



RESEARCH ARTICLE

Mesh adaptation framework for embedded boundary methods for computational fluid dynamics and fluid-structure interaction

Raunak Borker¹ | Daniel Huang² | Sebastian Grimberg¹  | Charbel Farhat^{1,2,3}  | Philip Avery⁴ | Jason Rabinovitch⁵

¹Department of Aeronautics and Astronautics, Stanford University, Stanford, California

²Institute for Computational and Mathematical Engineering, Stanford University, Stanford, California

³Department of Mechanical Engineering, Stanford University, Stanford, California

⁴US Army Research Laboratory, Adelphi, Maryland

⁵Entry, Descent, Landing and Formulation Group, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California

Correspondence

Raunak Borker, Department of Aeronautics and Astronautics, Stanford University, Durand Building, 496 Lomita Mall, Stanford, CA 94305-4035.
Email: raunakborker@gmail.com

Funding information

NASA Early Stage Innovations, Grant/Award Number: Grant NASA-NNX17AD02G; NASA Jet Propulsion Laboratory, Grant/Award Number: Contract JPL-RSA No. 1590208; Stanford Graduate Fellowship

Summary

Embedded Boundary Methods (EBMs) are often preferred for the solution of Fluid-Structure Interaction (FSI) problems because they are reliable for large structural motions/deformations and topological changes. For viscous flow problems, however, they do not track the boundary layers that form around embedded obstacles and therefore do not maintain them resolved. Hence, an Adaptive Mesh Refinement (AMR) framework for EBMs is proposed in this paper. It is based on computing the distance from an edge of the embedding computational fluid dynamics mesh to the nearest embedded discrete surface and on satisfying the y^+ requirements. It is also equipped with a Hessian-based criterion for resolving flow features such as shocks, vortices, and wakes and with load balancing for achieving parallel efficiency. It performs mesh refinement using a parallel version of the newest vertex bisection method to maintain mesh conformity. Hence, while it is sufficiently comprehensive to support many discretization methods, it is particularly attractive for vertex-centered finite volume schemes where dual cells tend to complicate the mesh adaptation process. Using the EBM known as FIVER, this AMR framework is verified for several academic FSI problems. Its potential for realistic FSI applications is also demonstrated with the simulation of a challenging supersonic parachute inflation dynamics problem.

KEYWORDS

boundary layer, embedded boundary methods, fluid-structure interaction, mesh adaptation

1 | INTRODUCTION

Viscous Fluid-Structure Interaction (FSI) problems arise in many engineering applications including, to name only a few, parachute inflation dynamics,¹⁻³ flapping wing flight,^{4,5} and aircraft aeroelasticity.⁶⁻⁸ Three computational frameworks have been developed so far for the numerical solution of such problems: The Lagrangian approach (for example, see the works of Maire et al⁹ and Georges et al¹⁰), the Arbitrary Lagrangian Eulerian (ALE) setting,¹¹⁻¹⁵ and the Eulerian computational framework.¹⁶⁻¹⁸ The Lagrangian approach is typically impractical for FSI problems characterized by high-speed flows. The ALE setting can be challenged by FSI problems that feature large deformations and large, nonrigid displacements and/or rotations. This is because ALE methods for FSI rely on numerical algorithms for distorting at each time

step the Computational Fluid Dynamics (CFD) mesh to keep it body fitted,¹⁹⁻²² and all such algorithms are vulnerable to mesh entanglement. The ALE setting is also unfeasible without remeshing for FSI problems characterized by topological changes such as those that may occur in aerodynamic topology optimization, and FSI problems with crack propagation.²³ On the other hand, the Eulerian framework, which uses a fixed, nonbody-fitted CFD mesh and relies on tools from computational geometry^{24,25} to track the fluid-structure interface, avoids all aforementioned difficulties and limitations. For this reason, Eulerian methods such as “immersed (or embedded) boundary,” “fictitious domain,” and “ghost fluid-structure” methods have gained prominence in the literature for highly nonlinear FSI problems. In this paper, all such methods are collectively referred to as Embedded Boundary Methods (EBMs).

Nevertheless, EBMs have their own shortcomings, primarily for viscous FSI problems. Indeed, being Eulerian in nature, EBMs do not track in principle a boundary layer at a fluid-structure interface. A naive remedy for this issue is to use a high-resolution CFD mesh in large parts of the computational fluid domain through, which embedded structures are expected to sweep. This approach ensures that the boundary layer is always resolved, no matter where a structure moves in these parts of the computational fluid domain or how it deforms. For most practical problems of interest, however, this fix makes the resulting mesh size unaffordable for very high Reynolds numbers. An alternative approach is to use an ALE formulation of EBMs such as that described in the work of Farhat and Lakshminarayan.²⁶ This approach is feasible and practical for a class of FSI problems with low to moderate Reynolds numbers. In the higher Reynolds number regime, Adaptive Mesh Refinement (AMR) becomes necessary for tracking a boundary layer that forms around a discretized structure embedded in a CFD mesh.

Most of the literature discussing AMR for EBMs focuses on the context of structured CFD meshes.²⁷⁻³⁰ In this context, AMR algorithms usually start with a regular Cartesian mesh and hierarchically split individual mesh cells into 2^d subcells, where $d = 2, 3$ denotes the dimension of the computational domain. Typically, these algorithms follow a patch-based refinement approach that generates nonconforming meshes. Such meshes can complicate the semidiscretization process, especially in the context of a *vertex*-based Finite Volume (FV) method. However, two AMR approaches have been recently presented in the works of Hachem et al³¹ and Abgrall et al³² for maintaining mesh conformity on unstructured embedding meshes. Both of them adapt an embedding mesh so that the error of the solution at the embedded interface is reduced. For this purpose, both of these AMR methods rely on an appropriate mesh metric and on local remeshing techniques, and both of them have been demonstrated for flows past rigid bodies only. Hence, the main objective of this paper is to present an AMR framework for EBMs that is applicable to highly nonlinear FSI applications, can operate on unstructured as well as structured meshes, and maintains in all cases mesh conformity. This framework, a preliminary version of which was recently overviewed in the conference paper,³³ allows EBMs to track the boundary layer, locate flow features such as shocks, vortices, and wakes and keep them at all time instances resolved. It is based on local coarsening and refinement techniques and therefore is amenable to parallel implementation. Furthermore, this framework is equipped with a load balancing approach that enables it to achieve parallel efficiency on parallel processors.

In general, an AMR scheme is organized around two steps: a first step, where an adaptation criterion is invoked to identify the edges or cells of the mesh to be coarsened and refined; and a second step, where these edges or cells are appropriately adapted. In the AMR framework proposed in this paper for EBMs, the first step is performed in two parts. In the first one, the time-dependent distance from an edge of the embedding CFD mesh to the nearest embedded discrete surface, whose shape may deform and evolve in time, is computed to infer, given a specified length scale, the length of this edge that allows resolving the boundary layer. In the second part of the first step, a Hessian-based error criterion is invoked to detect a flow feature and adapt the embedding mesh to resolve it. The second step is performed using the Newest Vertex Bisection (NVB) method.³⁴⁻⁴² Like any other cell bisection method, NVB bisects a selected mesh cell by halving only one of its edges known as the refinement edge. Using a judicious choice of the refinement edges, the shape regularity and cell conformity of the adapted mesh are guaranteed at all time instances. The entire AMR framework is incorporated in the AERO Suite^{6,7} for the simulation of highly nonlinear FSI problems. In particular, it is interfaced with the EBM of this software suite known as FIVER (Finite Volume method with Exact two-material Riemann problems)^{23,43-47} and demonstrated for several academic problems and one realistic FSI application.

The remainder of this paper is organized as follows. In Section 2, the scope and applicability of the proposed AMR framework for EBMs are discussed. Next, the mesh adaptation criteria for resolving a boundary layer and other flow features are described in detail in Section 3. Section 4 focuses on the mesh adaptation process based on the NVB method and on its parallel implementation. Section 5 covers other essential implementation details. Section 6 demonstrates the proposed AMR framework using four FSI problems. The first three of these are of the academic type: they serve the purpose of verifying the implementation of this framework in AERO Suite. The fourth considered FSI problem is a challenging, highly nonlinear FSI problem pertaining to the dynamics of the supersonic inflation of a parachute. Finally, conclusions are offered in Section 7.

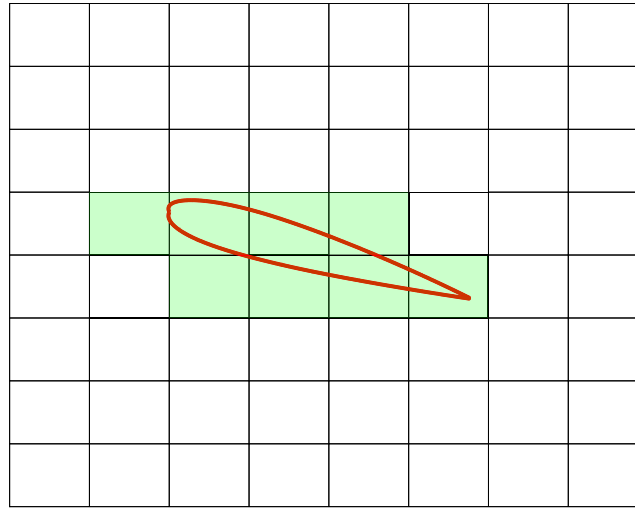


FIGURE 1 Embedding of an airfoil in a two-dimensional fluid mesh: Intersected cells are filled with green color to highlight the lack of relevance of both mesh directions to the flow direction [Colour figure can be viewed at wileyonlinelibrary.com]

2 | SCOPE

Usually, boundary layer meshing is discussed in the context of body-fitted meshes. For computational efficiency, such meshes are typically highly anisotropic in the vicinity of a wall boundary, where the flow has a gradient in the direction orthogonal to the wall that is much larger than that in the direction along this wall. On the other hand, EBM are designed on the premise that an embedding mesh is agnostic to the position and/or orientation of the embedded discrete body, which raises the issue of how to define anisotropy for embedding meshes. For body-fitted meshes, one can define for the cells in the vicinity of the wall boundaries a main direction along the body, and at least one direction orthogonal to it. For embedding meshes, however, one cannot always identify a direction that is particularly relevant to the flow. For example, Figure 1 shows a simple airfoil that is embedded in a two-dimensional fluid mesh. This figure illustrates the fact that the cells intersected by the airfoil (and colored in green) do not exhibit an orientation that is relevant to the flow past this airfoil. Hence, it is not obvious how to refine in this case any intersected cell only in the direction orthogonal to the flow along the body, unless remeshing is performed to modify at least the part of the embedding mesh that is located in the vicinity of the embedded airfoil. For this reason, it is postulated here that in the context of an EBM, the concepts of anisotropy and cell (element) aspect ratio are ill posed and therefore AMR is most practically performed in a manner that can be described as isotropic.

The mesh adaptation criteria adopted in this paper are edge-based: as such, they are mesh agnostic, which is they are applicable to structured, unstructured, and hybrid meshes composed of arbitrarily shaped cells with different numbers of vertices. Nevertheless, the adaptation process is limited to simplicial meshes. This restriction is due to the NVB method used for local coarsening and refinement of the embedding CFD mesh. It comes, however, with two major benefits: mesh conformity is always ensured, and the refinement process is completely reversible, which is the adapted CFD mesh can always be sufficiently coarsened to recover its previous configuration (and therefore, its initial configuration) without having to explicitly store the mesh refinement history.

Finally, the AMR framework for EBM presented in this paper is equally applicable to steady-state and unsteady CFD and FSI computations. For the sake of clarity and simplicity, it is described in the context of a single embedded discrete surface that may be rigid or flexible, static or dynamic, and denoted in all cases by $\Gamma_w(t)$, where t denotes time. The extension of this description to the case of multiple embedded discrete surfaces is straightforward.

3 | MESH ADAPTATION CRITERIA

In this work, an *edge*-based approach, as opposed to a *cell*-based one, is chosen for performing mesh adaptation. Hence, the adaptation criteria adopted here for tracking the boundary layer and other flow features and keeping them at all time instances resolved designate the edges of the embedding CFD mesh to be refined or coarsened. Specifically, the mesh

adaptation criterion chosen for tracking the boundary layer is based on a critical length scale; that chosen for locating other flow features is based on an approximation of the error associated with the prediction of a Quantity of Interest (QoI) using its Hessian (curvature).

Throughout this paper, the embedding CFD mesh is denoted by \mathcal{T} , a cell in this mesh is denoted by T , and the set of edges in this mesh is denoted by $\mathcal{E}(\mathcal{T})$. Furthermore, three subsets of $\mathcal{E}(\mathcal{T})$ are defined for the purpose of explaining the mesh adaptation process:

- $\mathcal{E}_r(\mathcal{T})$, the set of edges marked for refinement.
- $\mathcal{E}_c(\mathcal{T})$, the set of edges marked for coarsening.
- $\mathcal{E}_s(\mathcal{T})$, the set of edges that satisfy the adaptation criterion and therefore have the appropriate size.

3.1 | Tracking and efficiently resolving the boundary layer

The proposed mesh adaptation approach for tracking the boundary layer forming around a discrete surface embedded in a CFD mesh \mathcal{T} is based on the simple idea of partitioning \mathcal{T} edge wise into three *dynamic*, nonoverlapping zones, which is three nonoverlapping regions of the computational fluid domain whose boundaries evolve in time according to the dynamics of the embedded surface. At any time instance $t^{(n)}$ of an unsteady computation (or the final step of a pseudotime stepping procedure in the case of a steady-state CFD computation), these three zones are defined and characterized as follows (see Figure 2):

- An *inner zone* $Z_i^{(n)}$ delimited by: $\Gamma_w(t)$; and an intermediate surface $S_{it}(t)$ characterized by a time-independent, user-specified distance to the wall D_i . In this zone, the *minimum* edge length is greater than or equal to a time-dependent threshold $h_i^{(n)}/2$, the *maximum* edge length is smaller than or equal to the corresponding time-dependent threshold $h_i^{(n)}$, and $h_i^{(n)}$ is computed as explained below. The thickness of this zone, and thus, the parameter D_i is typically chosen to be a multiple (two to three times) of the initially estimated parameter h_i .
- An *outer zone* $Z_o^{(n)}$ delimited by: An intermediate surface $S_{to}(t)$ characterized by a time-independent, user-specified distance to the wall d_o ; and the outer boundary of the computational fluid domain. Here, the *minimum* edge length is greater than or equal to the time-independent threshold h_o defined as the *average* edge length of the initial outer zone $Z_o^{(0)}$. The parameter d_o is typically specified such that the outer zone begins a few layers of thickness h_o away from the body: It affects how gradually the mesh size varies away from its value near the embedded surface.

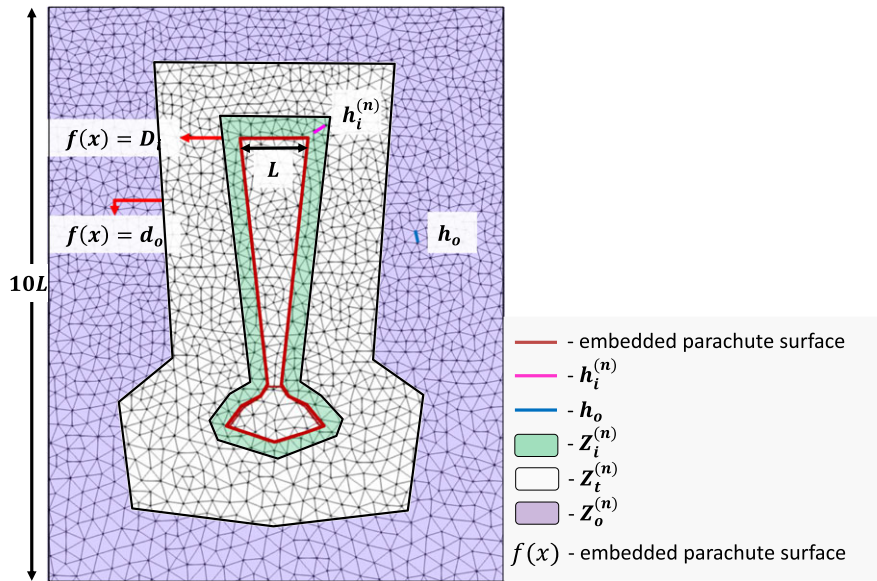


FIGURE 2 Embedding of a two-dimensional mockup model of a parachute in a two-dimensional fluid mesh: Definition and characterization of the three dynamic zones $Z_i^{(n)}$, $Z_o^{(n)}$ and $Z_t^{(n)}$ ($Z_o^{(n)} = (d_o, D_o^{(n)}, h_o)$, $Z_i^{(n)} = (0, D_i, h_i^{(n)})$) [Colour figure can be viewed at wileyonlinelibrary.com]

- A *transition zone* $Z_t^{(n)}$ delimited by: $S_{it}(t)$; and $S_{io}(t)$. In this zone, the *typical* edge length at time $t^{(n)}$, $h_t^{(n)}$, evolves progressively with the distance to $\Gamma_w(t)$ from $h_t^{(n)} = h_i^{(n)}$ to $h_t^{(n)} = h_o$.

