Computational Algorithms for Tracking Dynamic Fluid-Structure Interfaces in Embedded Boundary Methods

K. Wang[1], J. Grétarsson[1], A. Main[1] and C. Farhat*[1,2,3]

[1] *Institute for Computational & Mathematical Engineering*

[2] *Department of Aeronautics and Astronautics*

[3] *Department of Mechanical Engineering*

Stanford University, Stanford, CA 94305, U.S.A

SUMMARY

A robust, accurate, and computationally efficient interface tracking algorithm is a key component of an embedded computational framework for the solution of fluid-structure interaction problems with complex and deformable geometries. To a large extent, the design of such an algorithm has focused on the case of a closed embedded interface and a Cartesian Computational Fluid Dynamics (CFD) grid. Here, two robust and efficient interface tracking computational algorithms capable of operating on structured as well as unstructured three-dimensional CFD grids are presented. The first one is based on a projection approach, whereas the second one is based on a collision approach. The first algorithm is faster. However, it is restricted to closed interfaces and resolved enclosed volumes. The second algorithm is therefore slower. However, it can handle open shell surfaces and underresolved enclosed volumes. Both computational algorithms exploit the bounding box hierarchy technique and its parallel distributed implementation to efficiently store and retrieve the elements of the discretized embedded interface. They are illustrated, and their respective performances are assessed and contrasted, with the solution of three-dimensional, nonlinear, dynamic fluid-structure interaction problems pertaining to aeroelastic and underwater implosion applications. Copyright

Received . . .

Prepared using fldauth.cls DOI: 10.1002/fld

Int. J. Numer. Meth. Fluids 2011; **00**:2–35
KEY WORDS: embedded boundary method, immersed boundary method, fluid-structure interaction, interface tracking, Eulerian
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/fld

1. INTRODUCTION

Debuted in 1972 for the coupled fluid-structure simulation of blood flows through elastic heart valves [1], immersed boundary methods have gained tremendous popularity during the last four decades in Computational Fluid Dynamics (CFD), under different names. These include embedded boundary [2], fictitious domain [3], and Cartesian [4] methods. All of these and other related methods which are popular nowadays for a large variety of flow simulations around fixed [4, 5, 3, 2, 6, 7], moving [9, 10, 11, 7], and deformable [1, 12, 13, 14, 15, 7] bodies, are collectively referred to in this paper as embedded boundary methods. They are particularly attractive for dynamic fluid-structure interaction (FSI) problems characterized by large structural motions and deformations [16] or topological changes [7], for which most alternative arbitrary Lagrangian-Eulerian (ALE) methods [17, 18, 19] are often unfeasible.

Embedded boundary methods simplify the gridding task as they operate on non body-fitted grids. Most of them are designed for computations on Cartesian grids [1, 9, 10, 12, 13, 14, 11, 15, 6], but some have also been tailored for computations on unstructured meshes [20, 7]. However, embedded boundary methods complicate the treatment of wall boundary conditions in general [4, 5, 3, 2, 9, 10, 11], and fluid-structure transmission conditions in particular [1, 12, 13, 14, 15, 7]. This is essentially because a non body-fitted CFD grid does not contain a native representation of the wet surface of the body of interest. To address this issue, embedded boundary methods for CFD rely on a variety of interface treatment techniques. Typically, these require tracking the position of the embedded interface (or collection of interfaces representing the entire wet surface of the body of interest) with respect to the non body-fitted grid. This paper focuses on this specific aspect of embedded boundary methods for CFD in the context of three dimensions and arbitrary interfaces and grids. For the

*Correspondence to: Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305, U.S.A.

interface treatment itself and load computation algorithms, the reader can consult, for example, reference [7] and the related reference [8]. For fluid-structure time-dependent coupling algorithms, the reader is referred to [16, 19] and the references cited therein.

A comprehensive computational tool for tracking a dynamic embedded interface takes as input the positions and connectivities of a non body-fitted Eulerian CFD grid and and a Lagrangian surface representing the interface. Usually, its output includes some or all of the following *instantaneous* information:

(1) Status of each CFD grid point identifying the medium in which it resides.

(2) Location of the closest points on the embedded interface to a selected set of CFD grid points.

(3) Identification of the intersection of the edges of the CFD and embedded interface.

So far, the computational methods described in the literature for computing the outputs highlighted above have focused primarily on closed embedded interfaces (e.g. surfaces of solid bodies) and Cartesian CFD grids [5, 9, 10, 11, 13]. The closed interface assumption has led to several algorithms for determining item (1) highlighted above that are based on inspecting the outward normals of simplices in the vicinity of a given grid point [5, 10, 11]. This assumption is limiting however as many FSI problems such as flapping wings and parachutes involve open thin shell surfaces. The Cartesian grid assumption simplifies the task associated with item (2) highlighted above. It has led to the development of both geometric [5, 9, 10, 11] and non-geometric [13] closest point transform algorithms. However, unstructured grids can be computationally advantageous even for embedded methods, particularly for complex geometries. Furthermore, most computational algorithms that have been proposed in the literature for computing the intersections of a given CFD grid and a given embedded interface are based on identifying the penetrating edges of the CFD grid as those edges whose end-points lie in different media [9]. However, such algorithms fail when the embedded interface is closed and intersected twice by one or several edges of the CFD grid (see Figure 5, Case II). This scenario is likely to occur when the body is thin, and the volume enclosed by its surface is underresolved by the CFD grid (see Section 4.1). It is detrimental to accuracy as it leads to the leaking of fluid through the body [7].

Prepared using fldauth.cls
DOI: 10.1002/fld

This paper focuses on filling the gaps identified above in the computational technology for tracking dynamic interfaces in embedded boundary methods for CFD. To this effect, it presents two robust computational algorithms for tracking a dynamic fluid-structure interface with respect to a three-dimensional, arbitrary CFD grid. Both are capable of delivering all tracking information outlined above. The first algorithm is based on a projection approach and presented in Section 3.1. It is robust and fast. However, it is highly accurate only when the embedded interface is closed, and its enclosed volume is sufficiently resolved by the CFD grid. The second computational algorithm is presented in Section 3.2. It is motivated by the methods proposed in [21, 12] for computational graphics. It is based on a collision approach, robust, and highly accurate whether the embedded interface is closed or open, and the volume it encloses is sufficiently resolved by the CFD grid or not. However, it is slower than its counterpart described in Section 3.1. Both computational algorithms exploit the concept of a bounding box hierarchy [22] to efficiently store and access the elements of a discrete interface. A parallel distributed implementation of this concept is described in Section 3.3. Finally, both proposed computational algorithms are illustrated in Section 5 with the solution of three-dimensional, nonlinear, dynamic FSI problems pertaining to aeroelastic and underwater implosion applications. Their respective performances are also assessed and contrasted in that section.

2. EMBEDDED COMPUTATIONAL FRAMEWORK FOR FLUID-STRUCTURE INTERACTION

A mathematical model for a three-dimensional dynamic FSI problem typically involves: (1) governing fluid equations within a fluid domain $\Omega_F^\star(t) \subset \mathbb{R}^3$, where t denotes time, (2) governing structural dynamics equations within a structural domain $\Omega_S(t) \subset \mathbb{R}^3$, (3) transmission conditions at the fluid-structure interface $\Sigma_E(t) \subset \Sigma_S(t)$, where $\Sigma_S(t) = \partial\Omega_S(t)$ denotes the structural domain boundary, (4) Dirichlet and/or Neumann boundary conditions at the remaining fluid and structural domain boundaries, and (5) initial conditions (at $t = 0$) for the fluid and structural state vectors. In the setting of an Eulerian embedded boundary method for CFD and FSI, $\Omega_F^\star(t)$ is replaced

DOI: 10.1002/fld

by the extended fluid domain $\Omega_F = \Omega_F^\star(t) \bigcup \Omega_S(t)$ which therefore includes $\Omega_S(t)$ as a fictitious fluid domain region and is time-invariant (see Figure 1). Furthermore, the governing fluid equations are formulated in the Eulerian framework, whereas the structural dynamics equations are typically formulated in the Lagrangian setting.

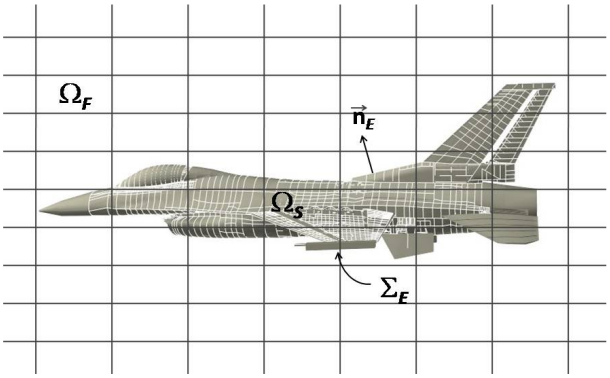

Figure 1. Domain setting of an Eulerian embedded method for fluid-structure interaction: extended fluid domain Ω_F, structural domain Ω_S, embedded surface Σ_E, and outward normal $\vec{n_E}$ to Σ_E.

The governing fluid and structural equations are coupled via appropriate transmission conditions at the fluid-structure interface. For example for an inviscid fluid and a flexible structure, these conditions can be written as [7]

$$\left(\vec{v} - \frac{\partial \vec{u}}{\partial t}\right) \cdot \vec{n_E} = 0 \qquad \text{on } \Sigma_E, \tag{1}$$

and

$$\left(\sigma_{ij} + \sigma_{im}\frac{\partial u_j}{\partial x_m} + p\delta_{ij}\right) \vec{n_E} - \mathcal{T}_j = 0 \qquad \text{on } \Sigma_E, \quad j = 1,\ 2,\ 3, \tag{2}$$

where $\vec{v}$ denotes the fluid velocity vector, p denotes the fluid pressure, $\vec{u}$ is the structural displacement vector, $\vec{n_E} = \vec{n_E}(t)$ is the outward unit normal to $\Sigma_E = \Sigma_E(t)$, $\sigma = \sigma(t)$ denotes the second Piola-Kirchhoff stress tensor of the structure, δ_{ij} denotes the Kronecker delta, and $\mathcal{T}_j$ denotes the tractions due to external forces whose origin is not due to the fluid.

Let $\mathcal{D}_h$ denote the discretization of the extended fluid domain Ω_F by an arbitrary non body-fitted Eulerian CFD grid, and let $\mathcal{D}_h^E$ denote the discretization by triangles of the fluid-structure interface

Prepared using fldauth.cls
DOI: 10.1002/fld

Σ_E. Note that $\mathcal{D}_h^E$ can, but does not need to, coincide with the wet subset of the discretization of Σ_S. In an embedded boundary method, the discrete fluid-structure interface $\mathcal{D}_h^E$ is immersed in $\mathcal{D}_h$ which otherwise does not contain a representation of this interface. For this reason, the implementation in a CFD solver of the semi-discretization of the transmission conditions (1) and (2) is not straightforward. It requires first tracking the position of $\mathcal{D}_h^E$ with respect to $\mathcal{D}_h$ — that is, generating some or all of the following information, depending on the specifics of the chosen interface treatment algorithm:

(1) Status of each CFD grid point identifying the medium in which it resides. For an FSI problem with a single fluid medium, this status determines whether a grid point belongs to the fluid domain or not. For an FSI problem with multiple fluid media, it also determines the specific fluid domain it belongs to.
(2) Location of the closest points on the embedded discrete interface $\mathcal{D}_h^E$ to a selected set of points in $\mathcal{D}_h$.
(3) Identification of the intersection of the edges of $\mathcal{D}_h$ with $\mathcal{D}_h^E$.

