c) Coarse resolution.

(d) Extreme growth ratio and boundary layer anisotropy.

Figure 11: 2D coastlines meshes.

Mesh 11c is notable for the regions in which opposing fronts collide with very different cell spacing, as can be seen in the top corners of the map as well as across the bottom in Antarctica. Mesh 11d demonstrates

[a]With many smaller islands removed.

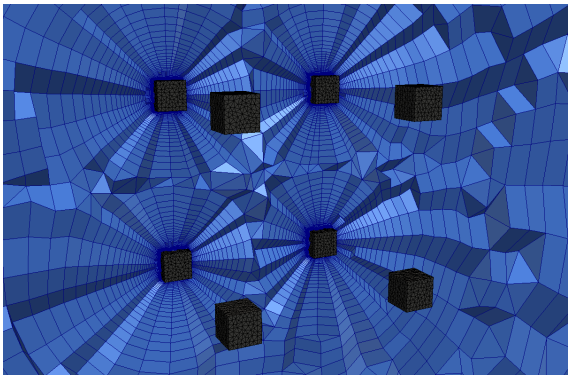
the ability of our scheme to handle extremely high growth ratios and was generated with $g = 2.0$. Typical growth ratios for CFD meshes are less than 1.2. It also has cells of aspect ratio up to 20,000 in the boundary layer, showing that we can handle extreme anisotropy. Finally, meshes 11a and 11b best demonstrate the consistency of our front collision repair algorithm due to the high concentration of cells and large number of advancing front layers.

	Boundary Edges	Total Cells	Percentage Quads	t_0	g	Time (seconds)
Fine	49915	2456314	98.87	0.001	1.1	302
Medium	38925	650635	96.56	0.01	1.2	93
Coarse	5048	50951	95.29	0.05	1.3	5.4
Extreme	49915	599127	95.31	0.0001	2.0	83

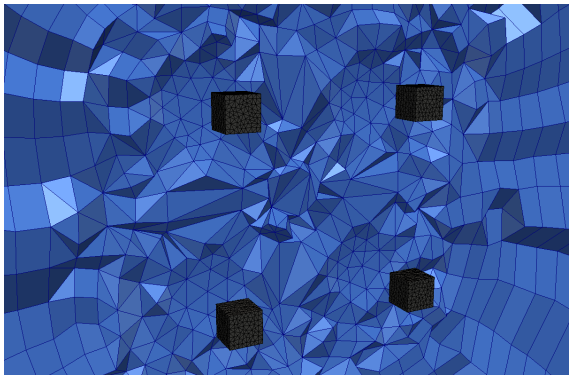
Table 2: 2D coastlines mesh statistics.

C. 3D Front Collision

Our first test geometry for the 3D code is a set of eight disconnected cube surfaces, arranged at the corners of a large cube centered at the origin. This test case highlights each of the three different front collision types that can be encountered in 3D. Firstly, we have front overlaps where the surfaces marching from each pair of adjacent cubes collide. Twelve of these occur simultaneously — one for each *edge* of the large cube. This leaves six topological tunnels — one for each *face* of the large cube. After these tunnels are closed, we are left with a closed cavity around the origin. We march inward and finally collapse the cavity, completing the mesh.



(a) Slice through an edge of the cube, showing front overlap and tunnel closure.



(b) Slice through the origin, showing tunnel closure and cavity collapse.

Figure 12: Medium resolution 3D cubes geometry mesh.

Most notable is the quality of the tetrahedra in the front overlap regions in Figure 12a. Our smoothing scheme forces the thickness of the tetrahedron layer to follow the prescribed growth ratio relative to the previous layers on both sides of the merge, and adjusts the positions of vertices on both sides to allow well-shaped tetrahedra. The effects of the collision do not propagate far outward as more layers are added to the mesh; rather, they are visible only within a few layers of the intersection.

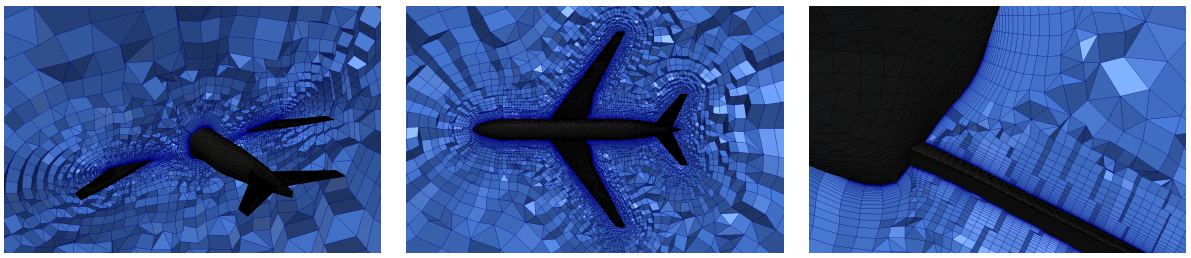
D. Drag Prediction Workshop Geometry

Finally, we present a realistic 3D airplane mesh to demonstrate the applicability of our scheme to modern CFD simulations. We use as input a surface mesh of the NASA Common Research Model transonic wing-body-tail model, generated for the 4th AIAA CFD Drag Prediction Workshop. Airplane geometries present particular

	Boundary Faces	Total Cells	Percentage Prisms	t_0	g	Layers	Time (seconds)
Coarse	10477	99624	93.74	0.05	1.16	35	11
Medium	20977	257509	95.07	0.01	1.14	50	31
Fine	54465	740351	95.30	0.005	1.12	65	94

Table 3: 3D cubes mesh statistics.

challenges, specifically at complex corners where the sharp trailing edge of a wing meets the fuselage. Though we are not able to handle these consistently, our method has successfully generated meshes for certain cases. Aubry and Löhner[16] have explored methods for generating boundary layer meshes at complex corners which could be integrated with our scheme for improved robustness.