The practical implementation of the notion of a geometrical progression of the typical edge length in $Z_t^{(n)}$ calls for subdividing this transition zone into subzones $Z_{t_j}^{(n)}$, $j = 1, \dots, m^{(n)}$, where the number of subzones $m^{(n)}$ at time $t^{(n)}$ depends on the chosen geometric progression rule, $h_i^{(n)}$, and h_o . Hence, $m^{(n)}$ is a time-dependent quantity. The objective of keeping the boundary layer at all times efficiently resolved calls for determining $h_i^{(n)}$ according to the *desired* value of the dimensionless distance to the wall of the first layer of vertices away from the wall boundary. This distance is denoted here by y_1^{+*} , where the subscript 1 designates the first layer of vertices away from $\Gamma_w(t)$, the superscript \star designates the desired value of y_1^+ at any vertex of the first layer of vertices away from $\Gamma_w(t)$, and the absence of the superscript (n) highlights the fact that the desired value of y_1^{+*} , y_1^{+*} , is time independent. This in turn calls for distinguishing between two cases: the case where a wall function is introduced for near-wall modeling and therefore $y_{1_{\min}}^{+*} \leq y_1^{+*} \leq y_{1_{\max}}^{+*}$, where $y_{1_{\min}}^{+*}$ and $y_{1_{\max}}^{+*}$ are time independent and user specified; and the case where the governing equations of the fluid subsystem are “integrated to the wall,” for which $y_1^{+*} = 1$. In either case, it follows that for unsteady computations, $h_i^{(n)}$ is as announced above a time-dependent quantity because y_1^{+*} depends on the time-dependent flow solution. Hence, unlike h_o , which is determined only once, at the beginning of the CFD or FSI simulation of interest, $h_i^{(n)}$ is computed at the beginning of each time step of such a simulation.

It follows that the proposed approach for performing AMR on an embedding CFD mesh \mathcal{T} to track a boundary layer and keep it at all times efficiently resolved relies on (see Figure 2):

- The computation of a function distance to the wall $f(X)$, where $X \in \mathbb{R}^3$.
- The exploitation of this function to define at any time instance (or pseudotime instance) $t^{(n)}$ of a CFD or FSI computation three evolving zones and $m^{(n)}$ evolving subzones. Each of these regions of \mathcal{T} can be represented by the triplet $(d^{(n)}, D^{(n)}, h^{(n)})$, where: $d^{(n)}$ and $D^{(n)}$ are the distances to $\Gamma_w(t)$ of the two delimiting boundary surfaces of a zone or subzone that are the closest to and farthest from the wall boundary at time $t^{(n)}$, respectively, which defines these two surfaces as the level sets $f(x) = d^{(n)}$ and $f(x) = D^{(n)}$; $h^{(n)}$ denotes the generic edge length of an element; $d_i^{(n)} = 0$; $D_i^{(n)} = D_i$ and $d_o^{(n)} = d_o$ are time independent and user specified; $D_o^{(n)}$ is time dependent and determined by the position of the outer boundary of the computational fluid domain relative to $\Gamma_w(t^{(n)})$; and $d_t^{(n)} = D_i$ and $D_t^{(n)} = d_o$ are time-independent quantities.
- The adaptation (refinement and/or coarsening) of \mathcal{T} in each zone or subzone so that the boundary layer is at all times efficiently resolved, which is guided for this purpose by: The value of y_1^{+*} in the inner zone; the *average* edge length h_o of the initial outer zone; and a geometric progression from $h_i^{(n)}$ to h_o in the transition zone. The latter geometric progression determines the number of transition subzones $m^{(n)}$, and for each transition subzone $Z_{t_j}^{(n)}$, $j = 1, \dots, m^{(n)}$, the distances $d_{t_j}^{(n)}$ and $D_{t_j}^{(n)}$ to $\Gamma_w(t)$, with $d_{t_1}^{(n)} = D_i$ and $D_{t_{m^{(n)}}}^{(n)} = d_o$.

Specifically, mesh adaptation is performed in *levels*, as follows. At each level, a bisection is used (as eluded to in Section 1 and discussed in detail in Section 4) to perform mesh refinement, and a single level of vertex deletion is carried out to perform mesh coarsening.⁴⁸ Therefore:

- At each mesh adaptation level, an edge length is halved, doubled, or untouched.
- The number of transition subzones (which is the same as the number of transition steps) needed at each mesh adaptation level to evolve $h_t^{(n)}$ in the transition zone $Z_t^{(n)}$ from $h_i^{(n)}$ to h_o is given by

$$m^{(n)} = \begin{cases} \text{ceil} \left[\log_2 \left(\frac{h_o}{h_i^{(n)}} \right) \right], & \text{if } \frac{h_o}{h_i^{(n)}} > 1 \\ 0, & \text{otherwise (no transition zone)} \end{cases}$$

- The representation $(d_{t_j}^{(n)}, D_{t_j}^{(n)}, h_{t_j}^{(n)})$ of each subzone $Z_{t_j}^{(n)}$ can be written as $(D_{t_{j-1}}^{(n)}, D_{t_j}^{(n)}, h_{t_j}^{(n)})$, $\forall j \in [1, m^{(n)}]$, where

$$D_{t_0}^{(n)} = D_i, \quad D_{t_j}^{(n)} = D_i r^{(n)j}, \quad h_{t_j}^{(n)} = h_o / 2^{m^{(n)}-j}, \quad \forall j \in [1, m^{(n)}]$$

$$\text{and } r^{(n)} = \left(\frac{d_o}{D_i} \right)^{\frac{1}{m^{(n)}}}.$$

The characteristic ratio $r^{(n)}$ of the geometric progression described above ensures a smooth gradation of $h_t^{(n)}$ from $h_i^{(n)}$ to h_0 .

Given an initial CFD mesh and the zone parameters computed based on the associated CFD or FSI solution, the proposed approach for mesh adaptation in the boundary layer described above is summarized in Algorithm 1 below and graphically depicted in Figure 3. Specifically, this figure illustrates AMR during a single time step, in the boundary layer around a two-dimensional mockup model of a parachute that is embedded in the CFD mesh. In this example: $d_o = 1.2L$, where L denotes the initial size of the canopy; $D_o^{(0)} = 4L$; $h_o = 0.12$; $D_i = h_o/5$; and three mesh adaptation levels are required to resolve the boundary layer.

Algorithm 1 Mesh adaptation using a distance criterion

```

1: input: CFD mesh  $\mathcal{T}$ , zone parameters  $(Z_i, Z_o, Z_t)$ 
2: output:  $\mathcal{E}_r(\mathcal{T})$ ,  $\mathcal{E}_c(\mathcal{T})$ ,  $\mathcal{E}_s(\mathcal{T})$ 
3: initialize:  $zoneToEdges = \{ \}$ , map from a zone to its edges
4: for  $e \in \mathcal{E}(\mathcal{T})$  do
5:    $[\delta_1, \delta_2] = \text{compute distance to } \Gamma_w \text{ of both vertices of } e$ 
6:    $\delta = \min[\delta_1, \delta_2]$ 
7:   if  $r > 1$  then
8:      $index = \text{ceil} \left[ \frac{\log(\delta/D_i)}{\log r} \right]$ 
9:   else
10:     $index = 0$ 
11:   end if
12:   if  $index < 1$  then
13:      $zoneToEdges(Z_i).append(e)$ 
14:   else if  $index > m$  then
15:      $zoneToEdges(Z_o).append(e)$ 
16:   else
17:      $zoneToEdges(Z_{t_{index}}).append(e)$ 
18:   end if
19: end for
20: for  $z$ : loop over all the zones do
21:    $h$ : edge length defining  $z$ 
22:   for  $e \in zoneToEdges(z)$  do
23:     if  $\text{length}(e) > h$  then
24:       add  $e$  to  $\mathcal{E}_r(\mathcal{T})$ 
25:     else if  $\text{length}(e) \leq h/2$  then
26:       add  $e$  to  $\mathcal{E}_c(\mathcal{T})$ 
27:     else
28:       add  $e$  to  $\mathcal{E}_s(\mathcal{T})$ 
29:     end if
30:   end for
31: end for

```

3.2 | Capturing and resolving the flow features

Besides the boundary layer, a flow feature such as a shock wave, vortex filament, or shear layer is also important in a (compressible) flow. Underresolving such a feature inevitably leads to a large approximation error

$$e_h^{(n)} = u^{(n)} - u_h^{(n)}, \quad (1)$$

where $u^{(n)}$ denotes the exact solution at the time instance $t^{(n)}$ of the fluid problem (CFD) or subproblem (FSI), and $u_h^{(n)}$ is a discrete approximation of this solution at the same time instance. This approximation error is not necessarily known *a*

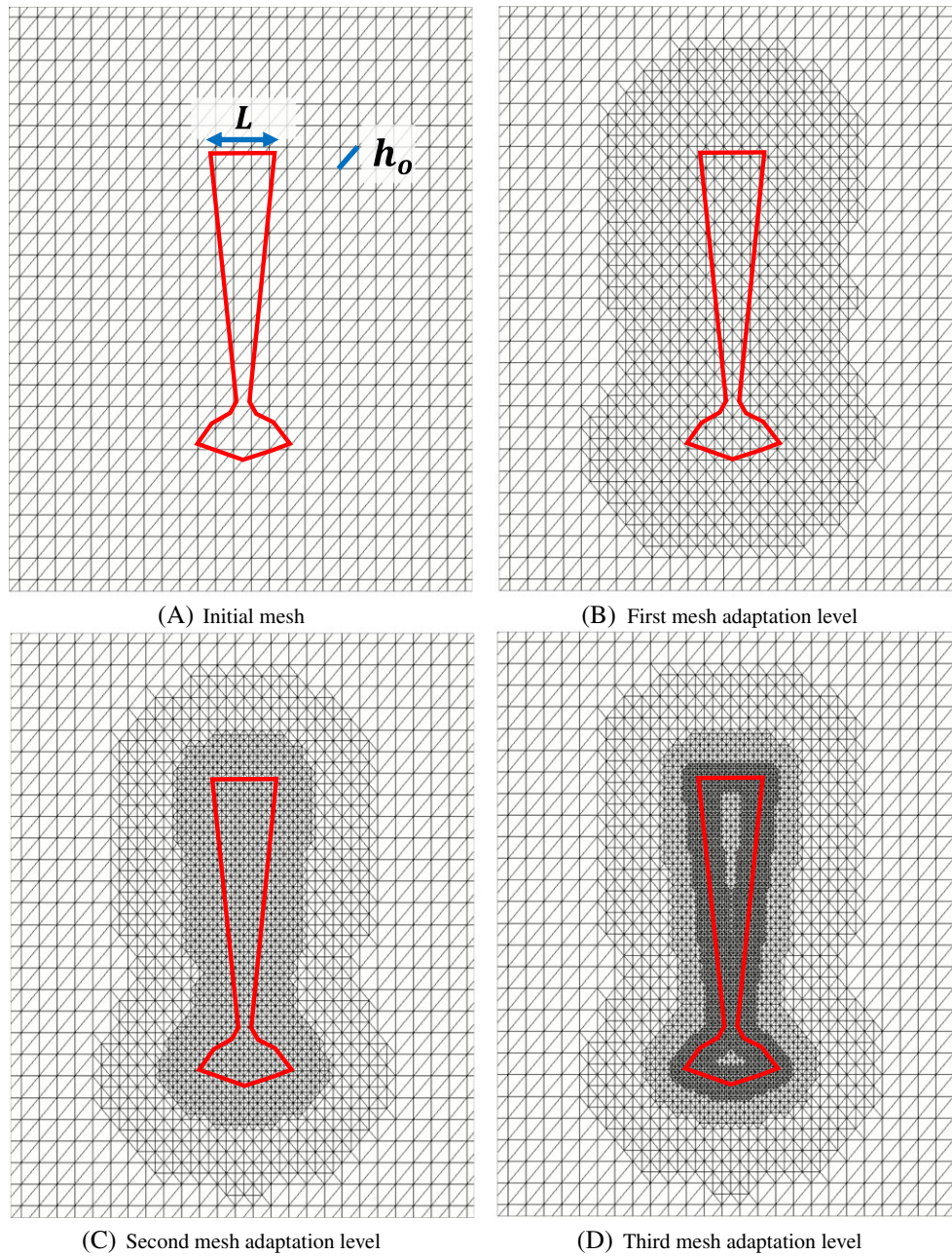


FIGURE 3 Application of Algorithm 4 to the fluid mesh of Figure 2 ($D_o^{(0)} = 4L$; $d_o = 1.2L$; $h_o = 0.12$; $D_i = h_o/5$) in which a mockup parachute model is embedded: Three mesh adaptation levels during the first computational time step [Colour figure can be viewed at wileyonlinelibrary.com]

priori. However, at any time instance $t^{(n)}$, it can be estimated *a posteriori* using the interpolation error

$$\mathcal{E}_h^{(n)} = u^{(n)} - \Pi_h u^{(n)}, \quad (2)$$

where Π_h is an interpolation operator relevant to the semidiscretization method underlying the CFD computation. Assuming that Cea's lemma⁴⁹ holds, the approximation error (1) can be bounded by the interpolation error as follows:

$$\|e_h^{(n)}\|_2 \leq C \|\mathcal{E}_h^{(n)}\|, \quad (3)$$

where C is a mesh-independent constant.