In general, item (1) above is needed in most embedded boundary methods for the purpose of fluid-structure demarcation. Item (2) is commonly used in ghost-cell based computational methods [11] for finding the "image" of a "ghost" node in the "real" fluid domain. Item (3) has been needed at least for two purposes: to define a surrogate fluid-structure interface where to enforce the transmission conditions [7], and to reshape boundary cells in a cut-cell based computational method [9].

Next, two different computational algorithms are presented for tracking a dynamic fluid-structure interface with respect to a three-dimensional, arbitrary CFD grid and generating the information outlined above. Both algorithms take as input the instantaneous position and connectivity of an embedded discrete interface and a non body-fitted CFD grid. Both operate independently from the specifics of the numerical scheme chosen for solving the coupled fluid and structural equations. For this reason, both algorithms can be incorporated into virtually any embedded boundary method for CFD. The following cycle of the simplest staggered solution procedure [24] equipped with the

Prepared using fldauth.cls
DOI: 10.1002/fld

embedded boundary method for CFD described in [7] illustrates the contribution of either algorithm to the solution of a fluid-structure interaction problem:

1. In the Computational Structural Dynamics (CSD) solver, send the updated displacement and velocity of the fluid-structure interface to the CFD solver.
2. In the CFD solver, update the position of the embedded discrete interface $\mathcal{D}_h^E$, and track it with respect to the CFD grid $\mathcal{D}_h$ using either ALGORITHM 1 or ALGORITHM 2 presented in Section 3. Then, compute edge-based fluid-fluid and fluid-structure fluxes as follows. For each edge (V_i, V_j) in $\mathcal{D}_h$ connecting the vertices V_i and V_j:

   2.1. if both V_i and V_j belong to the fluid domain $\Omega_F^\star$ and edge (V_i, V_j) does not intersect $\mathcal{D}_h^E$, compute the fluid-fluid flux between V_i and V_j as usual.

   2.2. if only one of the two vertices belongs to $\Omega_F^\star$, declare that (V_i, V_j) intersects $\mathcal{D}_h^E$. In this case, solve a one-dimensional fluid-structure Riemann problem between the "active" vertex and the interface. Then, compute a fluid-structure flux using the fluid interface state obtained from the Riemann solver (see [7] for details).

   2.3. if both V_i and V_j belong to $\Omega_F^\star$ and edge (V_i, V_j) intersects $\mathcal{D}_h^E$, solve a one-dimensional fluid-structure Riemann problem between each vertex and the interface, then compute a fluid-structure flux on each side of the interface.

   2.4. if neither V_i nor V_j belongs to the fluid domain $\Omega_F^\star$, do not perform any computation.

3. In the CFD solver, integrate the semi-discretized fluid equations from time t to time $t + \Delta t$, compute the fluid-induced load on a surrogate interface, distribute it on the computed fluid-structure intersection points, and send the distributed load to the CSD solver.
4. In the CSD solver, integrate the semi-discretized structural equations from time t to time $t + \Delta t$ using the flow-induced load received from the CFD solver.

Prepared using fldauth.cls
DOI: 10.1002/fld

3. DESIGN OF AN INTERFACE TRACKER

Two approaches are presented here for designing an interface tracker that meets the objectives outlined above. The first one is based on projections. It leads to a tracker that is "optimal" when the embedded discrete interface is a closed surface and the volume it encloses is resolved by the CFD grid. The second approach is based on collisions. It leads to an interface tracker that can handle an open surface such as a membrane sheet, and an underresolved enclosed volume such as that arising from a thin wing application. It also leads to an interface tracker that is in general more robust and accurate than that based on projections, but at the expense of a reasonable increase in computational complexity.

Both interface trackers discussed herein take the following input:

(1) Nodes (grid points) of an arbitrary three-dimensional CFD grid $\mathcal{D}_h$ — which can but does not have to be a Cartesian grid — and edge-to-node and node-to-node connectivities of this grid.
(2) Nodes, and element-to-node and node-to-element connectivities of an embedded discrete interface $\mathcal{D}_h^E$.

In addition, the projection-based approach requires the outward normals $\vec{n_E}$ (see Figure 1) to the geometric surface primitives of $\mathcal{D}_h^E$ which are assumed here to be triangles. This is a weak assumption as any n-noded surfacic element with $n > 3$ can be easily divided into several triangular elements.

Also, both interface trackers presented below take advantage of the bounding box hierarchy [22]. This technique is a popular form of spatial partitioning. It is used to rapidly determine whether a grid point of $\mathcal{D}_h$ lies near or far from the embedded discrete interface $\mathcal{D}_h^E$. The interface tracker based on the collision approach relies on a robust edge-triangle collision detection algorithm [23] to determine whether a given edge of $\mathcal{D}_h$ intersects a particular triangle of $\mathcal{D}_h^E$. In both cases, the computation of intersections is speeded up by the bounding box hierarchy technique which is used to eliminate for a given edge all but a few candidate triangles from the list of its potential intersectors in $\mathcal{D}_h^E$.

Prepared using fldauth.cls
DOI: 10.1002/fld

3.1. Projection-based approach

This interface tracking approach is based on the principle that only the edges of the CFD grid connecting a node lying in the fluid medium $\Omega_F^\star(t)$ and a node lying in the structural medium $\Omega_S(t)$ can be classified as an intersecting fluid-structure edge. Following this principle and assuming that (1) the embedded discrete interface has an interior and an exterior — which is equivalent to assuming that the embedded interface Σ_E is a closed surface, and (2) the interior volume it encloses is resolved by the CFD grid — that is, any edge of $\mathcal{D}_h$ that intersects $\mathcal{D}_h^E$ intersects it at a single point, the medium in which each node of a given edge of the CFD grid can be identified at each time-step of a simulation using orthographic projections and flood-fill as described in ALGORITHM 1 below:

ALGORITHM 1

1. For each grid point $V_i \in \mathcal{D}_h$, construct an axis-aligned bounding box b_i defined as the smallest axis-aligned box containing V_i and its adjacent nodes.
2. Construct an axis-aligned bounding box hierarchy B_E which stores the triangles of $\mathcal{D}_h^E$.
3. For each grid point $V_i \in \mathcal{D}_h$, determine if it lies close to $\mathcal{D}_h^E$, and if it does, find the location of its closest point on $\mathcal{D}_h^E$ and determine the status s_i of V_i. This step can be performed as follows:

    3.1. Set $s_i = -1$ to indicate that, so far, the status of V_i is unknown.

    3.2. Using the axis-aligned bounding box hierarchy B_E, find the set of candidate triangles $\mathcal{C}(V_i) \subset \mathcal{D}_h^E$ that may contain the closest point to V_i denoted here by V_i'. These are the triangles in $\mathcal{D}_h^E$ whose bounding boxes intersect b_i.

    3.3. If $\mathcal{C}(V_i) \neq \emptyset$, find the location of V_i' and compute $\phi(V_i', V_i)$, the signed distance from V_i' to V_i. If $\phi(V_i', V_i) > 0$, set $s_i = 0$ to indicate that V_i is inside the fluid domain $\Omega_F^\star(t)$. If $\phi(V_i', V_i) \leq 0$, set $s_i = 1$ to indicate that V_i is inside Ω_S and thus outside $\Omega_F^\star(t)$.

Prepared using fldauth.cls
DOI: 10.1002/fld

4. Determine the status of all remaining CFD grid points using a flood-fill algorithm as follows. For each grid point $V_i \in \mathcal{D}_h$, if $s_i \neq -1$, loop through its adjacent nodes $N(V_i)$. For each $V_k \in N(V_i)$, if $s_k = -1$, set $s_k = s_i$.
5. Compute the intersections between the edges of $\mathcal{D}_h$ and elements of $\mathcal{D}_h^E$ as follows. For each edge (V_i, V_j), if $s_i \neq s_j$, identify this edge as a fluid-structure intersecting edge. Then, cast a ray from V_i to V_j and find the intersection point on $\mathcal{D}_h^E$.

The critical components of ALGORITHM 1 described above are the search for a given grid point V_i of its closest point $V_i' \in \mathcal{D}_h^E$, and the calculation of $\phi(V_i', V_i)$. These are explained next in details.

3.1.1. Closest point on the embedded interface to a given CFD grid point To find V_i', the closest point to V_i lying in each triangle $T_k \in \mathcal{C}(V_i)$ is first determined. It is denoted here by $V_i^{(k)}$. To this end, V_i is projected onto the plane containing T_k. The projection point is uniquely determined by its barycentric coordinates (ξ_A, ξ_B, ξ_C) with respect to T_k, where A, B, and C denote the vertices of T_k (see Figure 2). If all three coordinates are non-negative, the projection point is $V_i^{(k)}$. If one or two coordinates are negative, the projection point lies outside T_k. In this case, $V_i^{(k)}$ is located either on the line containing an edge of T_k, or at one of its vertices. Therefore, V_i is reprojected in this case onto each edge of T_k corresponding to a negative coordinate. For example, if $\xi_A < 0$, V_i is projected onto the line determined by $\overline{BC}$. If the projection point ends up on an edge of T_k, it is $V_i^{(k)}$. Otherwise, the vertex of T_k that has the minimum distance to V_i is $V_i^{(k)}$. Finally, $V_i' \in \mathcal{D}_h^E$ is identified as

$$V_i' = \arg \min_{T_k \in \mathcal{C}(V_i)} \|V_i - V_i^{(k)}\|_2.$$

3.1.2. Signed distance between a CFD grid point and its closest point on the embedded interface The magnitude of $\phi(V_i', V_i)$ is simply $\|V_i - V_i'\|_2$, which is trivial to compute once V_i' has been determined. Hence, it remains only to find the sign of this distance. To this effect, three different cases must be treated separately:

(a) V_i' is inside a triangle $T_k \in \mathcal{C}(V_i)$.