(a) Slice through cross-section of the fuselage.

(b) Slice through the horizontal plane.

(c) Close-up view of marching from a concave corner.

Figure 13: DPW4 volume mesh.

Marching out from the concavity formed between the wing face and the fuselage illustrates the ability of our scheme to collapse the front cleanly and with well shaped tetrahedra, as can be seen in Figure 13. We see also the simplification of the front marching from the main body as the layer thickness increases and the aspect ratio tends toward unity. The mesh consists of 2,928,887 cells — 2,510,473 prisms (85.7%), 256,679 tetrahedra (8.8%) and 161,735 pyramids (5.5%) — and was generated in 338 seconds.

VI. Conclusion

In developing a consistent method of repairing front collisions and other overlapping geometry in 3D, this paper overcomes a significant obstacle in the generation of full advancing layer meshes and opens the door for further investigation of the method. Our results serve to demonstrate the viability of full advancing layer methods for complex 3D geometries, as well as the quality of meshes that can be generated fully automatically using such a scheme.

Future work could include integration with more robust methods for handling complex corners in the input geometry,[16] and with metric space computations for handling anisotropy in the tangent plane,[17] in order to enable automatic meshing of challenging high lift configurations and the like.

References

[1]Pirzadeh, S., "Unstructured viscous grid generation by the advancing-layers method," *AIAA Journal*, Vol. 32, No. 8, Aug 1994, pp. 1735–1737.

[2]Marcum, D. L. and Weatherill, N. P., "Unstructured grid generation using iterative point insertion and local reconnection," *AIAA Journal*, Vol. 33, No. 9, Sep 1995, pp. 1619–1625.

[3]Steinbrenner, J. and Abelanet, J., chap. Anisotropic Tetrahedral Meshing Based on Surface Deformation Techniques, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, Jan 2007, 0.

[4]Steinbrenner, J. P., chap. Construction of Prism and Hex Layers from Anisotropic Tetrahedra, AIAA AVIATION Forum, American Institute of Aeronautics and Astronautics, Jun 2015, 0.

[5]Liu, T., Wang, L., Karman, S. L., and Hilbert, B., chap. Automatic 2D high-order viscous mesh generation by Spring-Field and vector-adding, AIAA SciTech Forum, American Institute of Aeronautics and Astronautics, Jan 2016, 0.

[6]Malosio, M., Pedrocchi, N., and Tosatti, L. M., "Algorithm to Offset and Smooth Tessellated Surfaces," *Computer-Aided Design and Applications*, Vol. 6, No. 3, 2009, pp. 351–363.

[7]Kim, S.-J., Lee, D.-Y., and Yang, M.-Y., "Offset Triangular Mesh Using the Multiple Normal Vectors of a Vertex," *Computer-Aided Design and Applications*, Vol. 1, No. 1-4, 2004, pp. 285–291.

[8]Aubry, R. and Löhner, R., "On the 'most normal' normal," *Communications in Numerical Methods in Engineering*, Vol. 24, No. 12, 2008, pp. 1641–1652.

[9]Amenta, N., Bern, M., and Eppstein, D., "Optimal Point Placement for Mesh Smoothing," *Journal of Algorithms*, Vol. 30, No. 2, 1999, pp. 302 – 322.

[10]Zou, M., Ju, T., and Carr, N., "An Algorithm for Triangulating Multiple 3D Polygons," *Proceedings of the Eleventh Eurographics/ACMSIGGRAPH Symposium on Geometry Processing*, SGP '13, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2013, pp. 157–166.

[11]Schroeder, W. J., Zarge, J. A., and Lorensen, W. E., "Decimation of Triangle Meshes," *SIGGRAPH Comput. Graph.*, Vol. 26, No. 2, July 1992, pp. 65–70.

[12]Chen, X. and McMains, S., "Polygon Offsetting by Computing Winding Numbers," 2005.

[13]Jung, W., Shin, H., and Choi, B. K., "Self-intersection Removal in Triangular Mesh Offsetting," 2004.

[14]Jacobson, A., Kavan, L., and Sorkine-Hornung, O., "Robust Inside-Outside Segmentation using Generalized Winding Numbers," *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)*, Vol. 32, No. 4, 2013, pp. 33:1–33:12.

[15]Dey, T. K., Fan, F., and Wang, Y., "An Efficient Computation of Handle and Tunnel Loops via Reeb Graphs," *ACM Trans. Graph.*, Vol. 32, No. 4, July 2013, pp. 32:1–32:10.

[16]Aubry, R. and Löhner, R., "Generation of viscous grids at ridges and corners," *International Journal for Numerical Methods in Engineering*, Vol. 77, No. 9, 2009, pp. 1247–1289.

[17]Alauzet, F. and Marcum, D., "A Closed Advancing-layer Method with Connectivity Optimization-based Mesh Movement for Viscous Mesh Generation," *Eng. with Comput.*, Vol. 31, No. 3, July 2015, pp. 545–560.