In the context of a vertex-based FV method, which is of particular interest to this work, the Hessian of the solution can be used to estimate the interpolation error as shown below (Equations (4-5)). Hence, it can be used to construct a mesh adaptation criterion. Intuitively, the Hessian conveys information about the curvature of an iso-contour of the solution or a related QoI. Since flow features typically exhibit large gradients and/or curvatures, a gradient- or Hessian-based error criterion can be expected to perform well at guiding AMR. For this reason, gradient- and Hessian-based error detectors have gained wide acceptance for AMR in the context of CFD (for example, see the works of Wong and Lele,⁵⁰ Löhner,⁵¹ and Peraire et al⁵²). For the sake of completeness, and to keep this paper as self-contained as possible, a simple error analysis is performed below to derive the role of the Hessian of a scalar QoI u_q in the construction of an edge-based error indicator for AMR. Similar and related derivations can also be found elsewhere, for example, in the works of Frey and Alauzet⁵³ and Ragusa.⁵⁴

To this end, $\mathcal{E}_h^{(n)}$ (2) is applied to the scalar QoI u_q using the standard linear interpolation operator Π_{P1} and expanded in a Taylor series about some point \mathbf{x}_0 to obtain

$$\mathcal{E}_h^{(n)}(\mathbf{x}) = \mathcal{E}_h^{(n)}(\mathbf{x}_0 + \Delta\mathbf{x}) = \underbrace{\left(u_q^{(n)}(\mathbf{x}_0) - \Pi_{P1}u_q^{(n)}(\mathbf{x}_0)\right)}_{\text{first term}} + \underbrace{\nabla \left(u_q^{(n)} - \Pi_{P1}u_q^{(n)}\right)(\mathbf{x}_0) \cdot \Delta\mathbf{x}}_{\text{second term}} + \underbrace{\int_0^1 \Delta\mathbf{x}^T H_{u_q}^{(n)}((1-s)\mathbf{x}_0 + s\mathbf{x}) \Delta\mathbf{x} ds}_{\text{third term}}, \quad (4)$$

where the superscript T designates the transpose operation, $H_{u_q}^{(n)}$ denotes the Hessian of $u_q^{(n)}$, Π_{P1} is constructed such that the first (constant) and second (gradient) terms of the above expansion vanish, and s is a scalar in $[0, 1]$. Note that for a linear interpolant, the Hessian of $\Pi_{P1}u_q^{(n)}$ vanishes and therefore does not appear in the third term of Equation (4). Hence, for any point located in space between \mathbf{x} and \mathbf{x}_0 and along the vector $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}_0$,

$$\max_{s \in [0,1]} \left| \mathcal{E}_h^{(n)}(s\mathbf{x} + (1-s)\mathbf{x}_0) \right| \leq \max_{s \in [0,1]} \left| \Delta\mathbf{x}^T H_{u_q}^{(n)}(s\mathbf{x} + (1-s)\mathbf{x}_0) \Delta\mathbf{x} \right|, \quad (5)$$

which shows that the maximum value of the Hessian of u_q along the vector $\mathbf{x} - \mathbf{x}_0$ bounds the interpolation error. Applying the above result to any edge \mathbf{e} connecting two vertices of \mathcal{T} denoted here by their positions in space \mathbf{x}_A and \mathbf{x}_B , it follows that

$$\max_{s \in [0,1]} \left| \mathcal{E}_h^{(n)}(s\mathbf{x}_A + (1-s)\mathbf{x}_B) \right| \leq \max_{s \in [0,1]} \left| \mathbf{e}^T \tilde{H}_{u_q}^{(n)}(s\mathbf{x}_A + (1-s)\mathbf{x}_B) \mathbf{e} \right|, \quad (6)$$

where the Hessian of the exact QoI u_q , which is typically unknown, has been replaced by an approximation denoted by $\tilde{H}_{u_q}^{(n)}$. Such an approximation can be evaluated, for example, by computing first the gradients of u_q at the vertices of \mathcal{T} , then approximating each gradient ∇u_q along an edge \mathbf{e} of \mathcal{T} by interpolating linearly its values at the vertices \mathbf{x}_A and \mathbf{x}_B . In this case, $\tilde{H}_{u_q}^{(n)}$ is constant along \mathbf{e} and therefore

$$\left| e_h^{(n)}(\mathbf{e}) \right| \leq C \left| \mathcal{E}_h^{(n)}(\mathbf{e}) \right| \leq \tilde{C} |\mathbf{e}^T \tilde{H}_{u_q}^{(n)} \mathbf{e}|,$$

where C and \tilde{C} are two mesh-independent constants. Then, given user specified, time-independent upper and lower thresholds for the error denoted here by e^{\max} and e^{\min} , respectively, the right-most term in inequality (6) can be used as an error criterion for the AMR procedure designed for capturing and resolving flow features and summarized in Algorithm 2.

Algorithm 2 Mesh adaptation using a Hessian-based criterion

- 1: **input:** CFD mesh \mathcal{T} , QoI u_q
 - 2: **output:** $\mathcal{E}_r(\mathcal{T})$, $\mathcal{E}_c(\mathcal{T})$, $\mathcal{E}_s(\mathcal{T})$
 - 3: estimate Hessian \tilde{H}_{u_q}
 - 4: **for** $\mathbf{e} \in \mathcal{E}(\mathcal{T})$ **do**
 - 5: $e_h = \mathbf{e}^T \tilde{H}_{u_q} \mathbf{e}$
 - 6: **if** $e_h > e_h^{\max}$ **then**
 - 7: add \mathbf{e} to $\mathcal{E}_r(\mathcal{T})$
 - 8: **else if** $e_h < e_h^{\min}$ **then**
 - 9: add \mathbf{e} to $\mathcal{E}_c(\mathcal{T})$
 - 10: **else**
 - 11: add \mathbf{e} to $\mathcal{E}_s(\mathcal{T})$
 - 12: **end if**
 - 13: **end for**
-

Note that in Algorithm 2, and for this matter, in any mesh adaptation algorithm described in the remainder of this paper, the coarsening threshold e^{\min} must be chosen to be not only different from e^{\max} , but several times smaller to minimize the possibility that the status of a mesh edge becomes stuck in an oscillatory state between being marked for refinement and marked for coarsening.

Note also that in practice, the QoI u_q can be chosen to be the pressure, density, velocity magnitude, vorticity magnitude, or any other scalar QoI associated with the solution of the fluid problem (CFD) or subproblem (FSI). In most cases, different reasonable QoIs yield similar results, but some flow features are captured only when specifying a certain QoI. For example, a contact discontinuity is captured by specifying the density as the QoI to be monitored (see the solution of the shock tube problem described in Section 6).

3.3 | Simultaneously resolving the boundary layer and the flow features

Mesh adaptation can be performed using simultaneously both criteria described in Section 3.3.1 and Section 3.3.2, by defining $\mathcal{E}_r(\mathcal{T})$ and $\mathcal{E}_c(\mathcal{T})$ appropriately. The case of mesh *refinement* is simple: If either criterion marks an edge for refinement, this edge is added to $\mathcal{E}_r(\mathcal{T})$. The case of mesh *coarsening* is treated as follows: If every criterion marks an edge for coarsening, this edge is added to $\mathcal{E}_c(\mathcal{T})$. The extended mesh adaptation procedure for an arbitrary number of adaptation criteria is laid out in Algorithm 3, where $\mathcal{E}_{i,r}(\mathcal{T})$ and $\mathcal{E}_{i,c}(\mathcal{T})$ denote the set of edges marked to be refined and coarsened by the i th mesh adaptation criterion, respectively, and $\mathcal{E}_{i,s}(\mathcal{T})$ denotes the set of edges marked as satisfying the requirement of the i th mesh adaptation criterion.

Algorithm 3 Mesh adaptation using multiple adaptation criteria

```

1: input: CFD mesh  $\mathcal{T}$ 
2: output:  $\mathcal{E}_r(\mathcal{T})$ ,  $\mathcal{E}_s(\mathcal{T})$ ,  $\mathcal{E}_c(\mathcal{T})$ 
3: for criterion  $i$  do
4:    $\mathcal{E}_r(\mathcal{T}) = \mathcal{E}_r(\mathcal{T}) \cup \mathcal{E}_{i,r}(\mathcal{T})$ 
5: end for
6: for criterion  $i$  do
7:   for  $e \in \mathcal{E}_{i,s}(\mathcal{T})$  do
8:     if  $e \notin \mathcal{E}_r(\mathcal{T})$  then
9:       add  $e$  to  $\mathcal{E}_s(\mathcal{T})$ 
10:    end if
11:  end for
12: end for
13: for criterion  $i$  do
14:   for  $e \in \mathcal{E}_{i,c}(\mathcal{T})$  do
15:     if  $e \notin \mathcal{E}_r(\mathcal{T})$  and  $e \notin \mathcal{E}_s(\mathcal{T})$  then
16:       add  $e$  to  $\mathcal{E}_c(\mathcal{T})$ 
17:     end if
18:   end for
19: end for

```

3.4 | Summary

The entire AMR process described in the previous sections is summarized in Algorithm 4, where:

- $\mathcal{T}^{(n)}$ denotes the mesh at the time instance $t^{(n)}$.
- $u^{(n)}$ denotes the solution vector associated with this mesh.
- Algorithm 6 is the algorithm adopted for refining a set of edges and is described in Section 4.1.
- Algorithm 8 is the algorithm adopted for coarsening a set of edges and is described in Section 4.2.

The generalization of this algorithm to more than two mesh adaptation criteria is straightforward.

Algorithm 4 Complete AMR algorithm using both distance (criterion 1) and Hessian-based (criterion 2) mesh adaptation criteria

```

1: initialize:  $D_i, d_o, h_o, y_1^{+*}, e_h^{\max}, e_h^{\min}, u^{(n)}$  (solution vector)
2: for time step:  $t^{(n)} \rightarrow t^{(n+1)}$  do
3:   initialize:  $\mathcal{E}_r(\mathcal{T}^{(n)}) = \emptyset, \mathcal{E}_c(\mathcal{T}^{(n)}) = \emptyset$ 
4:   begin compute the zone parameters
5:     update:  $D_o^{(n)}$ 
6:     compute: minimum  $h_i^{(n)}$  corresponding to  $y_1^{+*}$  using  $u^{(n)}$ 
7:     if  $\frac{h_o}{h_i^{(n)}} > 1$  then
8:        $m^{(n)} = \text{ceil} \left[ \log_2 \left( \frac{h_o}{h_i^{(n)}} \right) \right]$ 
9:     else
10:       $m^{(n)} = 0$ 
11:    end if
12:     $r^{(n)} = \left( \frac{d_o}{D_i} \right)^{\frac{1}{m^{(n)}}}$ 
13:     $Z_i^{(n)} = (0, D_i, h_i^{(n)}); \quad Z_o^{(n)} = (d_o, D_o^{(n)}, h_o); \quad D_{t_0}^{(n)} = D_i$ 
14:    for  $j = 1$  to  $m^{(n)}$  do
15:       $D_{t_j}^{(n)} = D_i r^{(n)^j};$ 
16:       $h_{t_j}^{(n)} = h_o / 2^{m^{(n)}-j};$ 
17:       $Z_{t_j}^{(n)} = (D_{t_{j-1}}^{(n)}, D_{t_j}^{(n)}, h_{t_j}^{(n)})$ 
18:    end for
19:  end
20:  do
21:    coarsen: using Algorithm 8 on  $\mathcal{E}_c(\mathcal{T}^{(n)}) \rightarrow \text{update } \mathcal{T}^{(n)}, \mathcal{E}_c(\mathcal{T}^{(n)}) = \emptyset$ 
22:    criterion 1: using Algorithm 1 on  $\mathcal{T}^{(n)}$  and the computed zone parameters  $\rightarrow \mathcal{E}_{1,r}(\mathcal{T}^{(n)}), \mathcal{E}_{1,c}(\mathcal{T}^{(n)}), \mathcal{E}_{1,s}(\mathcal{T}^{(n)})$ 
23:    criterion 2: using Algorithm 2 on  $\mathcal{T}^{(n)}$  and  $u^{(n)} \rightarrow \mathcal{E}_{2,r}(\mathcal{T}^{(n)}), \mathcal{E}_{2,c}(\mathcal{T}^{(n)}), \mathcal{E}_{2,s}(\mathcal{T}^{(n)})$ 
24:    resolve: using Algorithm 3 and both criteria  $\rightarrow \mathcal{E}_c(\mathcal{T}^{(n)})$ 
25:  while  $\mathcal{E}_c(\mathcal{T}^{(n)}) \neq \emptyset$ 
26:  do
27:    refine: using Algorithm 6 on  $\mathcal{E}_r(\mathcal{T}^{(n)}) \rightarrow \text{update } \mathcal{T}^{(n)}, \mathcal{E}_r(\mathcal{T}^{(n)}) = \emptyset$ 
28:    interpolate:  $u^{(n)} = \mathcal{I}(u^{(n)})$ , where  $\mathcal{I}(\cdot)$  interpolates  $u^{(n)}$  on the adapted mesh using its values on the mesh at the start of the adaptation level
29:    criterion 1: using Algorithm 1 on  $\mathcal{T}^{(n)}$  and the computed zone parameters  $\rightarrow \mathcal{E}_{1,r}(\mathcal{T}^{(n)}), \mathcal{E}_{1,c}(\mathcal{T}^{(n)}), \mathcal{E}_{1,s}(\mathcal{T}^{(n)})$ 
30:    criterion 2: using Algorithm 2 on  $\mathcal{T}^{(n)}$  and  $u^{(n)} \rightarrow \mathcal{E}_{2,r}(\mathcal{T}^{(n)}), \mathcal{E}_{2,c}(\mathcal{T}^{(n)}), \mathcal{E}_{2,s}(\mathcal{T}^{(n)})$ 
31:    resolve: using Algorithm 3 and both criteria  $\rightarrow \mathcal{E}_r(\mathcal{T}^{(n)})$ 
32:  while  $\mathcal{E}_r(\mathcal{T}^{(n)}) \neq \emptyset$ 
33:     $\mathcal{T}^{(n+1)} \leftarrow \mathcal{T}^{(n)}$ 
34: end for

```

4 | MESH ADAPTATION PROCESS

Here, the process of refining or coarsening the marked edges of the CFD mesh and some associated parallel implementation details are discussed. Edge refinement is implemented using the NVB method because it ensures the conformity of the adapted mesh. Edge coarsening is performed using an algorithm associated with the NVB method that does not require the explicit storage of the refinement history.

4.1 | Refinement algorithm and parallelization

Local mesh refinement typically generates nonconforming meshes with hanging vertices. Semidiscretizing the governing fluid equations in the presence of such vertices is straightforward in the case of a cell-centered FV method; however, it can

be challenging in the case of a vertex-based FV method or a Finite Element (FE) method. This issue is addressed here by performing edge refinement using the NVB method, which can be defined as a collection of local refinement algorithms for simplicial meshes. It was first proposed in the works of Mitchell³⁴ and Rivara³⁵ for two-dimensional triangular meshes (see Figure 6), then generalized in the works of Bänsch³⁶ and Kossaczky³⁷ to n -simplicial meshes in n -dimensions. The three-dimensional implementation of the NVB method adopted in this work is based on other works.^{38,39,48} In these references and the related literature, this AMR method is typically discussed in the context of element-based mesh refinement. However, this work focuses on edge-based refinement. For this reason, and because the modified NVB method presented in this work for edge-based refinement is built on top of its element-based counterpart, the original NVB method is first overviewed before its modified counterpart for edge-based refinement is described.