Prepared using fldauth.cls
DOI: 10.1002/fld

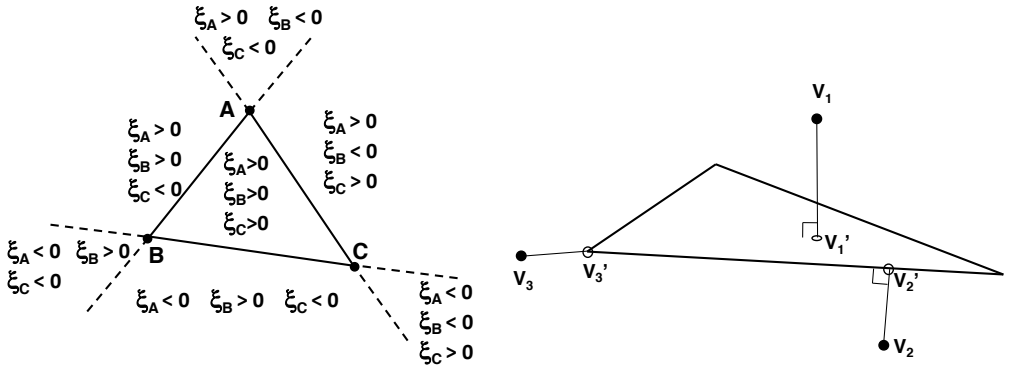

Figure 2. Signs of the barycentric coordinates of the projection point in different regions.

(b) V_i' is not inside a triangle $T_k \in \mathcal{C}(V_i)$ but is the projection of V_i onto an edge of a triangle $T_k \in \mathcal{C}(V_i)$.

(c) V_i' is neither according to case (a) or case (b), but is a vertex of $\mathcal{C}(V_i)$.

If V_i' falls in case (a), it is the projection of V_i onto the plane containing a triangle $T_k \in \mathcal{C}(V_i)$. In this case, $sign\left(\phi(V_i', V_i)\right)$ is determined as the sign of the dot product $\vec{n_k} \cdot \overrightarrow{V_i'V_i}$, where $\vec{n_k}$ is the unit outward normal to T_k and $\overrightarrow{V_i'V_i}$ denotes the spatial vector connecting the points V_i' and V_i.

If V_i' falls in case (b), it is the projection of V_i onto an edge of $T_k \in \mathcal{C}(V_i)$. In this case, the sign of $\phi(V_i', V_i)$ is determined using information from the two triangles of $\mathcal{D}_h^E$ sharing this edge. Two examples are shown in Figure 3. Using the notation shown in this figure, this case is handled as follows. First, V_i is projected onto the plane determined by one of the aforementioned two triangles that is not coplanar with it [†]. For example, suppose that Δ_{ABD} is that triangle, and P_{V_i} is the projection point. Then, vertex C in Δ_{ABC} is projected onto the same plane. The projection point is denoted by P_C. Next, the following scalar quantities s_v and s_c are introduced

$$s_v = \overrightarrow{P_{Vi}V_i} \cdot \vec{n_r},$$

$$s_c = \overrightarrow{P_C C} \cdot \vec{n_r},$$

[†] V_i cannot be coplanar with both triangles, otherwise it falls into case a or c.

Prepared using fldauth.cls
DOI: 10.1002/fld

where $\vec{n_r}$ is the unit outward normal to triangle Δ_{ABD}. Then if $s_v s_c > 0$ (see Figure 3–Left), $sign\big(\phi(V_i', V_i)\big)$ is determined as $sign(-s_v)$. Otherwise if $s_v s_c < 0$ (see Figure 3–Right), $sign\big(\phi(V_i', V_i)\big)$ is determined as $sign(s_v)$.

Finally, consider case (c) graphically depicted in Figure 4. The set of triangles adjacent to V_i' is denoted by $\mathcal{N}_i$ and shown in dark blue in Figure 4. Extended away from V_i', these triangles form an infinite open surface that is denoted here by $\tilde{\mathcal{N}}_i$ and colored in light blue in the same figure. Consider the plane crossing V_i and orthogonal to $V_i'\vec{V}_i$. For any point on this plane sufficiently far from V_i, its closest point on $\tilde{\mathcal{N}}_i$ is either on a face or on an edge (see Figure 4). In other words, this point, denoted here by $\tilde{V}_i$, falls into case (a) or case (b), and therefore $\phi(V_i', \tilde{V}_i)$ can be computed as discussed above. Finally, the sign of $\phi(V_i', V_i)$ is determined as the sign of $\phi(V_i', \tilde{V}_i)$.

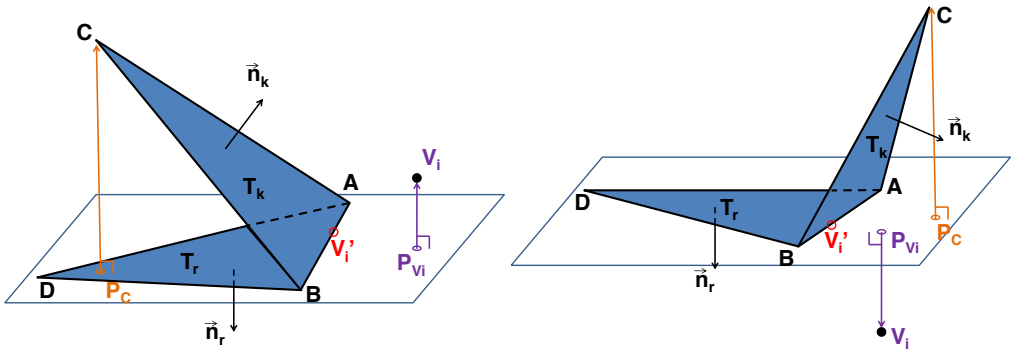

Figure 3. Determination of the signed distance $\phi(V_i', V_i)$ when V_i', the closest point to V_i on $\mathcal{D}_h^E$, lies on an edge of a triangle.

3.1.3. Remarks The following remarks are noteworthy:

- Since $\mathcal{D}_h$ is time-invariant, Step 1 of ALGORITHM 1 needs to be performed only once.
- Benefiting from the bounding box hierarchy constructed in Step 2 and the fast identification of the candidate triangles performed in Step 3.2, ALGORITHM 1 is an efficient algorithm with a computational complexity of the order of $O(N \log N_E)$, where N is the number of grid points in $\mathcal{D}_h$ and N_E is the number of triangles in $\mathcal{D}_h^E$.

Prepared using fldauth.cls
DOI: 10.1002/fld

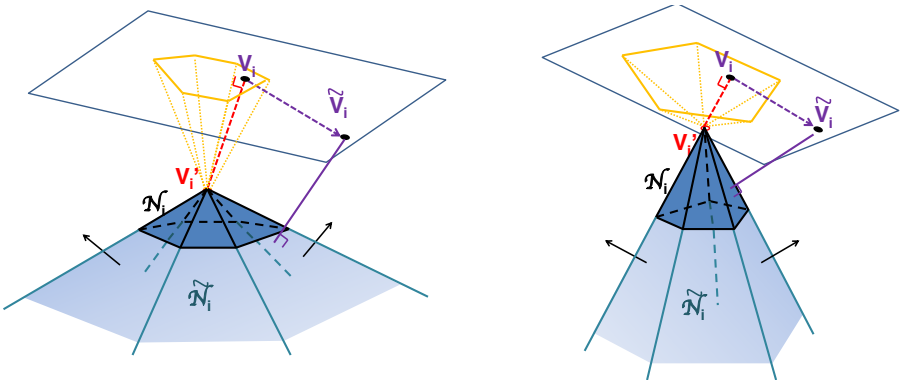

Figure 4. Determination of the signed distance $\phi(V_i', V_i)$ when V_i', the closest point to V_i on $\mathcal{D}_h^E$, is the vertex of a triangle.

- After the first time-step has been performed, Step 4 of ALGORITHM 1 can be accelerated by taking into account that the previous status of the CFD grid points is available as follows:

  4. For each grid point $V_i \in \mathcal{D}_h$, if $n_i = -1$, then $\mathcal{C}(V_i) = \emptyset$, which implies that V_i is far from Σ_E. For this V_i, set n_i to be the same as in the previous time-step.

- Step 5 of ALGORITHM 1 can be skipped if no intersection information is required by the embedded boundary method of interest.

3.2. Collision-based approach

When the embedded discrete interface $\mathcal{D}_h^E$ is a closed surface and the volume it encloses is not resolved by the fluid grid — for example, when the embedded interface is the wet surface of a thin wing and a single element of the CFD grid embeds pieces of both the upper and lower surfaces of the wing — or when $\mathcal{D}_h^E$ is an open surface like a membrane sheet, a different approach for tracking the interface becomes necessary. Indeed in the first case, the intersecting fluid-structure edge can connect two points that lie in the same fluid medium, and in the second case, there is no inside and outside to the embedded surface. To address the above issues, a robust geometric approach based on the point-simplex collision algorithm[23] is described here to determine whether an edge intersects $\mathcal{D}_h^E$ or not and provide the related interface tracking information.

Prepared using fldauth.cls
DOI: 10.1002/fld

The point-simplex collision algorithm is robust in the sense that it is guaranteed to never report a false negative — that is, to never have an edge erroneously miss $\mathcal{D}_h^E$. The trade-off for this robustness, however, is that such an algorithm occasionally reports false positives, either when an edge passes close to a sharp feature in $\mathcal{D}_h^E$, or when one grid point of an edge lies infinitesimally close to $\mathcal{D}_h^E$. The former case can be addressed by casting rays back and forth along the edge; if only one of these two rays intersects $\mathcal{D}_h^E$, it can be safely treated as a false positive and ignored. In the latter case where a grid point lies so close to $\mathcal{D}_h^E$ that determining numerically on which side of the interface it lies becomes challenging, the grid point can be flagged as "occluded". Then, every edge connected to this grid point can be automatically flagged as having intersected the embedded surface. In this work, occluded vertices are always considered to be inside Ω_S — even in the case of a membrane sheet.

3.2.1. Collision-based interface tracking algorithm The collision-based interface tracking algorithm can be described as follows:

ALGORITHM 2

1. For each CFD grid point $V_i \in \mathcal{D}_h$, construct its axis-aligned bounding box b_i.
2. Construct an axis-aligned bounding box hierarchy B_E that stores the triangles of $\mathcal{D}_h^E$.
3. Using the axis-aligned bounding box hierarchy B_E, find for each $V_i \in \mathcal{D}_h$ the set of triangles $\mathcal{C}(V_i) \subset \mathcal{D}_h^E$ whose bounding boxes intersect b_i.
   - 3.1 Thicken each triangle $T_k \in \mathcal{C}(V_i)$ by a numerical tolerance ϵ. If V_i lies inside any of the thickened wedges, flag it as occluded.
   - 3.2 Compute V_i' in the same manner as in Step 3.3 of ALGORITHM 1.
4. For every edge (V_i, V_j), cast a ray r_{ij} from V_i to V_j and another ray r_{ji} from V_j to V_i against the triangles in $\mathcal{C}(V_i) \cap \mathcal{C}(V_j)$, using the robust point-simplex intersection algorithm.
   - 4.1 If *both* r_{ij} and r_{ji} intersect a triangle in $\mathcal{D}_h^E$, classify the edge (V_i, V_j) as a fluid-structure intersecting edge and store the intersection point(s).

Prepared using fldauth.cls
DOI: 10.1002/fld

4.2 If *either* V_i or V_j is occluded, classify (V_i, V_j) as a fluid-structure intersecting edge and store the occluded node as the intersection point.

5. Determine the node status n_i using geometric means and its value at the previous time-step. If $\mathcal{C}(V_i) = \emptyset$, then keep n_i unchanged.

   5.1 For every triangle $T_k \in \mathcal{C}(V_i)$, use the point-simplex collision algorithm to determine whether T_k crosses over V_i during the given time-step. In this usage of the point-simplex collision algorithm, the point is fixed in space while the simplex travels from its position at time t^n to its position at time t^{n+1}. If any simplex crosses over V_i, set $n_i = -1$, indicating that the status may have changed and therefore must be redetermined.

   5.2 For every $V_i \in \{V_i \in \mathcal{D}_h : n_i = -1\}$, search for *visible* adjacent nodes V_j with $n_j \neq -1$ (here, a node V_i is said to be visible if (V_i, V_j) is not a fluid-structure edge). If such a node is found, set $n_i = n_j$ and repeat this procedure until the status of every node has been determined or no further updates are possible. Flag any remaining node V_i with a status $n_i = -1$ as being inside Ω_S.

3.2.2. Remarks The following remarks are noteworthy:

- Since $\mathcal{D}_h$ is time-invariant, Step 1 of ALGORITHM 2 needs to be performed only once.
- In this work, ϵ is set to 10^{-8}.
- Step 3.2 of ALGORITHM 2 can be skipped if V_i' is not needed by the embedded boundary method of interest.
- In Step 4, when either $\mathcal{C}(V_i)$ or $\mathcal{C}(V_j)$ is $\emptyset$, no ray is cast.
- At the beginning of the simulation, the status of a node is unknown. Hence, at $t = 0$, Step 5 is replaced by

  5 Perform a flood-fill on $\mathcal{D}_h$ in order to identify connected components that are separated by fluid-structure edges. These connected components are classified by user-provided information. The connected components that are not explicitly classified are flagged as being inside Ω_S.

Prepared using fldauth.cls
DOI: 10.1002/fld

Figure 5 illustrates the treatments by ALGORITHM 1 and ALGORITHM 2 of three different realistic cases.

In Case I, the embedded discrete interface is a closed surface (only part of it is drawn). The volume it encloses is resolved by the CFD grid. In this case, both algorithms give the same results.

In Case II, the embedded discrete interface is also a closed surface. However, the volume it encloses is not fully resolved by the CFD grid. In this situation, ALGORITHM 1 misses two edges that intersect the interface twice. These edges with double intersections are detected however by ALGORITHM 2.

In Case III, the embedded discrete interface is an open surface. In this case, ALGORITHM 1 fails to detect the correct intersections and attribute the correct statuses. On the other hand, ALGORITHM 2 delivers the correct results for all three types of tracking information.

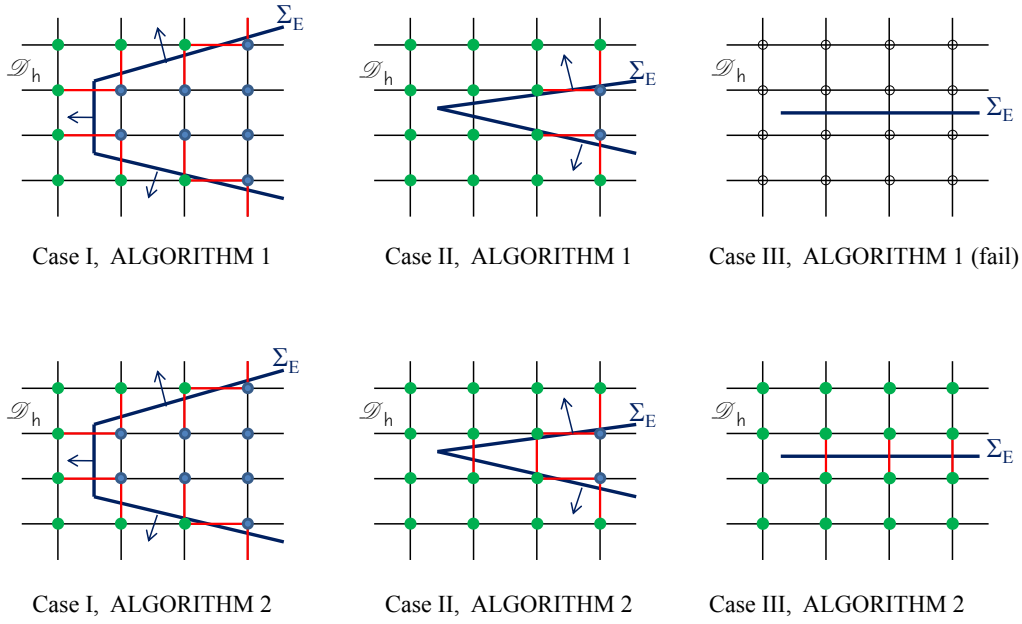

Figure 5. Illustration of ALGORITHM 1 and ALGORITHM 2 for three distinctive cases. A point in green (blue) color represents a CFD grid point lying the fluid (structure) region of the computational domain. An edge in red represents a fluid-structure intersecting edge.

Prepared using fldauth.cls
DOI: 10.1002/fld

3.3. Distributed bounding box hierarchy (scoping)

Both ALGORITHM 1 and ALGORITHM 2 described above call for the computation of the bounding box hierarchy B_E for the embedded discrete interface. For dynamic problems, B_E is time-dependent. For complex geometries or sufficiently refined discretizations, the computational cost associated with its construction is not negligible. Most importantly, for a massively parallel distributed CFD simulation where the computational domain is typically decomposed into subdomains, triangles in the embedded discrete interface lying far outside a given CFD subdomain are irrelevant for that subdomain. Therefore, adding them to the hierarchy is only detrimental to computational efficiency. Hence, the key idea here is to implement a distributed bounding box hierarchy for massively parallel computations where the hierarchy is constructed only over triangles near a given fluid subdomain, thereby creating a scope consisting only of triangles that are relevant to this particular subdomain.

Consider the simple two-dimensional example shown in Figure 6. Here, the extended computational fluid domain Ω_F, which includes the structural domain Ω_S shown in this figure as an oval, is decomposed in 16 subdomains for parallel computations, for example, on 16 processors. In the global bounding box hierarchy, the hierarchy is computed in each processor for the entire interface. However in the distributed bounding box hierarchy, only the component of the interface that is near a given subdomain is added to that subdomain's hierarchy. This scoping is illustrated in Figure 6 whose right side focuses on subdomain 10 and the component of the interface considered to be near it. Scoping reduces the size, construction CPU time, and query CPU time of the bounding box hierarchy.

Constructing a distributed bounding box hierarchy for either ALGORITHM 1 or ALGORITHM 2 begins with constructing a global bounding box hierarchy on each processor where all elements of $\mathcal{D}_h^E$ are assumed to be stored. Then, this hierarchy is used in the first time-step to identify in each subdomain — by the processor assigned to this subdomain — the triangles of $\mathcal{D}_h^E$ that are near each grid point of this subdomain (the candidate triangles). Next, the candidate triangles are gathered on a subdomain basis, each forming a scope or relevant component of the embedded discrete interface for

Prepared using fldauth.cls
DOI: 10.1002/fld

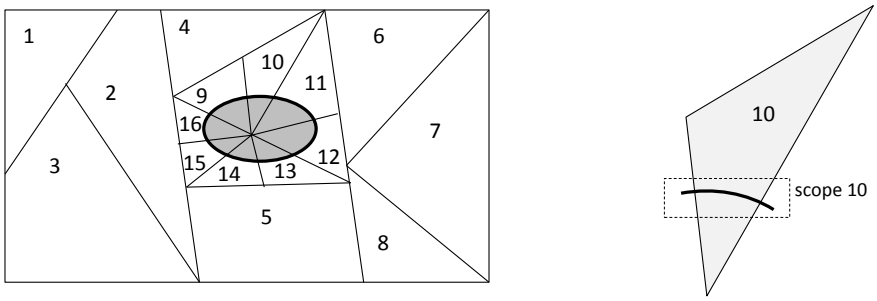

Figure 6. CFD subdomains and distributed bounding box hierarchy: scoping for subdomain 10 (right).

that subdomain. In the second time-step, the global bounding box hierarchy stored in each processor is replaced by its distributed counterpart where only the triangles within the scope of the subdomain mapped to this processor are stored.

In a dynamic fluid-structure simulation, the embedded discrete interface moves and/or deforms in time. Therefore in such a simulation, the scope of each subdomain must be updated, in principle, at every time-step. Assuming that the coupled fluid-structure analyzer obeys a time-step restriction that typically restrains $\mathcal{D}_h^E$ from crossing more than one layer of elements of the CFD grid $\mathcal{D}_h$ in a given time-step, the updated subdomain scope needs to include only: (1) the current candidate triangles for each grid point in the interior of this subdomain, and (2) any additional candidate triangle for each grid point at the boundary of this subdomain. To obtain the latter information, interprocessor communication is needed only locally — that is, between processors assigned to neighboring subdomains.

4. APPLICATIONS AND PERFORMANCE ASSESSMENT

The computational algorithms for interface tracking described in the previous sections were implemented in the compressible CFD solver AERO-F [17, 18] which is equipped with low-Mach preconditioning and both ALE [17, 18] and embedded boundary [7] computational frameworks.

Prepared using fldauth.cls
DOI: 10.1002/fld

Here, these algorithms are demonstrated on three different three-dimensional dynamic fluid-structure interaction problems.

First, the simulation of the heaving of a rigid thin wing is considered for the purpose of verification of AERO-F's embedded boundary method for CFD. This simulation also illustrates Case II of Figure 5. Verification is performed by comparing the CFD results obtained using the embedded computational framework of AERO-F equipped with both ALGORITHM 1 and ALGORITHM 2 for interface tracking, with those obtained using AERO-F's ALE framework which was successfully verified and validated in the past for many realistic applications [16, 17, 18]. Next, an underwater fluid-structure implosion problem for which experimental data is available is considered for the purpose of validation. This problem features large structural deformations and strong shock waves initiated at the fluid-structure interface, which challenges both interface tracking algorithms and embedded boundary methods for CFD. It also illustrates Case I of Figure 5. Finally, the aeroelastic simulation of the flapping of a pair of extremely thin and flexible wings is considered. This problem illustrates Case III of Figure 5. It also highlights the capability and robustness of the interface tracking ALGORITHM 2 with respect to open interfaces and large structural deformations.

In all problems outlined above, $\mathcal{D}_h$ is an unstructured tetrahedral mesh. The second and third problems are two-way coupled fluid-structure interaction problems. To solve them, the CFD solver AERO-F and the structural analyzer AERO-S [17, 18] are coupled using a provably second-order partitioned procedure similar to those described in references [24, 25, 26, 27, 16].

Finally, it is noted that all computations are performed in double-precision arithmetic on a parallel Linux cluster system.