Consider a conforming mesh \mathcal{T} made of tetrahedral elements. Each tetrahedron $T \in \mathcal{T}$ can be defined as the convex hull of its set of vertices $\{\alpha_0, \alpha_1, \alpha_2, \alpha_3\}$, where each $\alpha_i \in \mathbb{R}^3$. The NVB method uses the concept of a *tagged* tetrahedron, which is a tetrahedron that is identified by an ordered sequence $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)_\gamma$ and a type $\gamma \in \{0, 1, 2\}$. The purpose of the type is to uniquely define the ordered sequence for elements resulting from the bisection of such a tagged tetrahedron. Because a tagged tetrahedron may have three different types and connects by definition 4 different vertices, there are $3 \times 4! = 72$ different tetrahedra that can be defined using the same vertices $\alpha_0, \alpha_1, \alpha_2$, and α_3 . (For further details on how to tag an initial mesh, see Appendix A). Given a tagged tetrahedron $T = (\alpha_0, \alpha_1, \alpha_2, \alpha_3)_\gamma$, the edge $R(T) := \overline{\alpha_0\alpha_3}$ is designated as the *refinement edge* of element T . Hence, when T is marked for refinement, $R(T)$ is bisected by introducing the new mesh vertex $\alpha_{03} = (\alpha_0 + \alpha_3)/2$. This produces two “children” elements $s_1(T)$ and $s_2(T)$, as illustrated in Figure 4. Specifically, both new elements $s_1(T)$ and $s_2(T)$ are defined as per the bisection rules given in Table 1, where each child’s type is set to

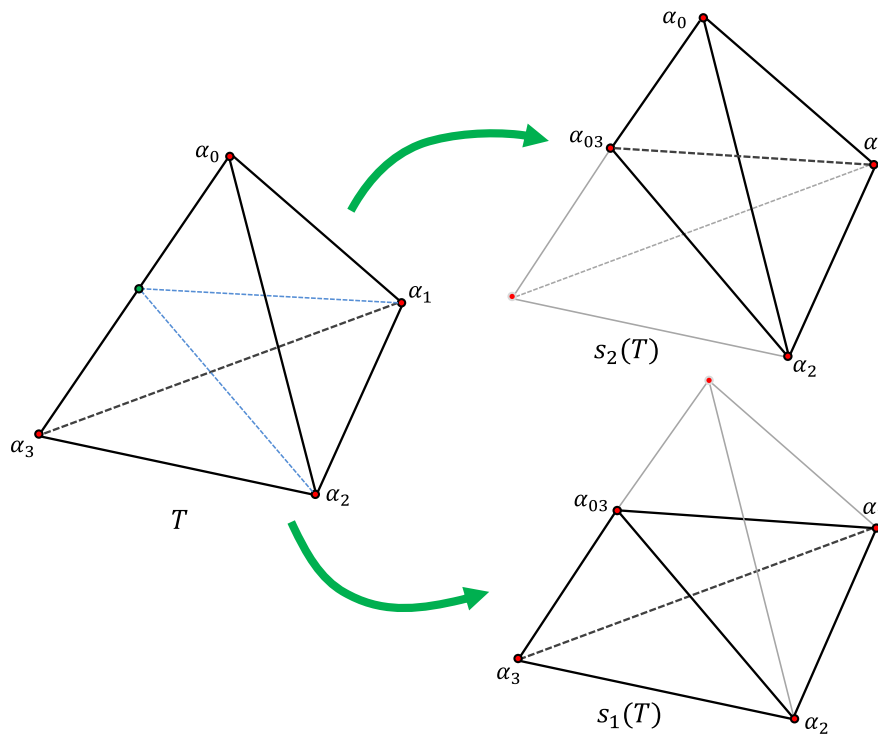


FIGURE 4 Bisection of the refinement edge $\overline{\alpha_0\alpha_3}$ of a tetrahedron and creation of two children tetrahedra [Colour figure can be viewed at wileyonlinelibrary.com]

TABLE 1 Bisection rules for a tetrahedron marked for refinement

γ	$s_1(T)$	$s_2(T)$
0	$(\alpha_0, \alpha_{03}, \alpha_1, \alpha_2)_1$	$(\alpha_3, \alpha_{03}, \alpha_2, \alpha_1)_1$
1	$(\alpha_0, \alpha_{03}, \alpha_1, \alpha_2)_2$	$(\alpha_3, \alpha_{03}, \alpha_1, \alpha_2)_2$
2	$(\alpha_0, \alpha_{03}, \alpha_1, \alpha_2)_0$	$(\alpha_3, \alpha_{03}, \alpha_1, \alpha_2)_0$

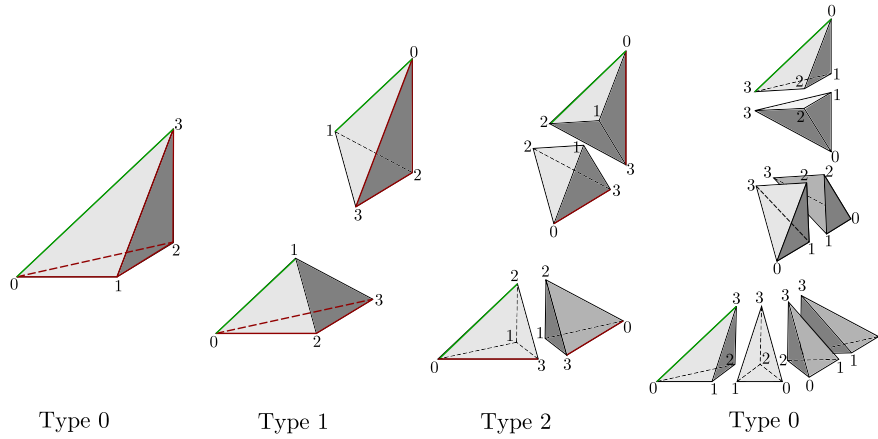


FIGURE 5 Illustration of the newest vertex bisection rules for a tetrahedron marked for refinement [Colour figure can be viewed at wileyonlinelibrary.com]

$((\gamma + 1) \bmod 3)$. By construction, the edge connecting the first and last vertices in the ordered sequence defining each of $s_1(T)$ and $s_2(T)$ defines the refinement edge of each of these two elements.

The aforementioned bisection rules and associated evolution of a tetrahedron type during the bisection process are illustrated in Figure 5. In this figure, the vertices of all tetrahedra are locally numbered to indicate their ordered sequence. The leftmost tetrahedron, assumed here to be the tetrahedron of interest (to be bisected), is assumed to be of type 0. Its refinement edge is shown in green color, and all of its other edges are shown in red color. Through the bisection process illustrated in this figure, the edges resulting from the bisection of the refinement edge are also shown in green color so that they can be visually tracked. Edges are only shown in red color if they also belong to the first (leftmost) tetrahedron. All newly added edges, as a result of bisection, are shown in black color. The reader can observe that the bisection rules are designed such that no newly added edge is bisected before each of the edges of the first tetrahedron (shown in red color) is bisected. In the rightmost set of tetrahedra, no red edge exists any longer: thus, all tetrahedra are tagged as type 0, and each green edge resulting from the bisection of the refinement edge of the original tetrahedron is now a refinement edge.

Note that if $R(T)$ lies on the boundary of the computational domain, its bisection does not introduce a nonconformity in the adapted mesh. On the other hand, if only T is marked for refinement and $R(T)$ is located within the interior of the computational domain, the bisection of $R(T)$ leads to a nonconforming adapted mesh. However, the NVB method avoids this mesh nonconformity by performing only *compatible* bisections, as described in the works of Stevenson⁴¹ and Bartels and Schreier⁴⁸ and summarized below.

The definition of a compatible bisection calls first for defining an *edge patch* $\mathcal{T}_{\mathbf{e}}$ of an edge $\mathbf{e} \in \mathcal{E}(\mathcal{T})$ as the set of elements that share the same edge \mathbf{e} , which is

$$\mathcal{T}_{\mathbf{e}} := \{T \in \mathcal{T} : \mathbf{e} \subset T\}. \quad (7)$$

Then, when all elements in $\mathcal{T}_{\mathbf{e}}$ have \mathbf{e} as their refinement edge, the bisection of \mathbf{e} is defined as a compatible bisection in definition 2.1 in the work of Bartels and Schreier.⁴⁸ Indeed, the bisection of \mathbf{e} does not introduce in this case any mesh nonconformity. The elements in the edge patch $\mathcal{T}_{\mathbf{e}}$ are said to be *compatibly divisible* with each other. The description of the original NVB method calls also for the definition of the following sets of elements:

- The set $\mathcal{E}_b(\mathcal{T})$ containing all refinement edges residing on the boundary of \mathcal{T} and marked for refinement. While this set is not particularly significant to the description of the serial implementation of the NVB method, it is necessary for the parallel implementation of this method described in Algorithm 7.
- For any $T \in \mathcal{T}$, the set $N(\mathcal{T}, T)$ consisting of all elements $T' \in \mathcal{T}$ that share a face with T and contain the refinement edge $R(T)$.

Then, the NVB method presented works of Stevenson⁴¹ and Bartels and Schreier⁴⁸ for element-based refinement can be summarized by Algorithm 5.

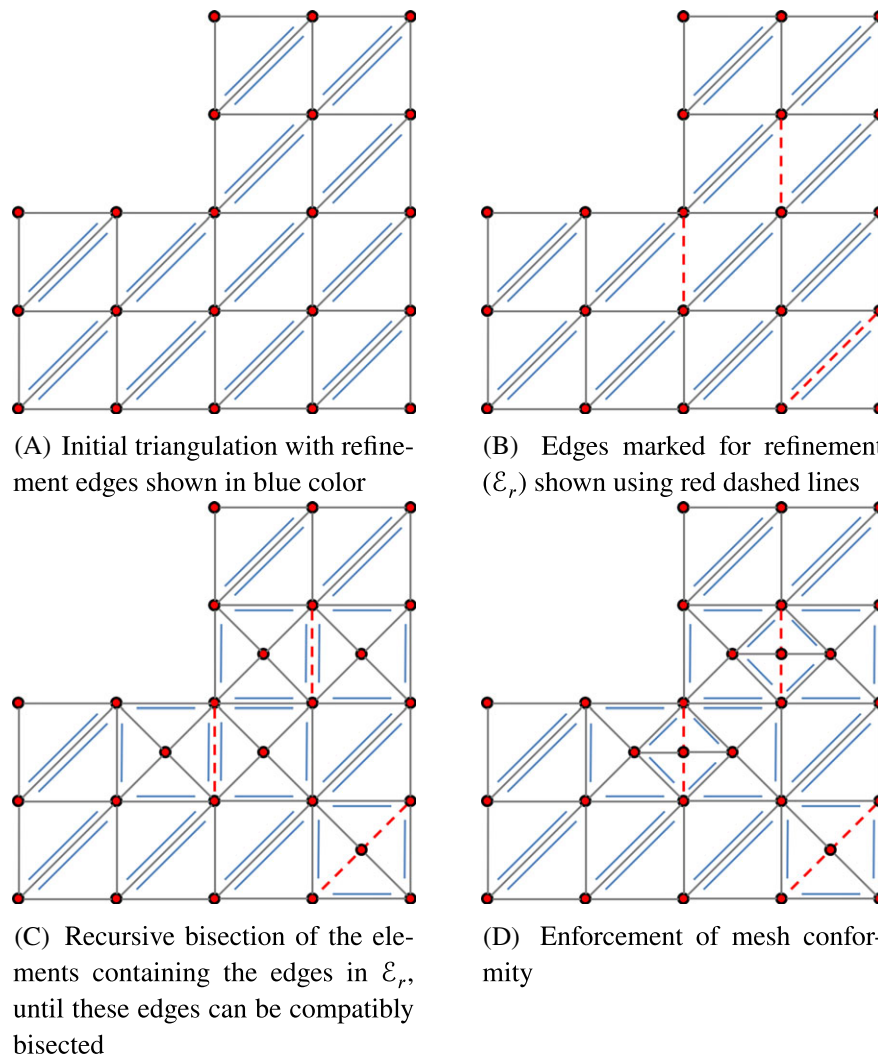


FIGURE 6 Newest vertex bisection method for edge-based refinement. [Colour figure can be viewed at wileyonlinelibrary.com]

Algorithm 5 was analyzed in the works of Kossaczky³⁷ and Stevenson,⁴¹ where it was shown that under a mild condition on the ordering and tagging of the initial mesh (see Appendix A), the recursion delivers the minimum conforming refinement \mathcal{T}' of \mathcal{T} (in the sense of the minimum number of added children elements) and preserves mesh quality.

The modified NVB method suitable for edge-based mesh refinement is summarized in Algorithm 6 and graphically depicted in Figure 6. For any edge \mathbf{e} marked for refinement, it refines the elements in $\mathcal{T}_{\mathbf{e}}$ until some element T' whose refinement edge is \mathbf{e} is encountered, which is guaranteed by the construction of Algorithm 5. At this point, T' is marked for refinement and the element-based NVB method summarized in Algorithm 5 is applied to T' . It follows that the edge-based NVB method outlined in Algorithm 6 is built on top of its element-based counterpart by adding an outer loop to Algorithm 5. Hence, the theoretical results stated in Bartels and Schreier⁴⁸ for Algorithm 5 hold also for Algorithm 6.

The parallel version of Algorithm 6 operates at the subdomain level. It calls for partitioning \mathcal{T} into subdomains \mathcal{T}^i , assigning a processor to each subdomain (or a collection of them), and applying Algorithm 6 locally to the edges of each subdomain. Note that bisecting an edge marked for refinement might necessitate refining many other edges that were not originally marked for refinement. This can be thought of as a propagation of the mesh refinement. In a distributed setting, such a propagation may reach the boundary of a subdomain, in which case it would require a communication between the processors assigned to neighboring subdomains so that the propagation can proceed as needed. In Algorithm 7, this is achieved via the communication and accumulation of the set $\mathcal{E}_b(\mathcal{T}^i)$ representing the set of refinement edges on the boundary of the subdomain \mathcal{T}^i .

Algorithm 5 NVB method for *element*-based mesh refinement

```

1: function REFINE( $\mathcal{T}, T$ )
2:   input: CFD mesh  $\mathcal{T}$ , element  $T$  marked for refinement
3:    $K = \emptyset; F = \{T\}$ 
4:    $\mathcal{E}_b(\mathcal{T}) = \emptyset$ 
5:   do
6:      $F_{\text{new}} = \emptyset$ 
7:     for  $T' \in F$  do
8:       for  $T'' \in N(\mathcal{T}, T')$  with  $T'' \notin F \cup K$  do
9:         if  $T''$  is compatibly divisible with  $T'$  then
10:           $F_{\text{new}} = F_{\text{new}} \cup \{T''\}$ 
11:        else
12:           $[\mathcal{T}, \mathcal{E}] = \text{REFINE}(\mathcal{T}, T'')$ 
13:          add to  $F_{\text{new}}$  the child of  $T''$  that shares with  $T'$  a face
14:           $\mathcal{E}_b(\mathcal{T}) = \mathcal{E}_b(\mathcal{T}) \cup \mathcal{E}$ 
15:        end if
16:      end for
17:    end for
18:     $K = K \cup F$ 
19:     $F = F_{\text{new}}$ 
20:  while  $F \neq \emptyset$ 
21:  create  $\mathcal{T}'$  from  $\mathcal{T}$  by simultaneously bisecting all  $T' \in K$ ; number the new vertices in chronological order; replace
   $T$  in the list of elements with  $s_1(T)$ , and append  $s_2(T)$  to the end of the list
22:  add to  $\mathcal{E}_b(\mathcal{T})$   $R(T') \forall T' \in K$  such that  $R(T')$  lies on the boundary of  $\mathcal{T}$ 
23:  return  $[\mathcal{T}', \mathcal{E}_b(\mathcal{T})]$ 
24: end function

```

Algorithm 6 NVB method for *edge*-based mesh refinement

```

1: function REFINE_EDGE( $\mathcal{T}, e$ )
2:   input: CFD mesh  $\mathcal{T}$ , edge  $e$  marked for refinement
3:    $\mathcal{E}_R := \{R(T) : \forall T \in \mathcal{T}_e\}$ 
4:    $\mathcal{E}_b(\mathcal{T}) = \emptyset$ 
5:    $T_{\text{tagged}} = \emptyset$ 
6:   if  $e \notin \mathcal{E}_R$  then
7:     for  $T \in \mathcal{T}_e$  do
8:        $[\mathcal{T}, \mathcal{E}] = \text{REFINE}(\mathcal{T}, T)$ 
9:       append to  $\mathcal{T}_e$  the child  $T'$  of  $T$  that has  $e$  as an edge
10:      remove  $T$  from  $\mathcal{T}_e$ 
11:       $\mathcal{E}_b(\mathcal{T}) = \mathcal{E}_b(\mathcal{T}) \cup \mathcal{E}$ 
12:      if  $R(T') = e$  then
13:         $T_{\text{tagged}} = T'$ 
14:        Break
15:      end if
16:    end for
17:  else
18:     $T_{\text{tagged}}$  is any  $T$  such that  $R(T) = e$ ,  $T_{\text{tagged}} = R^{-1}(e)$ 
19:  end if
20:   $[\mathcal{T}', \mathcal{E}] = \text{REFINE}(\mathcal{T}, T_{\text{tagged}})$ 
21:   $\mathcal{E}_b(\mathcal{T}) = \mathcal{E}_b(\mathcal{T}) \cup \mathcal{E}$ 
22:  return  $[\mathcal{T}', \mathcal{E}_b(\mathcal{T})]$ 
23: end function

```

Algorithm 7 Distributed NVB method for edge-based mesh refinement (to be performed by the processor assigned to subdomain i)

- 1: **communicate**: with processors assigned to neighboring subdomains in order to ensure that any shared edge marked for refinement is present in the refinement set of every subdomain containing this edge \rightarrow **update** $\mathcal{E}_r(\mathcal{T}^i) = \mathcal{E}_r(\mathcal{T}^i) \cup \{\mathcal{E}_r(\mathcal{T}^j) \cap \{\mathcal{E}(\mathcal{T}^i) \cap \mathcal{E}(\mathcal{T}^j)\}\} \forall$ neighboring subdomain j
 - 2: **do**
 - 3: $\mathcal{E}_b(\mathcal{T}^i) = \emptyset$
 - 4: **for** $\mathbf{e} \in \mathcal{E}_r(\mathcal{T}^i)$ **do** $[\mathcal{T}^i, \mathcal{E}] = \text{REFINE_EDGE}(\mathcal{T}^i, \mathbf{e})$
 - 5: $\mathcal{E}_b(\mathcal{T}^i) = \mathcal{E}_b(\mathcal{T}^i) \cup \mathcal{E}$
 - 6: **end for**
 - 7: **communicate**: status of the shared edges marked for refinement \rightarrow **update** $\mathcal{E}_r(\mathcal{T}^i)$ (ie repeat line 1)
 - 8: **while** $\mathcal{E}_r(\mathcal{T}^i) \neq \emptyset \forall i$
 - 9: assign to each vertex and each element of the CFD mesh a unique global number to maintain order in each subdomain, using Kahn's topological sort algorithm⁵⁵
-

For any element $T' \in \mathcal{T}'$ generated by bisecting $R(T)$ in an initial mesh \mathcal{T} , the *generation* of T' is defined as the number of bisections needed to create T' from T . In theorem 4.6 in the work of Alkämper and Klöforn,⁵⁶ it is shown that Algorithm 7 terminates in a finite number of iterations bounded by the maximum difference between the generation of any element marked for refinement (as a result of the marking of its refinement edge) and the generation of any other element in the mesh.