4.1. Verification for a transient subsonic flow past a heaving rigid wing

Here, the problem of computing the unsteady inviscid airflow past a rigid wing in heaving motion is considered. The wing has a root chord length $L_c = 22.0$ in, a semi-span $L_s = 30.0$ in, a tip chordlength $L_t = 14.5$ in, and a quarter-chord sweep angle of 45^o. Its panel aspect ratio is equal to 1.65 and its taper ratio is equal to 0.66. Its airfoil section is the NACA 65A004. Hence, this wing

Prepared using fldauth.cls
DOI: 10.1002/fld

is quite thin. Consequently, it is difficult to construct a non body-fitted CFD grid for this inviscid flow problem which resolves the volume enclosed by the wet surface of the wing without being unnecessarily too fine for the external flow computations. In other words, this problem is quite challenging for interface tracking algorithms.

The wet surface of the wing is discretized with $20,721$ grid points and $41,438$ triangles (Figure 7–Left). This discrete representation $\mathcal{D}_h^E$ is embedded in a non body-fitted CFD grid $\mathcal{D}_h^{NBF}$ with $105,030$ grid points and $609,576$ tetrahedra (Figure 7–Right).

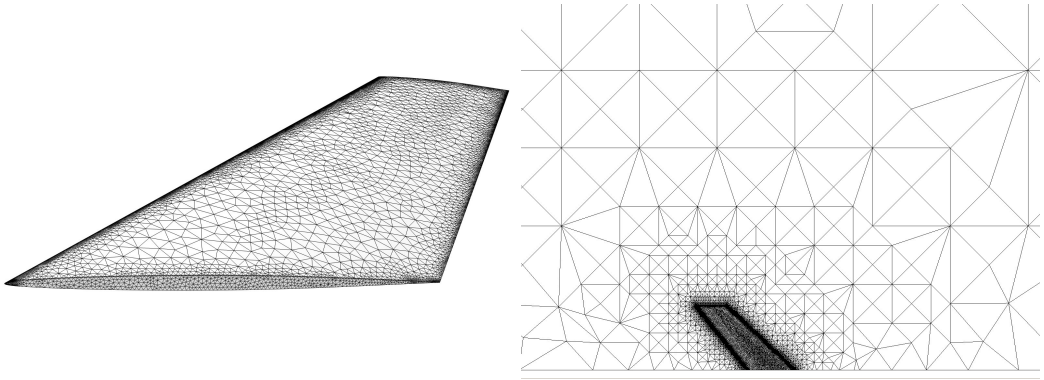

Figure 7. Thin wing in heaving motion: embedded discrete interface (Left) — cutview at $z = 0$ of inviscid non body-fitted CFD grid $\mathcal{D}_h^{NBF}$ (Right).

Before performing any flow computation, the performance of the interface tracking algorithms described in this paper is assessed by computing the intersections between $\mathcal{D}_h^{NBF}$ and $\mathcal{D}_h^E$. To this effect, Figure 8 displays the edges of $\mathcal{D}_h^{NBF}$ that are flagged as fluid-structure intersecting edges by ALGORITHM 1 and ALGORITHM 2. The reader can observe that, as anticipated, ALGORITHM 1 misses large swaths of the structure, essentially because the CFD grid does not resolve the structure everywhere. Hence, this is the same scenario as Case II of Figure 5. More specifically, ALGORITHM 1 does not recognize the CFD grid edges that intersect the embedded discrete interface twice — at both the upper and lower surfaces of the wing — as intersecting edges, because both vertices of each edge share the same status (see Step 5 of ALGORITHM 1).

Next, the concept of scoping for constructing a distributed bounding box hierarchy presented in Section 3.3 is illustrated. To this end, Figure 9 shows the scope decomposition of the embedded

Prepared using fldauth.cls
DOI: 10.1002/fld

discrete interface for a partitioning of the CFD grid in 32 subdomains. Compared to the $41,438$ triangles in $\mathcal{D}_h^E$, the largest subdomain scope has only $4,278$ triangles.

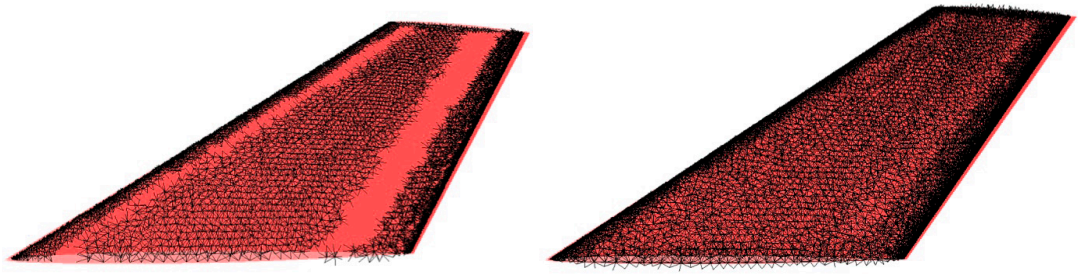

Figure 8. Thin wing in heaving motion: fluid-structure intersecting edges found by ALGORITHM 1 (Left) and ALGORITHM 2 (Right).

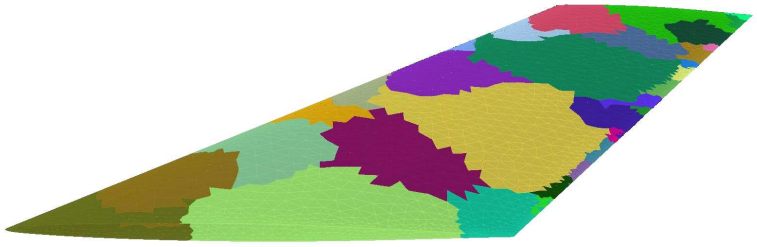

Figure 9. Thin wing in heaving motion: scope decomposition of the embedded discrete interface for 32 fluid computational subdomains (each color designates a relevant component of the interface for a specific subdomain for which a bounding box hierarchy is computed on the assigned CPU).

Next, the wing is set in the harmonic heaving motion characterized by the amplitude $h_a = 0.05$ in and the frequency $h_f = 500$ Hz. The free-stream conditions (Mach number, angle of attack, density, and pressure) are set to $M_\infty = 0.3$, $\alpha_\infty = 0^{\text{deg}}$, $\rho_\infty = 9.357255 \times 10^{-8}$ (lb/in^4).sec^2, and $p_\infty = 14.5$ psi, respectively. For the purpose of verification of the results generated by the embedded boundary method for CFD equipped with both interface tracking algorithms described in this paper,

Prepared using fldauth.cls
DOI: 10.1002/fld

a body-fitted grid $\mathcal{D}_h^{BF}$ with $896,167$ grid points and $4,664,720$ tetrahedra is also generated for the computation of an ALE reference solution of this problem.

Three *implicit* numerical simulations are performed for a time-interval corresponding to roughly five periods of the heaving motion:

1.1. A first simulation using the CFD grid $\mathcal{D}_h^{NBF}$, the embedded boundary method, and the interface tracking ALGORITHM 1.

1.2. Another simulation using the CFD grid $\mathcal{D}_h^{NBF}$, the embedded boundary method, and the interface tracking ALGORITHM 2.

1.3. A third simulation using the CFD grid $\mathcal{D}_h^{BF}$ and the ALE computational framework.

All three simulations outlined above are initialized with a uniform flow corresponding to the free-stream conditions specified above. The obtained time-histories of the lift are reported in Figure 10 for the first five periods of oscillation (0 sec $\leq t \leq 0.01$ sec). The lift predicted by simulation 1.3 can be considered as a reference lift for two reasons: (1) it is computed on a much finer CFD grid ($896,167$ grid points in $\mathcal{D}_h^{BF}$ versus $105,030$ in $\mathcal{D}_h^{NBF}$), and (2) as stated earlier, AERO-F's ALE computational framework has been successfully verified and validated for many aerodynamic and aeroelastic applications in the past. In Figure 10, the reader can observe that the lift predicted by simulation 1.2 is almost identical to the reference lift, whereas that predicted by simulation 1.1 is less accurate. This is not surprising given the inaccurate intersection results obtained by ALGORITHM 1 (see Figure 8).

Finally, Table I reports the CPU performance obtained on 32 processors for simulations 1.1 and 1.2. For this problem, ALGORITHM 2 for interface tracking is found to be three times slower than ALGORITHM 1. However, ALGORITHM 1 is not robust or accurate in this case. Furthermore, interface tracking with ALGORITHM 2 is also found to consume only 12% of the total CPU time. Indeed, the computational complexity of an implicit flow solver is in general of the order of $O(N \log N)$, and that of both interface tracking algorithms presented in this paper is of the order of $O(N \log N_E)$, where N and N_E denote the number of points in the CFD grid and number of elements in the embedded discrete interface. In most applications, N_E is much small than N because

Prepared using fldauth.cls
DOI: 10.1002/fld

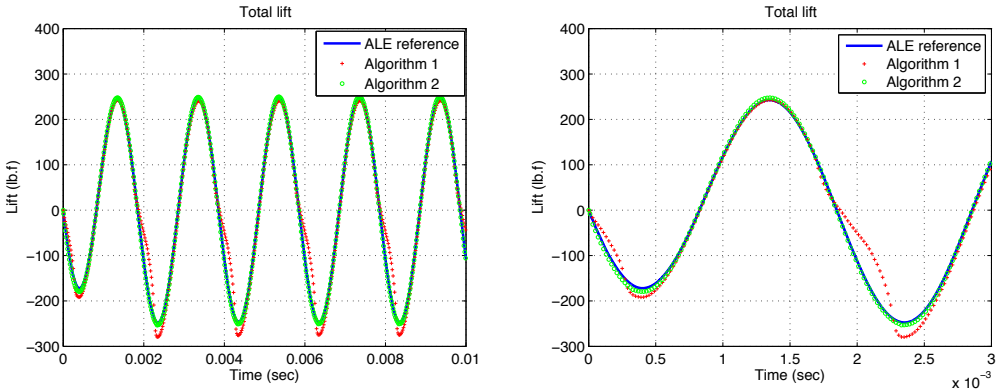

Figure 10. Thin wing in heaving motion: comparison of the lift time-histories predicted by all three performed numerical simulations.

Table I. Thin wing in heaving motion: CPU performance results on 32 processors.

	Simulation 1.1	Simulation 1.2
Flow computations	949 sec	1,019 sec
Finite volume fluxes and Jacobians	170 sec	198 sec
Linear system solution (GMRES)	629 sec	641 sec
Various other computations	150 sec	180 sec
Interface tracking	53 sec	154 sec
Total simulation time	1,042 sec	1,212 sec

$\mathcal{D}_h$ is a three-dimensional entity but $\mathcal{D}_h^E$ is a two-dimensional one. In this example, the ratio between N_E and N is roughly $e = \dfrac{N_E}{N} = 0.39$.

4.2. Validation for the implosive collapse of an air-filled cylindrical shell submerged in water

Next, the simulation of an underwater implosion experiment recently performed at the University of Texas at Austin is considered. This experiment features a transient high-speed multi fluid-structure interaction problem characterized by ultra-high compressions, strong shock waves, and large structural displacements and deformations. In this simulation, the embedded discrete interface is well resolved by the non body-fitted CFD grid from the beginning until the full collapse of

Prepared using fldauth.cls
DOI: 10.1002/fld

the structure. Hence, it can be considered as an illustration of Case I of Figure 5. In such a case, ALGORITHM 1 and ALGORITHM 2 for interface tracking can be expected to deliver similar results.

4.2.1. Experiment An air-filled aluminum cylinder (the specimen) of length $L_0 = 5.0014$ in, circular cross section with external diameter $D = 1.5007$ in, and thickness $t = 0.0280$ in is submerged in a rigid water tank. It has a maximum ovalization (imperfection) $\Delta_o = 0.083\%$. It is bonded at both ends to two rigid steel plugs which close the cylinder. The unbonded region of the cylinder has a length $L = 2D = 3.0014$ in (see Figure 11). The cylinder is maintained at the center of the tank by a set of bars attached to the rigid tank. It is surrounded by 6 pressure sensors that are distributed on the mid-plane (orthogonal to the axial direction of the cylinder) with radial distance $d = 2.5 \pm 0.25$ in to the center of the cylinder.

Initially, the water outside the cylinder and the air inside it are at rest ($v_w^0 = v_a^0 = 0$ in/sec). They have the same pressure $p_w^0 = p_a^0 = 14.5$ psi. The density of water is $\rho_w^0 = 9.357255 \times 10^{-5}$ (lb/in^4).sec^2. The density of air is $\rho_a^0 = 9.357255 \times 10^{-8}$ (lb/in^4).sec^2. Then, the water pressure is slowly increased at a constant rate until the cylinder collapses. The final hydrostatic pressure, under which the cylinder collapses, is $p_{co} = 690.5$ psi. Two photographs of the collapsed cylinder are shown in Figure 12. The time at which the cylinder starts to collapse is chosen to designate $t = 0$. The recorded pressure time-history by sensor 1 (see Figure 14) reveals first a gradual pressure drop of 118.0 psi before the cylinder gets into self-contact, followed by a sharp pressure rise with a peak of 1.16×10^3 psi afterward. This sharp pressure rise corresponds to the strong shock waves caused by self-contact of the structure.

4.2.2. Numerical simulations The geometric center of the cylinder is chosen as the origin of the Cartesian coordinate system, and its axial direction is chosen as the x-axis. In the structural dynamics sub-problem, half of the cylinder (length-wise) is modeled. Its aluminum material is represented as a nonlinear elasto-plastic medium with a Young modulus $E = 1.014 \times 10^7$ psi, Poisson ratio $\nu = 0.3$, and density $\rho^S = 2.599 \times 10^{-4}$ (lb/in^4).sec^2. The yield stress is set to 3.909×10^4 psi and the hardening modulus to 9.366×10^4 psi. The cylinder is discretized by a

Prepared using fldauth.cls
DOI: 10.1002/fld

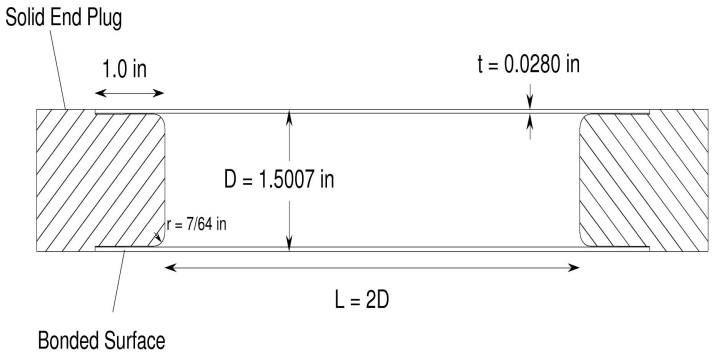

Figure 11. Schematic drawing of a cylindrical implodable with end caps designated by stripes (courtesy of Stelios Kyriakides).

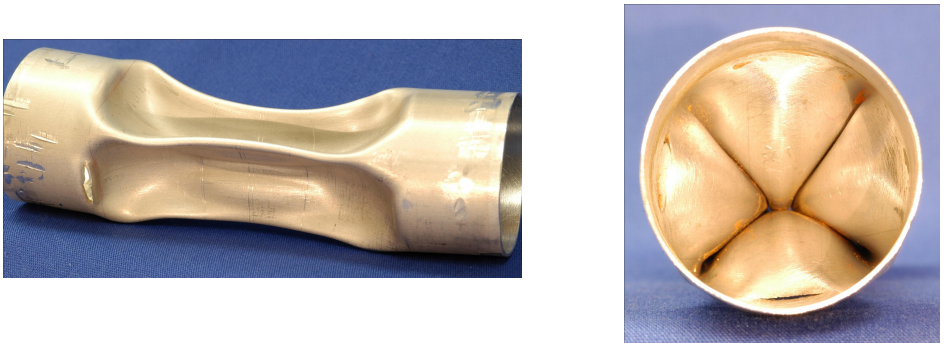

Figure 12. Photographs of the collapsed cylinder (courtesy of Stelios Kyriakides).

finite element model with $10,368$ four-noded shell elements. The steel plug to which the cylinder is bonded is represented by $1,833$ four-noded rigid shell elements. The density of these rigid elements are artificially set to $\rho^P = 4.7363 \times 10^{-2}$ (lb/in^4).sec^2 so that the total mass of these shell elements is equal to the total mass of the volumetric steel plug used in the experiment.

A Mode 4 sinusoidal imperfection is imposed on the geometry of the cylinder to trigger its collapse. More specifically, the circular cross section of the cylinder is replaced by $\{(r,\theta), 0 \leq$

Prepared using fldauth.cls
DOI: 10.1002/fld

$\theta < 2\pi\}$ with

$$r = r_0(1 - \Delta \cos 4\theta),$$

where $r_0 = 0.73635$ in is the radius of the true circular cross section measured to the mid-surface of the cylinder. The imperfection magnitude Δ is set to be equal to the maximum ovalization of the specimen in the experiment — that is, $\Delta = \Delta_o = 0.083\%$.

In the CFD sub-problem, air inside the cylinder is modeled as a perfect gas with a specific heat ratio $\gamma_a = 1.4$. Because of the ultrahigh compressions involved in this problem, water is modeled as stiffened gas whose equation of state can be written as

$$(\gamma - 1)\rho e = p + \gamma p_0,$$

where e denotes the internal energy per unit mass, and γ and p_0 are two constants set here to $\gamma_w = 4.4$ and $p_0 = 8.7 \times 10^7$ psi. The fluid computational domain is a rectangular box: $\Omega = \{(x, y, z) \in \mathcal{R}^3 : 0 \text{ in} \leq x \leq 12 \text{ in}, \ -10 \text{ in} \leq y \leq 10 \text{ in}, \ -10 \text{ in} \leq z \leq 10 \text{ in}\}$. An unstructured non body-fitted CFD grid $\mathcal{D}_h^{NBF}$ with $1,689,089$ grid-points and $10,064,277$ tetrahedra is generated to discretize Ω (Figure 13). A separate discrete representation of the surface of the cylinder $\mathcal{D}_h^E$ is constructed with $12,275$ grid points and $24,404$ triangles and immersed into this CFD grid.

Symmetry boundary conditions are applied to both the fluid and structure models at $x = 0$. Non-reflecting boundary conditions are applied to the remaining boundaries of the fluid domain. Sliding boundary conditions are applied to the plug as well as the bonded region of the cylinder so that only a free displacement along the x direction is allowed. At $t = 0$, the initial state of air inside the cylinder is set to ρ_a^0, v_a^0, and p_a^0. The initial state of water is set to ρ_w^0, v_w^0, and $p_a^* = p_{co} - 10$ psi. In the first 1.5×10^{-4} sec of the simulation, the water pressure is increased statically and linearly to p_{co}, whereas the air pressure inside the cylinder is maintained at p_a^0. At $t = 1.5 \times 10^{-4}$ sec where the collapse pressure p_{co} is reached, the compressible flow solver is activated.

Two numerical simulations are performed using AERO-F's embedded boundary method: one using ALGORITHM 1 for interface tracking (denoted here by simulation 2.1), and the other using ALGORITHM 2 for this purpose (denoted here by simulation 2.2). Both simulations are carried

Prepared using fldauth.cls
DOI: 10.1002/fld

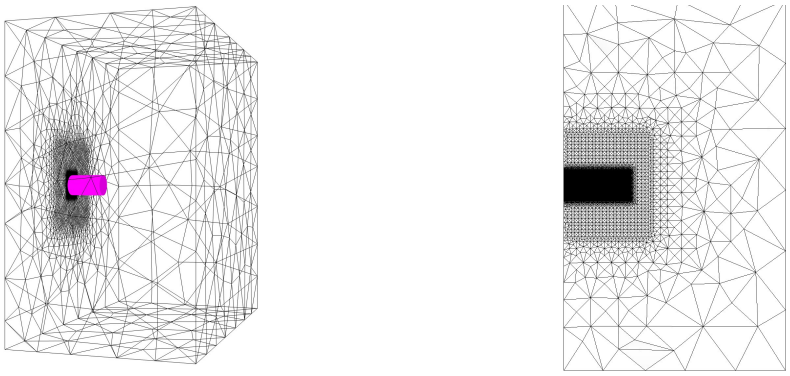

Figure 13. Underwater implosion problem: $\mathcal{D}_h^E$ (magenta color, Left) and a cut-view at $z = 0$ of $\mathcal{D}_h^{NBF}$ (Right).

out on 260 processors for AERO-F and a single processor for AERO-S. The water pressure time-histories are recorded at the sensor locations for the first $T = 1.5 \times 10^{-3}$ sec.

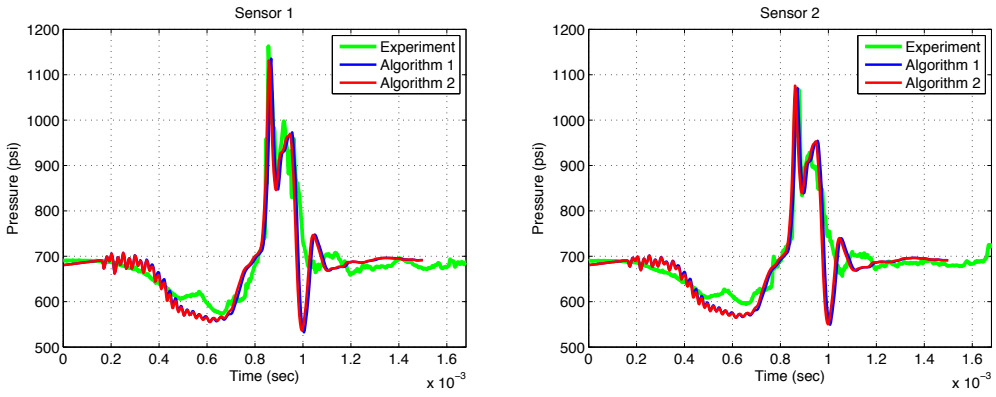

Figure 14. Underwater implosion problem: predicted and measured pressure time-histories (sensor 1, Left — sensor 2, Right).