4.2 | Coarsening algorithm and parallelization

The NVB method described in Section 4.1 allows complete reversal of the refinement process to obtain the initial mesh without *explicitly* storing its history.^{48,57} Nevertheless, this history is stored in this work *implicitly* by adopting a chronological vertex ordering to allow mesh coarsening, when needed. Furthermore, this numbering is chosen to be a *global* numbering, to facilitate load balancing on parallel processors.

In NVB parlance, a mesh vertex is called the *newest vertex* if its advent occurs after that of its neighbors in the NVB procedure. The NVB method guarantees that the newest vertex can be removed during mesh adaptation without introducing any mesh nonconformity. The chronological ordering is encoded in the vertex numbering, and therefore the newest vertex has always the highest vertex number among its neighbors. Specifically, the serial form of the algorithm adopted here for coarsening the mesh elements associated with the set of edges marked for coarsening, \mathcal{E}_c , is laid out in Algorithm 8. It is based on counterpart algorithms presented in the works of Bartels and Schreier⁴⁸ and Chen and Zhang.⁵⁷

Algorithm 8 Local coarsening algorithm

- 1: build the vertex set, $\mathcal{N}(\mathcal{T}) := \{\text{vertex } n \text{ such that all edges connected to this vertex are in } \mathcal{E}_c\}$
 - 2: sort the vertices in $\mathcal{N}(\mathcal{T})$ in descending order of their vertex number
 - 3: **for** $n \in \mathcal{N}(\mathcal{T})$ **do**
 - 4: **if** n is the newest vertex **then**
 - 5: remove the vertex n
 - 6: replace the pair of children elements $s_1(T)$ and $s_2(T)$ with their parent T , and define T by its ordered sequence and type based on Table 1
 - 7: remove all edges connected to n , and add the refinement edge of parent T (which contains n by definition)
 - 8: **end if**
 - 9: **end for**
 - 10: compress the gaps in the global numbering of the vertex, edge, and element sets formed as a result of deletions
-

In Algorithm 8, looping over the vertices in $\mathcal{N}(\mathcal{T})$ in descending order guarantees the completion of the coarsening process in a single iteration. Otherwise, some vertices that are not intended to be deleted as part of the coarsening would become newest vertices at some intermediate steps, which would necessitate multiple iterations of this algorithm to complete the coarsening process. Therefore, it is worth mentioning that the information pertaining to the pair of children

elements $s_1(T)$ and $s_2(T)$ on line 6 is encoded in the element numbering. During refinement of element T , the list of elements is modified by overwriting T with $s_1(T)$ and appending $s_2(T)$ to the end of the list.

The primary challenge with the parallelization of Algorithm 8 at the subdomain level is associated with deleting shared vertices. For any vertex shared by multiple subdomains and belonging to the sorted vertex set \mathcal{N} , it needs to be ensured that such a vertex is the newest vertex for each of the subdomains before it is deleted. This is accomplished here through appropriate communication as described below in Algorithm 9.

Algorithm 9 Distributed local coarsening algorithm (to be performed by the processor assigned to subdomain i)

- 1: **communicate**: to build the vertex set $\mathcal{N}(\mathcal{T}^i) = \{\text{vertex } n \text{ such that all edges connected to this vertex are in } \mathcal{E}_c\}$
 - 2: sort the vertices in $\mathcal{N}(\mathcal{T}^i)$ in descending order of their vertex number
 - 3: **communicate**: to determine the set of shared newest vertices $\mathcal{N}_{b_new}(\mathcal{T}^i) = \{n \in \mathcal{N}(\mathcal{T}^i)\}$ and n is a newest vertex and lies on the subdomain boundary
 - 4: **do**
 - 5: **for** $n \in \mathcal{N}(\mathcal{T}^i)$ **do**
 - 6: **if** n is a newest interior vertex or $n \in \mathcal{N}_{b_new}(\mathcal{T}^i)$ **then**
 - 7: execute lines 5-7 from the sequential algorithm
 - 8: **end if**
 - 9: **end for**
 - 10: **communicate**: repeat line 3
 - 11: **while** $\mathcal{N}_{b_new}(\mathcal{T}^i) \neq \emptyset$
 - 12: compress the gaps in the global numbering of the vertex, edge, and element sets
-

The number of iterations performed by Algorithm 9, and thus, the amount of interprocessor communications it incurs, is bounded by the length of the longest descending coarsening vertex chain. In Section 6, it will be shown that in practice, this is not inhibiting. Special attention should be paid to line 7 where the newest vertices are on the subdomain boundary. When any such vertex is deleted, the pair of children elements $s_1(T)$ and $s_2(T)$ are combined to form the parent element T . Two scenarios may be encountered: (1) both $s_1(T)$ and $s_2(T)$ reside in the same subdomain, in which case they are trivially combined; and (2) $s_1(T)$ and $s_2(T)$ belong to different subdomains, in which case T is assigned to the subdomain owning $s_1(T)$ and $s_2(T)$ is deleted from the subdomain it belongs to.

5 | OTHER ENABLERS

In the context of an EBM, the accuracy and efficiency of the AMR approach described so far rely on the computation of the distance to the wall, the tracking of the embedded fluid/structure interface, and the load balancing strategy. This section discusses these performance factors through a brief description of auxiliary tools that pertain to them.

5.1 | Fast computation of the distance to the wall

The computation at each time instance $t^{(n)}$, and for each vertex of the embedding fluid mesh, of its distance to the wall boundary $\Gamma_w(t)$ is an essential component of the AMR approach proposed in this paper, particularly for tracking and efficiently resolving a boundary layer during a viscous CFD or Eulerian FSI computation on a nonbody-fitted mesh using an EBM. It is also critical for many other computational modules in CFD such as, for example, the implementation of the Spalart-Allmaras (SA) turbulence model.⁵⁸ In the context of an unsteady, Eulerian, FSI computation, this component rapidly becomes computationally intensive if not carefully implemented. This is because during each time step, $\Gamma_w(t)$ evolves, and therefore the distance to the wall needs to be reevaluated at each vertex of the embedding fluid mesh. For example, the straightforward but naive approach for performing this task searches at each time step over the entire number of fluid mesh vertices, N_f , and the entire number of vertices of the discrete representation of the embedded surface, N_s . Hence, it has a computational complexity per time step of $\mathcal{O}(N_f N_s)$.

However, many methods exist in the literature for performing this task with a lower computational complexity and on massively parallel computing architectures. These include, perhaps most notably, the Fast Marching Method (FMM)⁵⁹

and the more recently developed Fast Sweeping Method⁶⁰ and Fast Iterative Method.⁶¹ The asymptotic complexity of the FMM and other marching type methods is $\mathcal{O}(N_f \log N_f)$, which is a substantial improvement over that of the naive approach. However, it was shown in the work of Grimberg and Farhat⁶² that for unsteady, viscous, FSI computations based on the SA turbulence model, which requires the computation of the distance to the wall, performed using explicit-explicit time stepping, which is typically the most robust and efficient approach for simulating highly nonlinear unsteady FSI problems with potentially material failure (for example, see the works of Wang et al²³ and Farhat et al^{45,63}), the cost of the computation of the distance to the wall using a fast marching type method can dominate the total computational cost of an Eulerian FSI simulation performed using an EBM, even in the absence of AMR.

Here, it is noted that as far as the proposed AMR framework is concerned, the accuracy of the computation of the distance to the wall matters primarily for the mesh adaptation criterion based on this distance (see Algorithm 1), which is in the boundary layer and its vicinity. In this case, an economical *estimation* of the distance function $f(X)$, where $X \in \mathbb{R}^3$, rather than the function itself can be computed, for example, using the fast approximate distance algorithm presented in the work of Grimberg and Farhat.⁶² This algorithm offers an even lower computational complexity than all aforementioned methods, but at the expense, in the worst case scenario, of a small degradation of the accuracy of the computed distance to the wall in the far field. It performs a single pass over the fluid mesh vertices ordered by mesh layer starting from the embedded surface, requires only the existing mesh connectivity data structure, does not perform any sorting of any data structure, and thus has a reduced computational complexity of $\mathcal{O}(N_f)$. When the mesh element size varies only as a function of its distance from the embedded surface, the mesh-layer-based vertex sorting reduces to sorting by the vertex distance value: In this case, the estimation algorithm presented in the work of Grimberg and Farhat⁶² becomes essentially a faster variant of the FMM. This type of mesh element size variation is typical of most CFD meshes, including those resulting from the AMR framework described above. Thus, the fast distance estimation algorithm described in the work of Grimberg and Farhat⁶² provides a useful means for computing more efficiently the distance to the wall in general, and accelerating Algorithm 1 in particular.

Still, for unsteady simulations where the position of the embedded surface is time dependent and thus the distance to the wall must be recomputed at each time step, it was also shown in the work of Grimberg and Farhat⁶² that the computational cost of even the fast aforementioned distance estimation algorithm can represent an unacceptable fraction of the total cost of an explicit-explicit viscous FSI simulation based on the SA turbulence model, and performed using an Eulerian EBM without AMR. For this reason, a predictor-corrector scheme for estimating the value of the distance to the wall based on updating this estimation only when otherwise a QoI that depends on it would become tainted by an unacceptable level of error, was also proposed in the work of Grimberg and Farhat.⁶² This scheme reduces significantly the overhead associated with computing the distance to the wall by adapting the frequency of reevaluating the distance function. For example, it was shown in the work of Grimberg and Farhat⁶² that monitoring an error metric defined as the relative change in the distance to the wall averaged over the computational fluid domain allows the forgoing of recomputing the distance function when the structural motion between two time steps is small enough relative to a specified tolerance. It was also shown in the work of Grimberg and Farhat⁶² that using this predictor-corrector scheme, coupled with the aforementioned fast approximate distance computation algorithm, consumes a negligible amount of overall CPU time in an unsteady FSI simulation performed using an EBM. Hence, this predictor-corrector scheme is adopted here to prevent the required knowledge of the distance to the wall in the proposed AMR framework from becoming a computational bottleneck.

5.2 | Interface tracker

Interface tracking is an essential component of not only the AMR approach described so far but also any EBM for CFD and/or FSI. Robust and computationally efficient interface tracking algorithms based on geometrical intersectors are available in the literature.^{24,25,64} For example, two such algorithms were discussed in the work of Wang et al²⁵: One that is representative of projection-based approaches, and another that is representative of collision-based approaches. Both of them have been shown to consume less than 10% of the total CPU time elapsed in typical Eulerian FSI simulations performed using an EBM. Although the projection-based interface tracker was shown to be the faster of the two algorithms, it is restricted to closed interfaces. Hence, the collision-based interface tracker, which is not restricted by this limitation and is capable of handling the embedding of open thin shell surfaces and subgrid resolution closed surfaces in arbitrary fluid meshes, is adopted in this work. For the sake of completeness, this algorithm is outlined below, together with the adjustments needed to suit the proposed AMR framework.

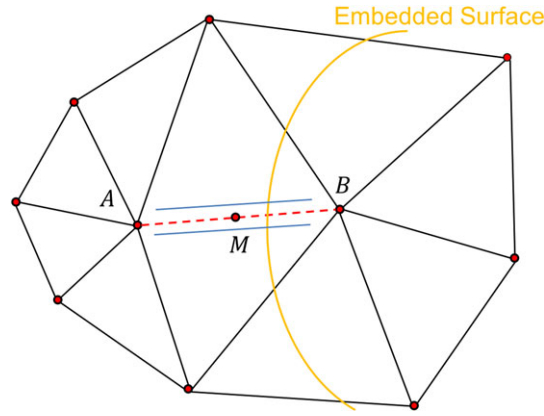


FIGURE 7 Illustration of the insertion in the computational fluid dynamics mesh of a new vertex (M) adjacent to the embedded surface (yellow line), due to the bisection of the edge \overline{AB} (note that the corresponding bisections of the adjacent elements are not shown) [Colour figure can be viewed at wileyonlinelibrary.com]

Algorithm 10 Collision-based interface tracking algorithm

- 1: build the bounding box hierarchy \mathcal{T}_{FS} for all triangle elements on the embedded surface (fluid-structure interface)
 - 2: for each vertex n_i of the CFD mesh, build the bounding box \mathcal{B}_i containing all vertices connected to this vertex by an edge
 - 3: for each vertex n_i of the CFD mesh, identify the set \mathcal{C}_i comprising all triangle elements of the embedded surface whose bounding boxes intersect \mathcal{B}_i
 - 4: for each edge $\overline{n_i n_j}$ of the CFD mesh, find its intersection point with the embedded surface and the corresponding element of this surface in $\mathcal{C}_i \cap \mathcal{C}_j$
 - 5: apply a flood fill algorithm²⁴ to populate the active/inactive status of each vertex of the CFD mesh
-

After each coarsening or refinement step, the above algorithm is invoked to compute the intersections between the embedded surface and the adapted CFD mesh, and to populate the active/inactive status of this same CFD mesh. A special treatment is used to populate the solution values at the new vertices of the CFD mesh adjacent to the embedded surface, which is the vertices introduced during the previous refinement step at the midpoints of the bisected edges that are intersected by the embedded surface. For example, consider the case shown in Figure 7. The intersected edge \overline{AB} is refined, leading to the insertion of a new vertex M in the CFD mesh. By locally applying steps 2-4 of Algorithm 10, it can be readily determined whether \overline{AM} or \overline{BM} intersects the embedded surface. In the example shown in Figure 7, \overline{AM} is not intersected but \overline{BM} is intersected. In this case, the density and pressure values at M are populated by constant extrapolation from A , which is

$$\rho_M = \rho_A \quad \text{and} \quad p_M = p_A,$$

where ρ_A and p_A are the density and pressure of the fluid at the vertex A , respectively. The velocity at M can be populated either by constant extrapolation from A , or by linear interpolation between A and the point at which the edge \overline{AB} intersects the embedded surface. Using constant interpolation gives

$$\vec{v}_M = \vec{v}_A,$$

whereas using linear interpolation gives

$$\vec{v}_M = \text{interpolation}(\vec{v}_A, \vec{v}_{\text{wall}}),$$

where \vec{v}_A denotes the velocity of the fluid at vertex A , and \vec{v}_{wall} denotes the velocity of the embedded surface at the intersection point.

5.3 | Load balancing strategy

It is well known that the parallel implementation of AMR, which is essential for large-scale CFD computations, requires a load balancing strategy. In this work, this strategy is based on a load imbalance indicator and mesh repartitioning using the ParMETIS⁶⁵ library. For simplicity, the indicator is defined as the ratio of the maximum and minimum numbers of vertices

mapped onto any processor. The number of mesh vertices relates linearly to the number of degrees of freedom (dofs) and therefore is a good, even if imperfect, indicator of the amount of work to be done by a processor. If the aforementioned ratio exceeds a certain user-defined threshold, load balancing is performed.