The predicted pressure time-histories are reported at two different sensors labeled here as sensor 1 and sensor 2 in Figure 14, together with the corresponding experimental data. As expected, AERO-F's embedded boundary method equipped with either interface tracking algorithm presented in this paper delivers almost the same results. More importantly, these results reproduce the main features of the experimental data, including the highest pressure peak and the width of the pressure jump.

Prepared using fldauth.cls
DOI: 10.1002/fld

The structural deformation of the specimen predicted at $T = 1.5 \times 10^{-3}$ sec by simulation 2.1 is shown in Figure 15. That predicted by simulation 2.2 is almost identical. The reader can observe that this deformation approaches well that obtained experimentally (see Figure 12).

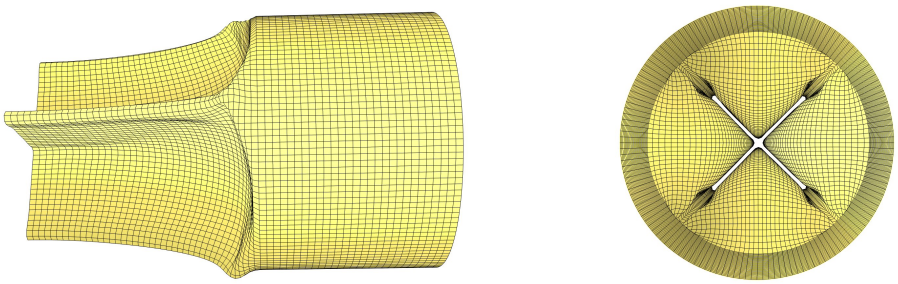

Figure 15. Underwater implosion problem: predicted deformation of the cylinder at $T = 1.5 \times 10^{-3}$ sec (simulation 2.1).

In this problem, the ratio between the number of elements N_E in the embedded discrete interface and the number of grid points N in the non body-fitted CFD is $e = \dfrac{N_E}{N} = 0.014$. The total CPU time elapsed in interface tracking is 566 sec in simulation 2.1 and $1,355$ sec in simulation 2.2. These timings account for only 1.9% and 4.1% of the total CPU time of simulation 2.1 and 2.2, respectively.

4.3. Application to the aeroelastic simulation of flapping wings

The aeroelastic simulation of a pair of ultra-thin flapping wings is considered here. The wings have a triangular shape. They are assumed to be made of a polyester film. For simplicity, the behavior of this film is assumed to be elastic and characterized by a Young modulus $E_p = 3.79 \times 10^9$ Pa, Poisson ratio $\nu_p = 0.35$, and density $\rho_p = 1.4 \times 10^{-3}$ g/mm^3. The wings have a uniform thickness of 0.16 mm only, except in some areas where structural reinforcement is implemented in the form of strips with a four times larger thickness equal to 0.64 mm (see Figure 16). Another structural reinforcement is made at the leading edges of the wings in the form of two stiff elastic fibers with the same circular cross section and the following material properties: $E_f = 5.55 \times 10^9$ Pa, $\nu_f = 0.34$,

Prepared using fldauth.cls
DOI: 10.1002/fld

and $\rho_f = 1.465 \times 10^{-3}$ g/mm^3. The cross sectional area of each fiber is $A_f = 0.916088$ mm^2 and its cross sectional moments of inertia about the local and centroidal x-, y-, and z-axis are $I_{xx} = 0.133566$ g.mm^2, and $I_{yy} = I_{zz} = 0.0667828$ g.mm^2, respectively. The wings are discretized by one layer of $1,152$ geometrically nonlinear three-noded shell elements with two different thicknesses as mentioned above. The fiber reinforcements are discretized by 48 two-noded geometrically nonlinear beam elements. Gravity is applied to this structural model with a uniform constant acceleration $\vec{g} = (0,\ 0,\ -9800$ mm/sec$^2)$. The displacement degrees of freedom at the two points located at the leading and trailing edges of the root chord (shared by both triangular wings) are fixed. The rotational degrees of freedom about the local y-axis at these two points are prescribed to the harmonic time-variation

$$\theta(t) = \theta_0 \sin(\omega t),$$

where $\theta_0 = 38^{\text{deg}}$ and $\omega = 14.5$Hz, which drives the flapping of both wings.

Because they are extremely thin, the wings considered here are extremely flexible. This makes the simulation of their structural dynamics particularly challenging as the finite element modeling of such thin structures is typically sensitive to numerical errors.

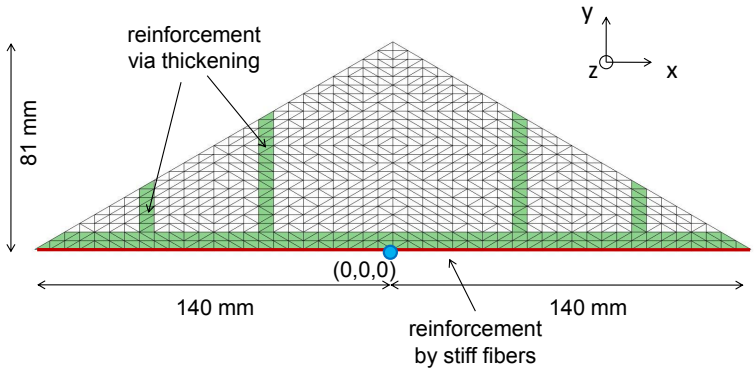

Figure 16. Ultra-thin flexible flapping wings: description and structural modeling.

The flow is assumed to be inviscid. From the physics viewpoint, this is a gross assumption. However, it has no effect on the illustration and evaluation of the interface *tracking* algorithms presented in this paper. To this effect, a non body-fitted CFD grid with $2,086,997$ grid points and

Prepared using fldauth.cls
DOI: 10.1002/fld

$12,462,983$ tetrahedra is generated (see Figure 17 and Figure 18). The $1,152$ surfacic triangles of the structural model described above are collected into a discrete fluid-structure interface that is embedded in this CFD grid. This embedded discrete interface is an open surface. Its dynamics can be tracked by ALGORITHM 2 but not by ALGORITHM 1. Hence, the simulation considered herein illustrates Case III of Figure 5.

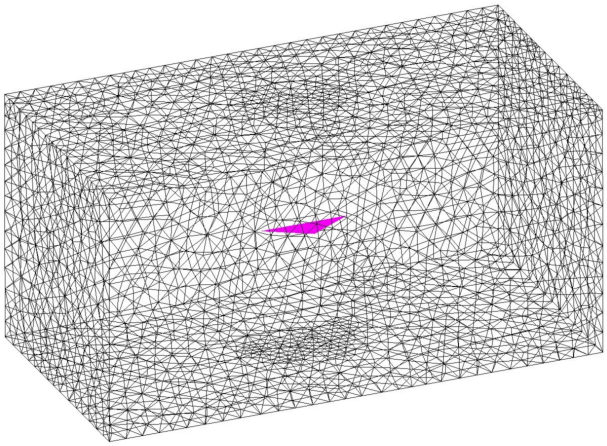

Figure 17. Ultra-thin flexible flapping wings: computational fluid domain for a dynamic aeroelastic simulation.

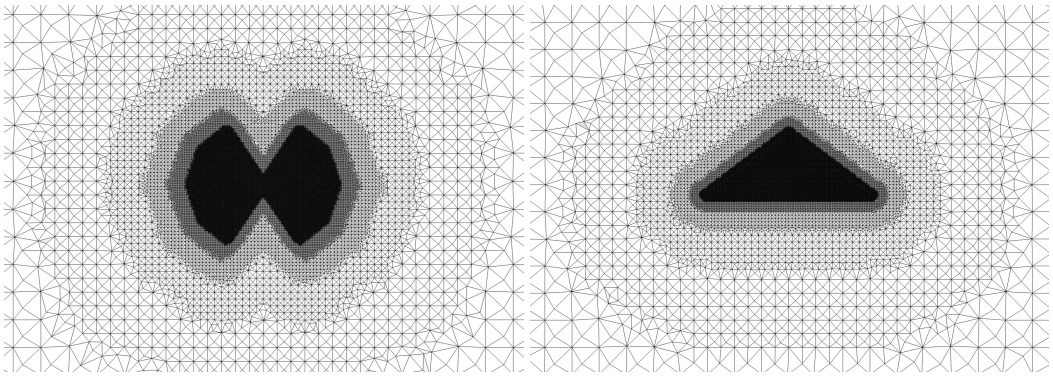

Figure 18. Ultra-thin flexible flapping wings: non body-fitted CFD grid — cutview at $y = 5$ mm (Left) and cutview at $z = 0$ mm (Right).

The aeroelastic simulation, referred to here as simulation 3.1, is performed using AERO-F's embedded boundary method equipped with ALGORITHM 2 for interface tracking and AERO-S. 260

Prepared using fldauth.cls
DOI: 10.1002/fld

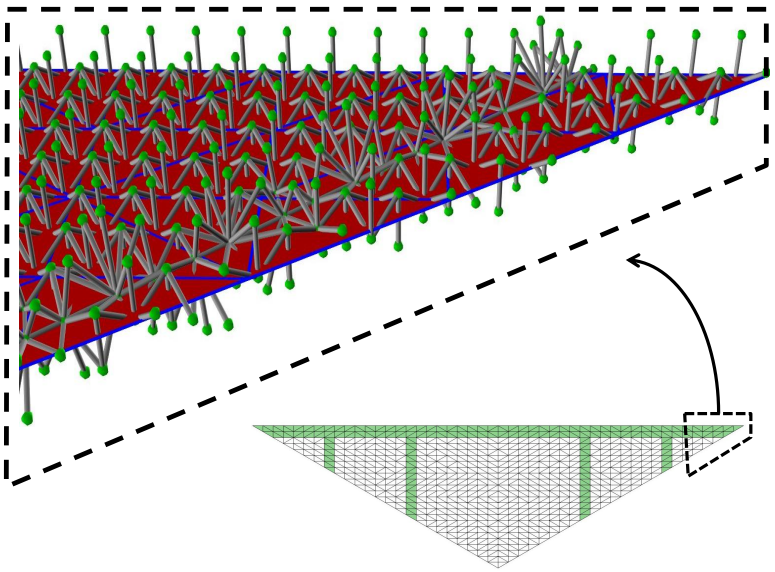

Figure 19. Ultra-thin flexible flapping wings: intersecting edges and node statuses identified by ALGORITHM 2.

processors are allocated for AERO-F and a single processor is allocated for AERO-S. At $t = 0$, the flow is initialized with the uniform state defined by $\rho = 1.3 \times 10^6$ g/mm^3, velocity $v = 0$ mm/sec, and pressure $p = 10^5$ g/(mm.s^2) $= 10^5$ Pa. Subsequently, the coupled aeroelastic responses of the flow and flapping wings are computed until $T = 0.3$ sec — which corresponds to roughly four flapping cycles.

Figure 19 displays the intersecting edges identified by ALGORITHM 2 near a wing tip at some time-instance t^n. These edges are shown as gray bars. The embedded discrete interface is shown in red. The status of the end-points of the intersecting edges is also color coded. The reader can observe that all end points of intersecting edges share the same green color, which indicates that these grid points are correctly identified as residing in the same flow medium (air).