6 | APPLICATIONS

First, the implementation of the proposed AMR framework for EBMs is verified through its application to the solution of a few academic FSI problems. Next, the potential of this framework for challenging, highly nonlinear, FSI applications is demonstrated with the simulation of a supersonic parachute inflation dynamics problem. In all cases, the flow is assumed to be compressible, and the considered EBM is the FIVER method^{123,43-47} implemented in AERO Suite.^{6,7} The combination of the proposed AMR framework and EBM FIVER is referred to as the AMR-EBM framework, and the numerical results obtained using this framework are labeled as AMR-EBM. Whenever comparisons are made between ALE and AMR-EBM results, such results are obtained using ALE and FIVER schemes of the same order of accuracy in space and time. In all cases, all computations are performed on a Linux cluster with 256 cores and 4 GB of memory/core, and all FSI simulations where the structure is flexible are performed with 255 cores assigned to the fluid subsystem and 1 core assigned to the structural subsystem.

6.1 | Shock tube problem

The shock tube problem introduced in the work of Sod⁶⁶ is often used as a benchmark problem. It has a known analytical solution, which makes it ideal for assessing the accuracy of computational schemes. Here, this inviscid problem is used to verify the implementation in AERO Suite of the proposed AMR framework equipped with the Hessian-based adaptation criterion.

The three-dimensional setup of this otherwise one-dimensional flow problem is shown in Figure 8. The tube has a square cross section of nondimensional size 0.1×0.1 , and a nondimensional length equal to 5. It contains two identical ideal gases separated by a thin membrane placed at its middle cross section. The initial states of the gases are

$$\rho_L = 1.0, p_L = 1.0, v_L = 0.0 \quad \text{and} \quad \rho_R = 0.125, p_R = 0.1, v_R = 0.0,$$

where ρ , p , and v denote the nondimensional density, pressure, and velocity, respectively, and the subscripts L and R designate the left and right sides of the membrane.

The computational fluid domain is set to the three-dimensional interior of the tube. Outlet boundary conditions are applied at its left and right ends, and slip boundary conditions are prescribed at all its wall surfaces. This domain is discretized using a uniform mesh with tetrahedral elements of nondimensional size $h = 0.05$. Time discretization is performed using the second-order explicit Runge-Kutta scheme, and an adaptive time step based on the CFL number of 0.5. At $t = 0$, the membrane is removed. The resulting flow contains a rarefaction wave, a contact discontinuity, and a shock. It is simulated until $T = 1.0$, using AMR and the Hessian criterion based on the flow density – that is, $u_q = \rho$. The thresholds e_h^{\max} and e_h^{\min} for the Hessian criterion are set to 5×10^{-3} and 1×10^{-3} , respectively, and mesh adaptation is performed every 10 time steps. At the final simulation time $T = 1.0$, none of the waves has reached either end boundary of the tube and therefore none of the waves has reflected on either end boundary.

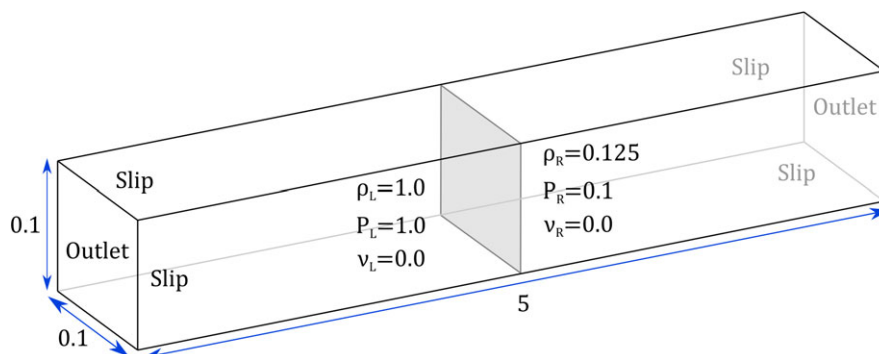


FIGURE 8 Shock tube: Problem setup. [Colour figure can be viewed at wileyonlinelibrary.com]

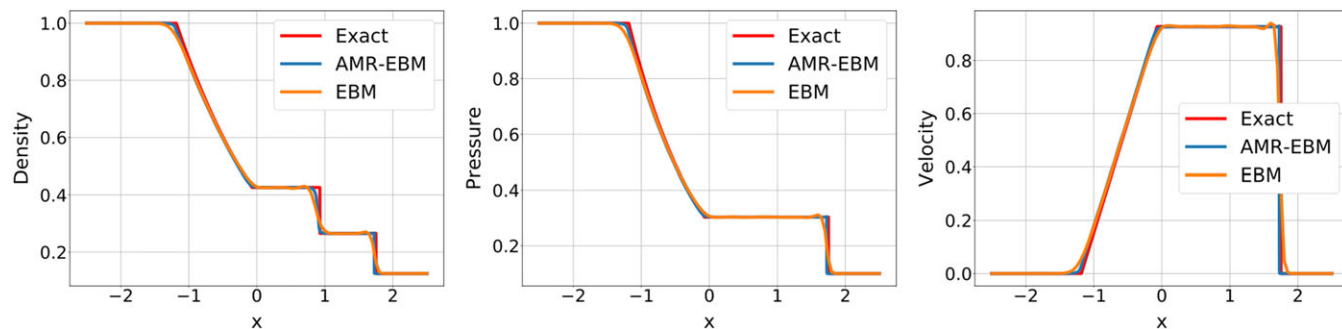


FIGURE 9 Shock tube problem: Comparison of the numerically approximated and exact profiles of the density, pressure, and velocity of the flow at time $t = 1.0$. AMR-EBM, adaptive mesh refinement–embedded boundary method [Colour figure can be viewed at wileyonlinelibrary.com]

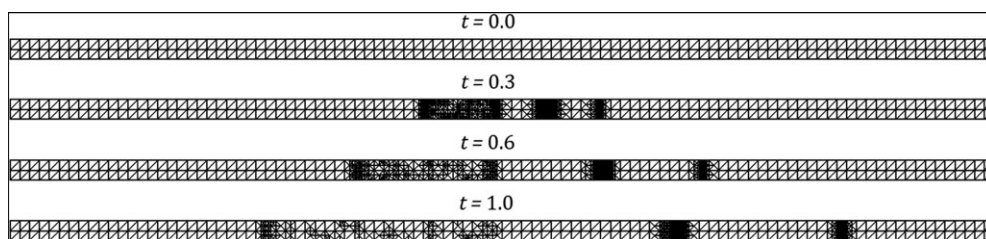


FIGURE 10 Shock tube problem: snapshots of the adapted mesh at $t = 0.0$, $t = 0.3$, $t = 0.6$, and $t = 1.0$

Figure 9 compares the density, pressure, and velocity profiles predicted with and without AMR at time $T = 1.0$ against their counterparts extracted from the exact, analytical solution of this shock tube problem. The reader can observe that the solution computed using AMR is almost identical to the exact solution, whereas the solution computed without AMR oscillates slightly near the shock. In particular, sharp discrete representations of the shock and contact discontinuity profiles are obtained, thanks to AMR.

Figure 10 reports on the time evolution of the adapted mesh via the display of snapshots of this mesh taken at different simulation times. It can be observed that the Hessian-based mesh adaptation criterion tracks and resolves all of the shock, contact discontinuity, and rarefaction waves.

6.2 | Heaving airfoil problem

The problem considered here is designed to verify the implementation in AERO Suite of the proposed AMR framework equipped with the distance-based mesh adaptation criterion, and therefore to verify the ability of this framework to track and resolve the boundary layer. Its setup is graphically depicted in Figure 11. Essentially, the problem consists in computing the two-dimensional, unsteady, viscous flow past a NACA 0012 airfoil with a nondimensional chord $c = 1$, when this airfoil is set in the heaving motion described by

$$q(t) = 0.2(1 - e^{-t^2}) \sin t, \quad (8)$$

where q denotes the nondimensional displacement of the airfoil in the y -direction. For this purpose, the computational domain is chosen to be the three-dimensional domain constituted by the square of nondimensional size $100c \times 100c$ in the xy -plane containing the airfoil, and of nondimensional thickness in the z direction equal to 0.005. This computational domain is centered at the middle of the chord of the airfoil. The origin of the xyz coordinate system is located at the leading edge of the airfoil. The boundary conditions applied to this computational domain are as specified in Figure 11.

The viscous flow is assumed to be laminar with a chord-based Reynolds number $Re = 5,000$. The free-stream Mach number and angle of attack are set to $M_\infty = 0.5$ and $\alpha_\infty = 3^\circ$, and therefore the viscous flow is subsonic.

First, a nondimensional reference solution is computed for this problem in the time interval $[0, 10]$ corresponding to one-and-a-half periods of the heaving motion (8), using a body fitted, tetrahedral mesh, and AERO Suite. Time discretization is performed using the Discrete Geometric Conservation Law–obeying, ALE extension of the second-order, explicit, Runge-Kutta scheme described in the work of Brogniez et al⁶⁷ and an adaptive fluid time step based on the CFL number of

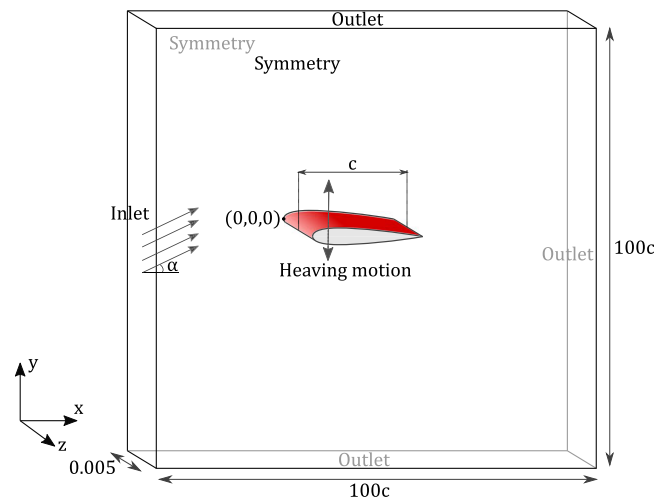


FIGURE 11 Heaving airfoil: Problem setup [Colour figure can be viewed at wileyonlinelibrary.com]

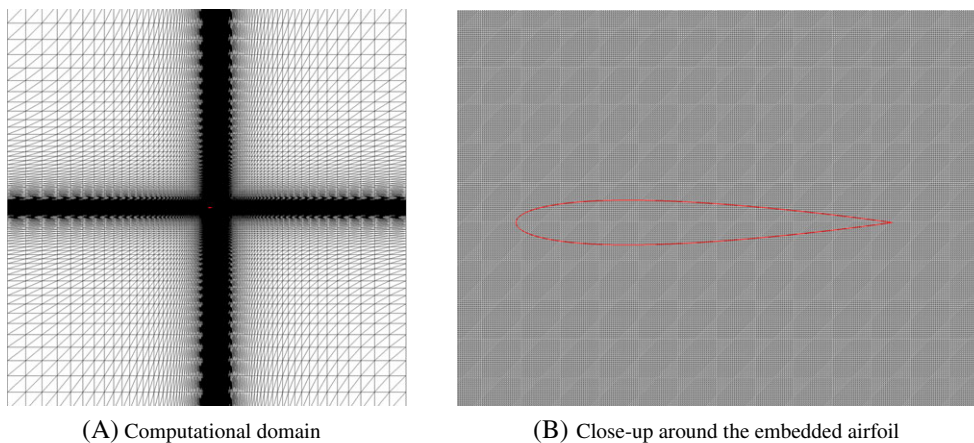


FIGURE 12 Heaving airfoil problem: Initial (background) computational fluid dynamics mesh [Colour figure can be viewed at wileyonlinelibrary.com]

0.8. In this case, the body-fitted CFD mesh is generated such that it resolves the boundary layer around the initial position of the airfoil. During the ALE flow computations, it is rigidly moved according to the same heaving motion (8). Therefore, the boundary layer is tracked and resolved at all times during this first ALE simulation. Specifically, the body-fitted CFD mesh is generated using GMSH.⁶⁸ It has a single layer of tetrahedral elements in the z direction, because the flow problem considered here is two dimensional. It is sized such that the computed lift and drag forces are numerically converged. In particular, the characteristic element edge length near the airfoil is specified to 8×10^{-3} so that the boundary layer that forms around the airfoil is well resolved. For this purpose, the boundary layer thickness is estimated a priori using a flat plate boundary layer theory to be $6.9 \times 10^{-2} c$. To capture the wake generated by the heaving motion, the region of the computational domain defined by the box $[-0.5 c \leq x \leq 6 c, -c \leq y \leq c]$ is meshed such that the typical element edge length in this region is around 5×10^{-3} . The resulting body-fitted CFD mesh has 1,676,071 vertices.

Next, a nondimensional solution for this problem is computed using a nonbody-fitted tetrahedral mesh, AERO Suite, and its EBM FIVER. The embedding CFD mesh, which is shown in Figure 12, has initially 626,862 vertices. It is a rather fine mesh because it resolves the wake expected to develop during the heaving motion. This is because the sole purpose of this problem is to verify the ability of the proposed AMR framework equipped with the distance-based mesh adaptation criterion to track and resolve the boundary layer. For this purpose, the parameters of the mesh adaptation distance criterion are set as follows: $d_o = 10^{-1}$, $h_o = 8 \times 10^{-3}$, $D_i = 4 \times 10^{-3}$, and $y_1^{+*} = 1.0$.

Each of the aforementioned simulations is initialized using the steady-state flow solution computed for the same problem setup but with a fixed airfoil (see Figure 13B). For the ALE simulation, this steady-state solution is computed on the initial position of the ALE mesh. On the other hand, the simulation based on the EBM FIVER is performed on a

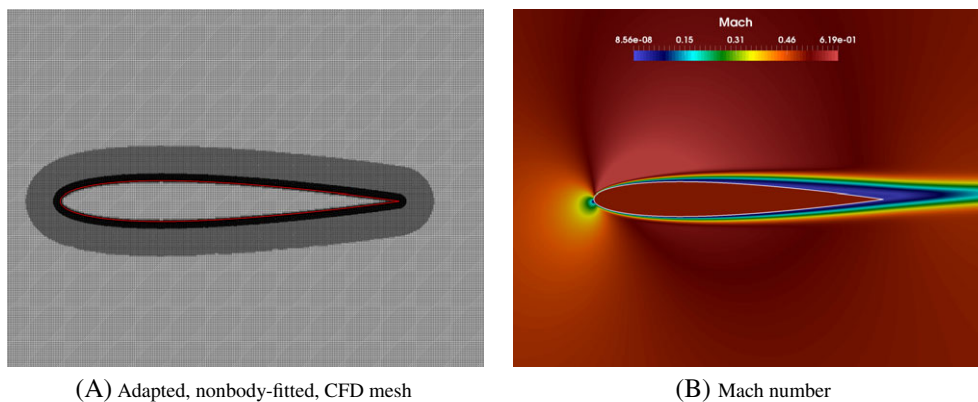


FIGURE 13 Heaving airfoil problem: Initial mesh adaptation and computed steady-state, viscous flow solution. CFD, computational fluid dynamics [Colour figure can be viewed at wileyonlinelibrary.com]

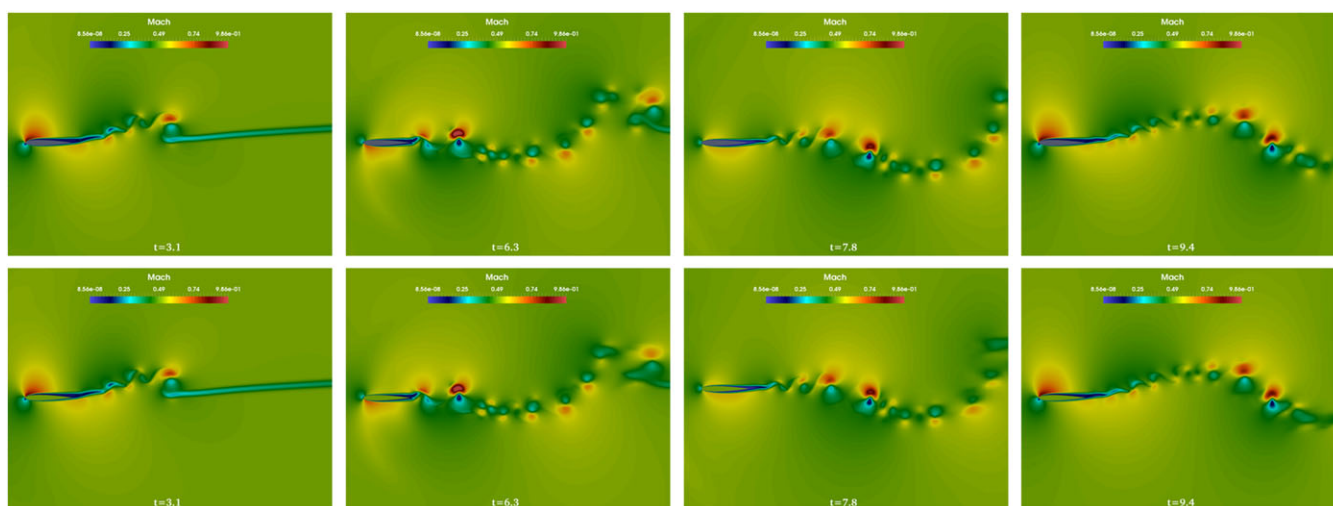


FIGURE 14 Heaving airfoil problem: Contour plot snapshots of the local Mach number computed using the Arbitrary Lagrangian Eulerian (top) and Adaptive Mesh Refinement-Embedded Boundary Method (bottom) frameworks: $t = 3.1$, $t = 6.3$, $t = 7.8$, and $t = 9.4$ (left to right). [Colour figure can be viewed at wileyonlinelibrary.com]

version of the initial embedding mesh adapted using the same distance criterion but for the steady-state flow solution. The initially adapted embedding mesh has 708,753 vertices. It is shown in Figure 13A, together with the steady-state flow solution computed on this non body-fitted CFD mesh.