The time-history of the displacement degree of freedom along the z-axis at one wing tip is reported in Figure 20. Its maximum amplitude is about 100 mm — that is, 71% of a wing semi-span. Color plots of the fluid pressure field and structural deformation at six different time-instances are shown in Figure 21. The large structural displacements and rotations and the high frequency content of the vibrations are evident.

Prepared using fldauth.cls
DOI: 10.1002/fld

In this coupled fluid-structure simulation, the cost of interface tracking is negligible. More specifically, it consumes only 108 seconds of the 5.9 hours of simulation — that is, 0.5% of the total simulation time. This is because in this case, $e = \dfrac{N_E}{N} = 5.5 \times 10^{-4}$.

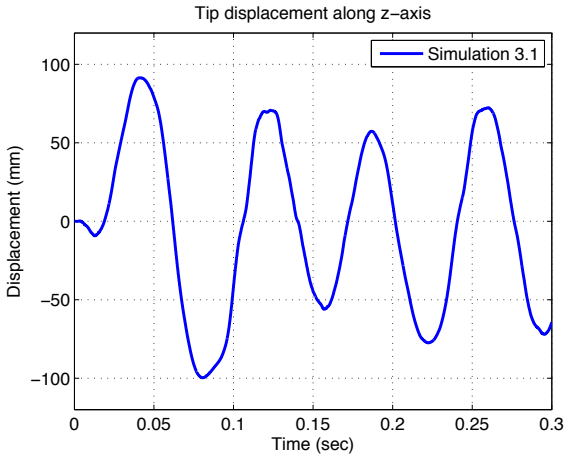

Figure 20. Ultra-thin flexible flapping wings: time-history of a tip displacement along the z-axis.

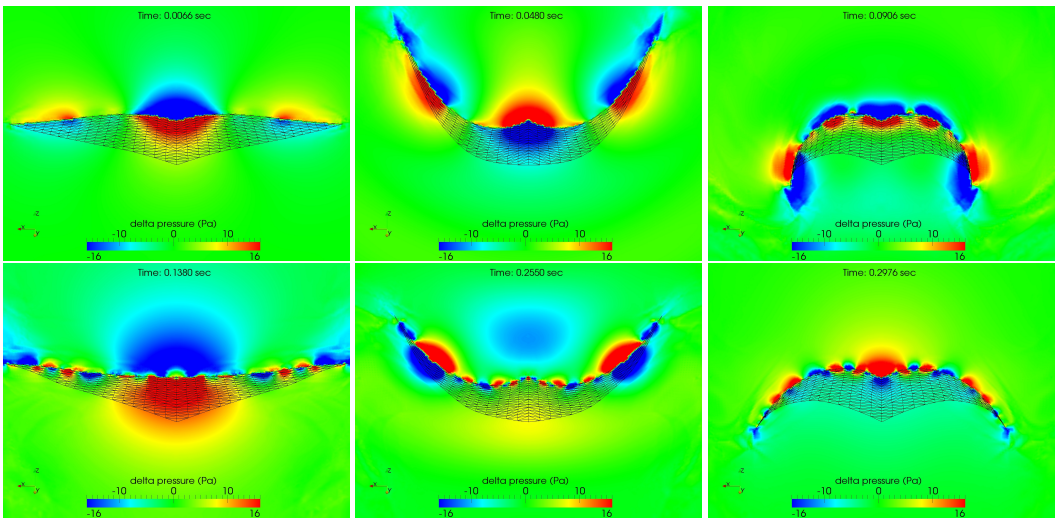

Figure 21. Ultra-thin flexible flapping wings: snapshots of the fluid pressure (cutview at $y = 20$ mm) and structural deformation at six different time-instances.

Prepared using fldauth.cls
DOI: 10.1002/fld

5. CONCLUSIONS

Interface tracking algorithms are critical components of embedded boundary computational frameworks for dynamic fluid-structure interaction problems with complex and deformable geometries. To this effect, two tracking algorithms capable of operating on structured as well as unstructured three-dimensional CFD grids have been presented in this paper. The first one is based on a projection approach, whereas the second one is based on a collision approach. The first algorithm is faster. However, it is restricted to closed interfaces and resolved enclosed volumes. The second algorithm is more versatile as it can handle open surfaces and underresolved enclosed volumes. Both computational algorithms have been equipped with a parallel distributed bounding box hierarchy to efficiently store and retrieve the elements of the discretized embedded interface. As a result, both interface tracking algorithms deliver a fast parallel performance. For several challenging three-dimensional, nonlinear, dynamic fluid-structure interaction problems arising from aeroelastic and underwater implosion applications, they were found to consume a small to tiny percentage of the total simulation CPU time.

ACKNOWLEDGMENTS

The authors acknowledge partial support by the Office of Naval Research under Grant N00014-06-1-0505 and Grant N00014-09-C-015, partial support by the Army Research Laboratory through the Army High Performance Computing Research Center under Cooperative Agreement W911NF-07-2-0027, and partial support by The Boeing Company under Contract Sponsor Ref 45047. The content of this publication does not necessarily reflect the position or policy of any of these supporters, and no official endorsement should be inferred. The authors also thank Dr. Michel Lesoinne for his contribution to the design of the projection-based interface tracking algorithm.

REFERENCES

Prepared using fldauth.cls
DOI: 10.1002/fld

1. Peskin CS. Flow patterns around heart valves: a numerical method. *Journal of Computational Physics* 1972; **10**:252-271.
2. Kreiss HO, Petersson A. A second-order accurate embedded boundary method for the wave equation with Dirichlet data. *SIAM Journal of Scientific Computation* 2006; **27**:1141–1167.
3. Glowinski R, Pan TW, Kearsley AJ, Periaux J. Numerical simulation and optimal shape for viscous flow by a fictitious domain method. *International Journal for Numerical Methods in Fluids* 2005; **20**:695–711.
4. Johansen H, Colella P. A Cartesian grid embedded boundary method for Poisson's equation on irregular domains. *Journal of Computational Physics* 1998; **147**:60–85.
5. Udaykumar H, Mittal R, Shyy W. Computation of solid-liquid phase fronts in the sharp interface limit on fixed grids. *Journal of Computational Physics* 1999; **153**:535–574.
6. Mittal R, Iaccarino G. Immersed boundary methods. *Annual Review of Fluid Mechanics* 2005; **37**:239–261.
7. Wang K, Rallu A, Gerbeau J-F, Farhat C. Algorithms for interface treatment and load computation in embedded boundary methods for fluid and fluid-structure interaction problems. *International Journal for Numerical Methods in Fluids*, DOI: 10.1002/fld.2556.
8. Farhat C, Rallu A, Shankaran S. "A higher-order generalized ghost fluid method for the poor for the three-dimensional two-phase flow computation of underwater implosions," *Journal of Computational Physics* 2008; **227**:7674–7700.
9. Udaykumar H, Mittal R, Rampunggoon P, Khanna A. A sharp interface Cartesian grid method for simulating flows with complex moving boundaries. *Journal of Computational Physics* 2001; **174**:345–380.
10. Gilmanov A, Sotiropoulos F. A hybrid Cartesian/immersed boundary method for simulating flows with 3D, geometrically complex, moving bodies. *Journal of Computational Physics* 2005; **207**:457–492.
11. Mittal R, Dong H, Bozkurttas M, Najjar FM, Vargas A, von Loebbecke A. A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries. *Journal of Computational Physics* 2008; **227**:4825–4852.
12. Guendelman E, Selle A, Losasso F, Fedkiw R. Coupling water and smoke to thin deformable and rigid shells. *SIGGRAPH 2005, ACM TOG 24* 2005; 973–981.
13. Deiterding R, Radovitzky R, Mauch S, Noels L, Commings J, Meiron D. A virtual test facility for the efficient simulation of solid material response under strong shock and detonation wave loading. *Engineering with Computers* 2006; **22**:325–347.
14. Cirak F, Deiterding R, Mauch S. Large-scale fluid-structure interaction simulation of viscoplastic and fracturing thin-shells subjected to shocks and detonations. *Computers and Structures* 2007; **85**:1049–1065.
15. Grétarsson J, Kwatra N, Fedkiw R. Numerically Stable Fluid-Structure Interactions Between Compressible Flow and Solid Structures. *Journal of Computational Physics* 2011; **230**:3062–3084.
16. Farhat C, Rallu A, Wang K, Belytschko T. Robust and provably second-order explicit-explicit and implicit-explicit staggered time-integrators for highly nonlinear fluid-structure interaction problems. *International Journal for Numerical Methods in Engineering* 2010; **84**:73–107.

Prepared using fldauth.cls
DOI: 10.1002/fld

17. Farhat C, Geuzaine P, Brown G. Application of a three-field nonlinear fluid-structure formulation to the prediction of the aeroelastic parameters of an F-16 fighter. *Computers & Fluids* 2003; **32**:3–29.
18. Geuzaine P, Brown G, Harris C, Farhat C. Aeroelastic dynamic analysis of a full F-16 configuration for various flight conditions. *AIAA Journal* 2003; **41**:363–371.
19. Farhat C, Lesoinne M, Maman N. Mixed explicit/implicit time integration of coupled aeroelastic problems: three-field formulation, geometric conservation and distributed solution. *International Journal for Numerical Methods in Fluids* 1995; **21**:807–835.
20. Löhner R. Applied computational fluid dynamics techniques: an introduction based on finite element methods (second ed.), John Wiley & Sons (2008) ISBN 978-0-470-51907-3.
21. Robinson-Mosher A, Shinar T, Gretarsson J, Su J, Fedkiw R. Two-way coupling of fluids to rigid and deformable solids and shells. *SIGGRAPH 2008, ACM TOG 27* 2008; 46.1–46.9.
22. Agarwal PK, de Berg M, Gudmundsson J, Hammar M, Haverkort HJ. Box-trees and R-trees with near-optimal query time. *Discrete & Computational Geometry* 2002; **28**:291–312.
23. Bridson R, Fedkiw R, Anderson J. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph.* 2002; **21**:594–603.
24. Piperno S, Farhat C, Larrouturou B. Partitioned procedures for the transient solution of coupled aeroelastic problems - Part I: model problem, theory, and two-dimensional application. *Computer Methods in Applied Mechanics and Engineering* 1995; **124**: 79–112.
25. Piperno S, Farhat C. Partitioned procedures for the transient solution of coupled aeroelastic problems - Part II: energy transfer analysis and three-dimensional applications. *Computer Methods in Applied Mechanics and Engineering* 2001; **190**: 3147–3170.
26. Farhat C, Lesoinne M. Two efficient staggered procedures for the serial and parallel solution of three-dimensional nonlinear transient aeroelastic problems. *Computer Methods in Applied Mechanics and Engineering* 2000; **182**: 499–516.
27. Farhat C, van der Zee KG, Geuzaine P. Provably second-order time-accurate loosely-coupled solution algorithms for transient nonlinear computational aeroelasticity. *Computer Methods in Applied Mechanics and Engineering* 2006; **195**:1973–2001.

Prepared using fldauth.cls
DOI: 10.1002/fld