Figure 14 displays snapshots of the Mach contour plots of the unsteady flow solutions computed using both of the ALE and AMR-EBM computational frameworks. They feature vortices that are generated near the airfoil trailing edge and propagate downstream. The vortices computed using the AMR-EBM framework are almost identical to those computed using the ALE framework.

Figure 15 reports on the time evolution of the adapted embedding mesh by displaying three snapshots of this evolution that collectively cover one-and-a-quarter periods of the heaving motion, and are associated with the maximum amplitude and mean positions of the airfoil in the embedding computational domain. The number of vertices in each adapted CFD mesh shown in Figure 15 is 713,616 for the leftmost snapshot, 716,996 in the middle one, and 713,657 in the rightmost snapshot. This indicates that the size of the adapted CFD mesh remains at all times roughly equal to that of the initially adapted CFD mesh (708,753 vertices). This is expected because given that the heaving airfoil is rigid, the performed AMR should effectively translate the meshed boundary layer region, which it does.

Figure 16 compares the time histories of the lift and drag coefficients computed using the ALE and AMR-EBM frameworks. It shows that essentially, the AMR-EBM framework delivers the same results as its ALE counterpart. The perception that the curve labeled AMR-EBM is thicker than its counterpart associated with the ALE framework is due, as can be expected, to the sensitivity of the EBM computations to the noise generated by the nonbody-fitted nature of the

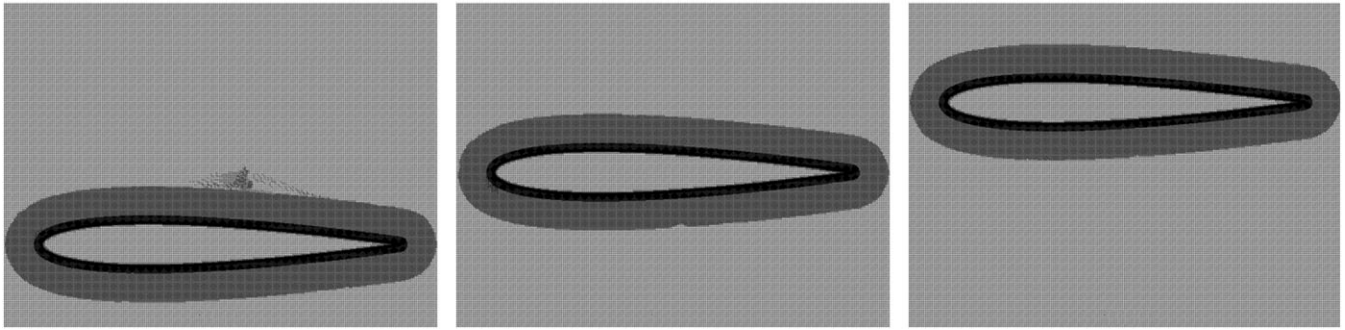


FIGURE 15 Heaving airfoil problem: Time evolution of the adapted embedding mesh: $t = 4.7$, $t = 6.3$, and $t = 7.8$ (left to right)

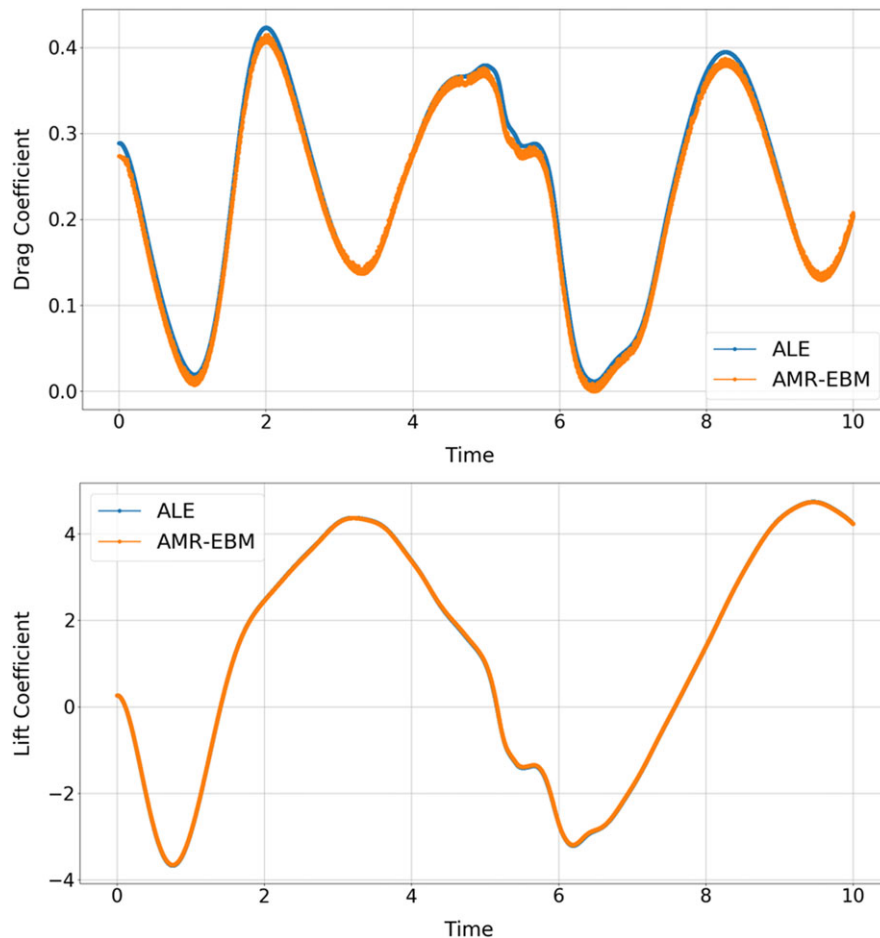


FIGURE 16 Heaving airfoil problem: Time histories of the lift and drag coefficients computed using the Arbitrary Lagrangian Eulerian (ALE) and Adaptive Mesh Refinement-Embedded Boundary Method frameworks (AMR-EBM) [Colour figure can be viewed at wileyonlinelibrary.com]

embedding and embedded meshes. In any case, all results reported in Figure 15 and Figure 16 demonstrate the ability of the proposed AMR framework to track a moving boundary layer and maintain it at all times well resolved.

6.3 | Flow past a sphere problem

Next, the problem of simulating numerically a supersonic viscous flow past a sphere of nondimensional diameter equal to one is considered. Both experimental and computational data are available in the literature for such a problem,⁶⁹⁻⁷¹ which makes it suitable for the validation of a computational framework. The problem setup considered here is shown in Figure 17. The flow parameters are chosen such that the free-stream Reynolds number based on the diameter of the sphere

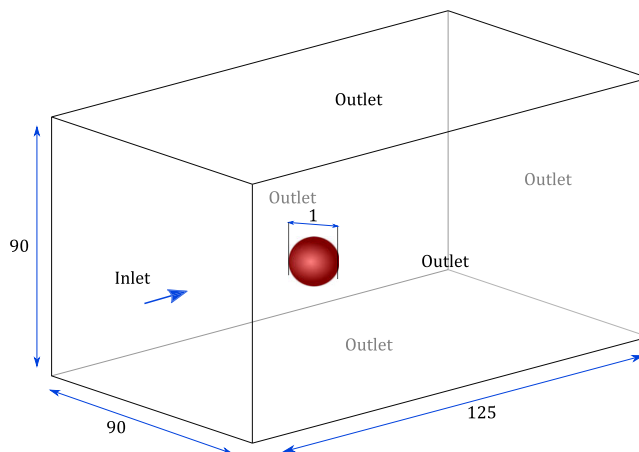


FIGURE 17 Flow past a sphere: Problem setup [Colour figure can be viewed at wileyonlinelibrary.com]

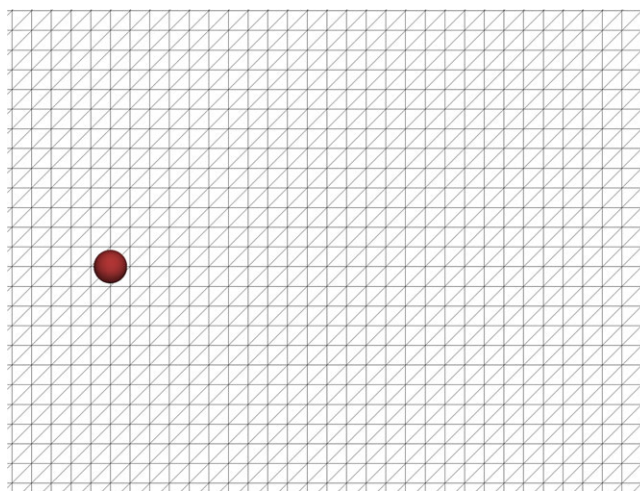


FIGURE 18 Flow past a sphere problem: Section of the initial mesh in the xy -plane of symmetry [Colour figure can be viewed at wileyonlinelibrary.com]

is $Re = 300$. Hence, the viscous flow is assumed to be laminar. Specifically, two different free-stream flow conditions are considered: $M_\infty = 1.5$ and $\alpha_\infty = 0^\circ$, and $M_\infty = 2.0$ and $\alpha_\infty = 0^\circ$. In each case, the flow has a steady-state solution.

The computational domain shown in Figure 17 is discretized using an initial tetrahedral mesh with 301,401 vertices. The two-dimensional cut of this background mesh through the xy -plane of symmetry is displayed in Figure 18. For each considered flow condition, the solution of the corresponding viscous flow problem has multiple interesting features: A boundary layer, flow separation, and a bow shock in front of the sphere. Hence, the AMR framework is configured here with both the distance-based and Hessian-based mesh adaptation criteria. The parameters of the distance-based criterion are set to $d_o = 2.0$, $h_o = 2.0$, $D_i = 0.01$ and $y_1^{+*} = 0.2$. For the Hessian-based criterion, the QoI u_q whose Hessian is computed and monitored is set to the magnitude of the flow velocity. The thresholds e_h^{\max} and e_h^{\min} are set to $e_h^{\max} = 5 \times 10^{-3}$ and $e_h^{\min} = 1 \times 10^{-3}$. For each performed steady-state simulation, the flow solution is initialized using the uniform flow associated with the free-stream boundary conditions and mesh adaptation is started when the 2-norm of the nonlinear residual relative to the 2-norm of the initial nonlinear residual becomes smaller or equal to 10^{-6} . Convergence is declared when the relative 2-norm of this residual becomes smaller than 10^{-8} .

Figure 19 shows for both considered free-stream flow conditions cutviews of the adapted embedding meshes in the xy -plane of symmetry, and the corresponding contour plots of the local Mach number. The adapted CFD meshes have 6.36 million vertices for $M_\infty = 1.5$, and 5.02 million vertices for $M_\infty = 2.0$.

The obtained numerical results are validated by comparing, for each considered free-stream flow conditions, the predicted shock stand-off distance and the flow separation angle with counterpart theoretical estimates⁶⁹ and experimental results^{70,71} reported in the literature. The shock stand-off distance is defined as the distance from the forward stagnation

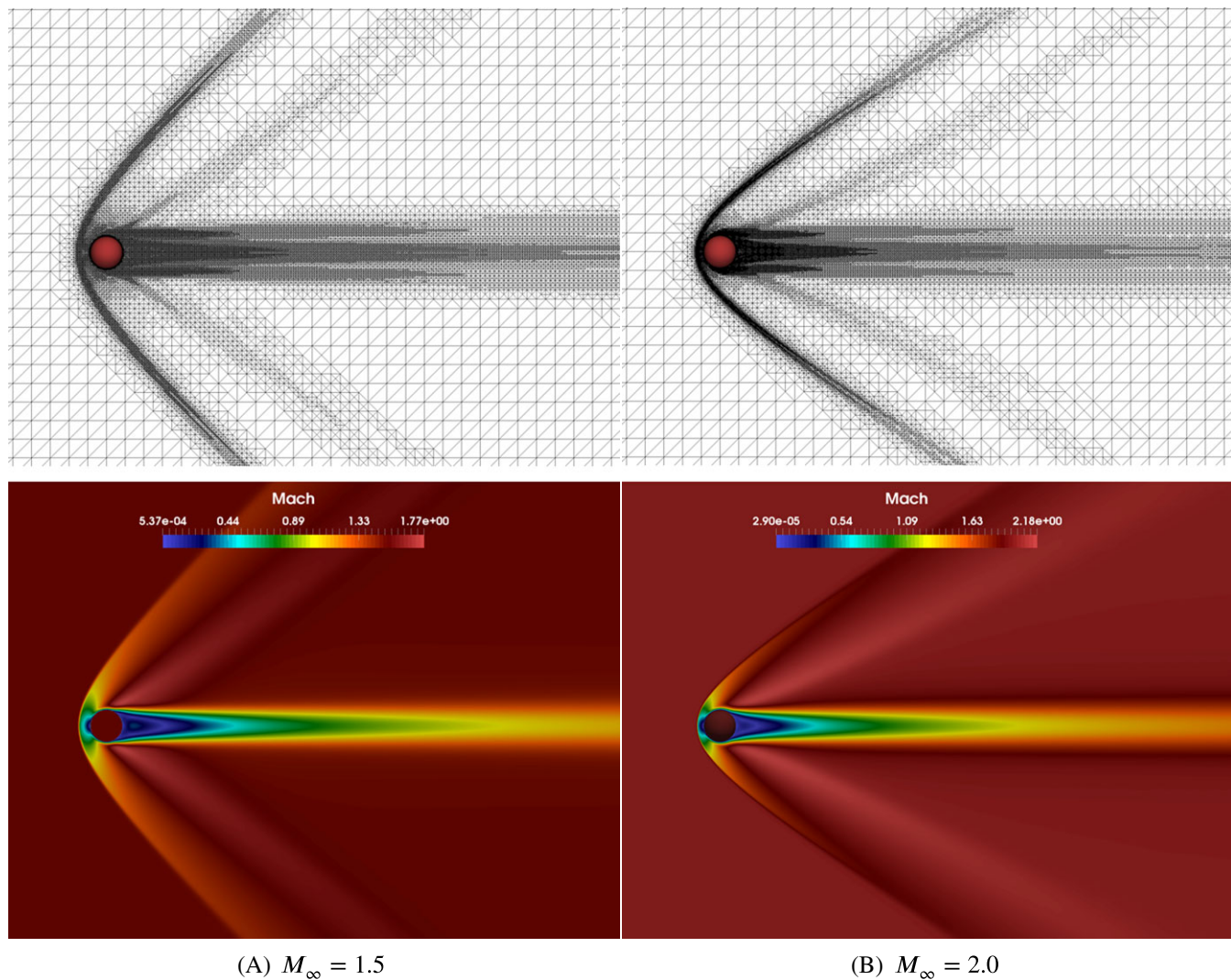


FIGURE 19 Flow past a sphere problem: Cutviews of the adapted embedding meshes and corresponding contour plots of the computed local Mach number [Colour figure can be viewed at wileyonlinelibrary.com]

point to the detached shock wave. The reader can observe in Figure 20 that the numerical results obtained using the proposed AMR-EBM framework are within the bounds of the variation between the results reported in the literature.

The flow separation angle is defined as the angle subtended at the center of the sphere by the arc connecting the forward stagnation point to the point on the sphere where the skin friction is zero. While there are many points on the sphere where the skin friction vanishes, due to symmetry, the point referenced here can be considered to be in the xy -plane passing through the center of the sphere. This angle can be measured in any plane passing through the center of the sphere and the forward stagnation point. For this problem, its computed values for both considered free-stream flow conditions are reported in Table 2. These values differ by about 4% from the counterpart numerical simulation values reported in the work of Nagata et al.⁷¹ This discrepancy can be considered to be acceptable because the reference values reported in Table 2 were obtained using a computational approach that was validated only in the subsonic regime, and in that regime, its predictions of the flow separation angle were tainted by relative errors of the order of 2.5%.⁷¹

6.4 | Supersonic parachute inflation dynamics problem

The last problem considered in this paper focuses on the inflation dynamics of a Disk-Gap-Band (DGB) parachute in a supersonic flow. Its main purpose is to demonstrate the potential of the AMR-EBM computational framework for challenging, highly nonlinear, FSI problems. Specifically, the problem setup considered here is shown in Figure 21A.

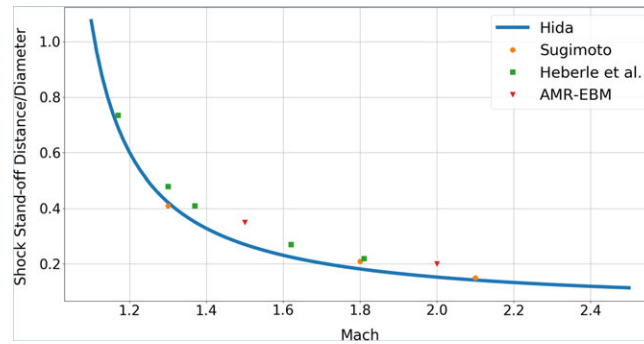


FIGURE 20 Flow past a sphere problem: Shock stand-off distance at $Re = 300$ – Sugimoto's experimental results (orange circles), experimental results of Heberle et al (green squares), Hida's theoretical estimations (blue line), and numerically predicted results using the Adaptive Mesh Refinement–Embedded Boundary Method (AMR-EBM) framework (red triangles) [Colour figure can be viewed at wileyonlinelibrary.com]

TABLE 2 Flow past a sphere problem: computed flow separation angles at two different free-stream Mach numbers

	$M_\infty = 1.5$	$M_\infty = 2.0$
AMR-EBM	132°	144°
Nagata et al ⁷¹	137°	150°

AMR-EBM, adaptive mesh refinement–embedded boundary method.

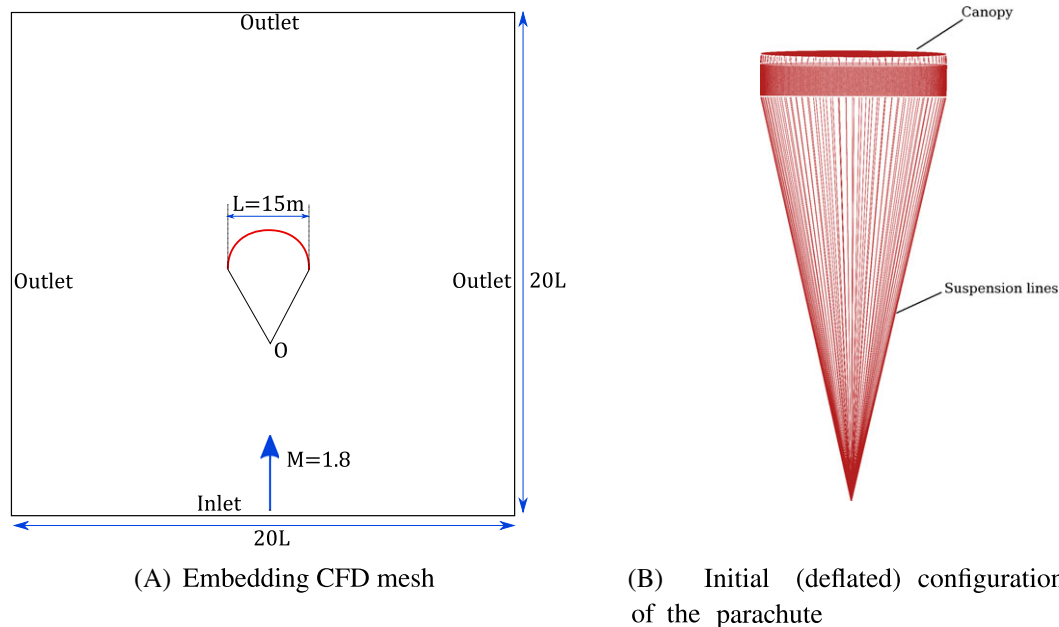


FIGURE 21 Supersonic parachute inflation dynamics: Problem setup. CFD, computational fluid dynamics [Colour figure can be viewed at wileyonlinelibrary.com]

The computational domain is a cube of size $20L \times 20L \times 20L$, where L is the diameter of the parachute canopy. The suspension lines of the parachute originate at the point O in Figure 21A and connect to the canopy.

The geometrical and mechanical properties of the parachute are summarized in Table 3.

The initial condition for the considered inflation problem is shown in Figure 21B: The suspension lines are already deployed but stress free; the canopy of the DGB parachute is deflated, in a flat position, and stress-free; and the supersonic

TABLE 3 Geometrical and material properties of the parachute⁷²

Component	Properties
Canopy	Diameter = 15.447 m Thickness = 7.607×10^{-5} m $E = 0.945$ GPa $\nu = 0.4$ $\rho = 1154.25$ kg/m ³ Porosity = 0.128
Suspension lines	Length = 36.56 m Radius = 1.6×10^{-3} m $E = 29.5$ GPa Density = 1154.25 kg/m ³

flow is incoming at $M_\infty = 1.8$, $\alpha_\infty = 90^\circ$, $\rho_\infty = 0.0067$ kg/m³, and $p_\infty = 260$ Pa. Note that the specified free-stream density and pressure conditions are similar to those observed in the standard Martian atmosphere.

The canopy of the DGB parachute consists of band gores and disk gores. These are discretized here by 69,756 geometrically nonlinear thin-shell ANDES elements⁷³ (although the membrane stiffness of the fabric is significantly larger than its bending stiffness, both stiffnesses are accounted for here). Each of the 80 suspension lines is discretized by 376 geometrically nonlinear beam elements. Consequently, the FE structural model of the DGB parachute is nonlinear and has 337,680 dofs.

Since the Martian atmosphere is mainly composed of carbon dioxide, the viscosity of this gas is modeled using Sutherland's viscosity law, which is $\mu = \mu_0 \sqrt{T}/(1 + T_0/T)$ – with $\mu_0 = 1.57 \times 10^{-6}$ kg/(m·s) and $T_0 = 240$ K. The Reynolds number based on the canopy diameter is 4.06×10^6 : Hence, the flow is assumed to have transitioned to the turbulent regime, which is modeled here using the SA turbulence model. The embedding computational fluid domain is initially discretized by a mesh composed of Kuhn simplices (see Appendix A). Specifically, this initial tetrahedral mesh contains 1,531,250 vertices and 8,948,832 tetrahedra.

First, the parachute configured in its initial position shown in Figure 21b is assumed to be rigid and the steady-state solution of the flow is computed. Next, the fluid state is initialized using this steady-state flow solution, the deformable structure is initialized using its initial state described above, and the FSI problem associated with the inflation of the DGB parachute is simulated in the time interval $[0, 0.2]$ s, during which the inflation process is expected to have completed, using the AMR-EBM framework configured as described below:

1. Time discretization of the FSI problem is performed using the second-order, time-accurate, implicit-explicit fluid-structure staggered solution procedure developed in the work of Farhat et al,⁶³ and the fluid-structure coupling time-step $\Delta t_{F/S} = 10^{-5}$ s.
2. AMR is applied using both the distance-based and Hessian-based mesh adaptation criteria as follows:
 - (a) In the evolving boundary layer, achieving the resolution of $y_1^+ = 1$ requires a typical edge length in the vicinity of the dynamic wall surfaces of the order of 10^{-4} m. Given the available computational resources, this leads to a prohibitively computationally intensive simulation. For this reason, a higher value of y_1^+ is inputted instead to Algorithm 4. Specifically, three different simulations are performed using three different values of y_1^+ , namely, $y_1^{+*} = 500$, $y_1^{+*} = 250$, and $y_1^{+*} = 125$, to examine the effect of y_1^+ on the predicted time history of the drag force. In all three simulations, the other parameters of Algorithm 4 are set as follows: $d_o = 1.0$ ms, $h_o = 1.0$ m, and $D_i = 2h_i$.
 - (b) For the Hessian-based criterion, two QoIs u_q are selected, namely, the flow density and its velocity, and therefore two Hessians are computed and monitored. Mesh refinement is performed when either Hessian-based criterion calls for mesh refinement, but mesh coarsening is performed when both Hessian-based criteria call for mesh coarsening. The thresholds e_h^{\max} and e_h^{\min} are set to $e_h^{\max} = 1 \times 10^{-1}$ and $e_h^{\min} = 2 \times 10^{-2}$.
3. AMR is performed every 50 time steps.

The time histories of the drag force computed by postprocessing all three aforementioned FSI simulations are plotted in Figure 22. They suggest mesh convergence in the boundary layer region (for the considered Reynolds Averaged Navier-Stokes computation) as the FSI simulation performed using $y_1^{+*} = 125$ leads to almost the same drag results as that

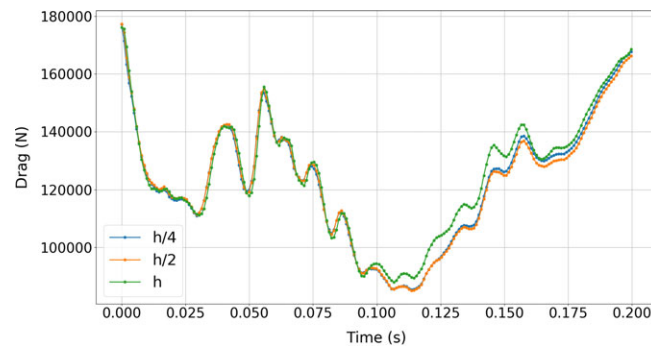
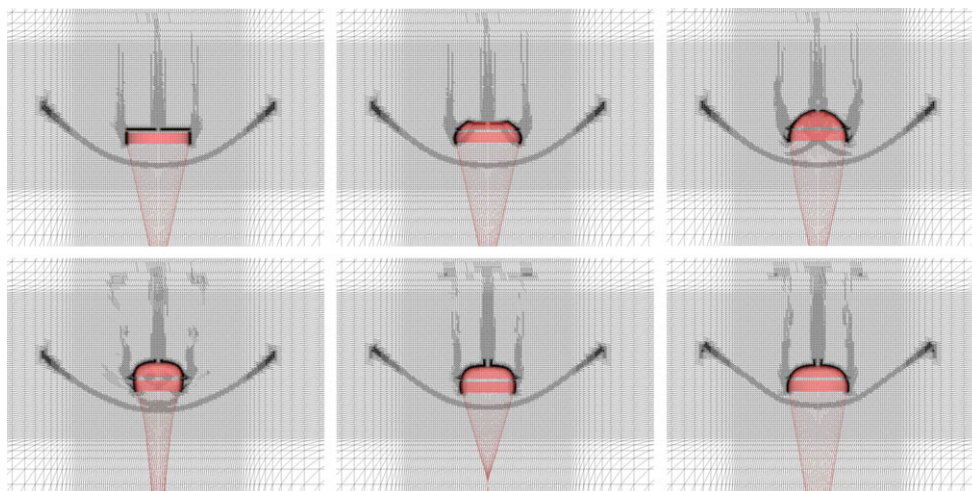
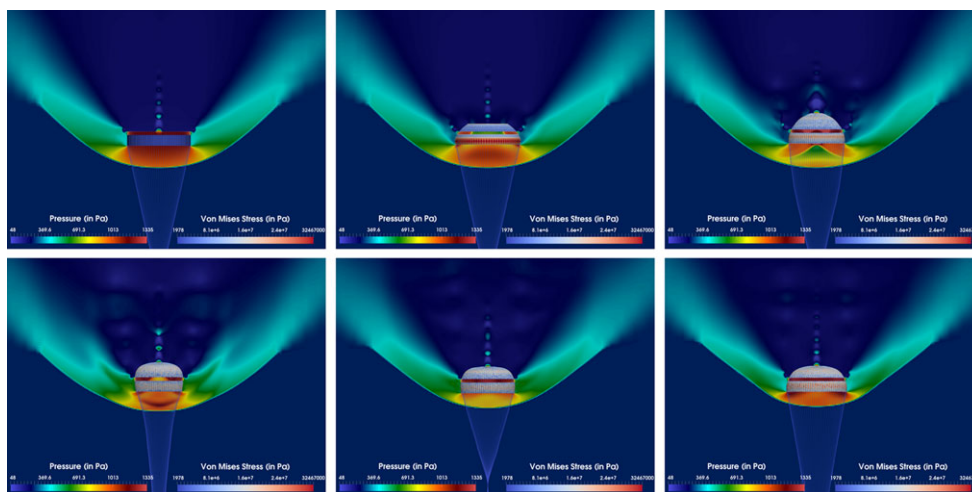


FIGURE 22 Drag histories of the supersonic parachute inflation dynamics problem computed using the Adaptive Mesh Refinement–Embedded Boundary Method framework and different mesh resolutions in the boundary layer ($y_1^{+*} = 500$) [Colour figure can be viewed at wileyonlinelibrary.com]



(A) Time evolution of the adapted embedding mesh



(B) Time evolution of the flow pressure and Von Mises stress in the parachute

FIGURE 23 Supersonic parachute inflation dynamics problem: Snapshots of the computed time evolution of the adapted computational fluid dynamics mesh and fluid-structure solution: $t = 0.0$ seconds, $t = 0.025$ seconds, $t = 0.05$ seconds, $t = 0.10$ seconds, $t = 0.15$ seconds, and $t = 0.20$ seconds (left to right, top to bottom) [Colour figure can be viewed at wileyonlinelibrary.com]

performed using $y_1^{+*} = 250$. This could be an indication that during the inflation process, the drag is mainly due to pressure drag. Figure 23 graphically depicts the time evolution of the adapted embedding mesh, the pressure of the fluid flow,

TABLE 4 Detailed performance of Adaptive Mesh Refinement (AMR) for the supersonic parachute inflation dynamics simulations performed on 256 cores of a Linux cluster

Simulation Component	Simulation 1 ($y_1^{+*} = 500$) Wall clock time (s)	Simulation 2 ($y_1^{+*} = 250$) Wall clock time (s)	Simulation 3 ($y_1^{+*} = 125$) Wall clock time (s)
AMR	3,929.07	5,854.46	10,110.75
Hessian precomputation	222.27	426.45	613.28
Edges marking	16.87	32.38	82.18
Mesh refinement	116.42	303.95	446.61
Mesh coarsening	286.29	476.57	689.79
Global ID assignment	123.37	384.90	1099.94
Global ID compression	71.02	199.36	588.83
Resizing and populating	181.41	256.44	316.04
Recomputation of F-S intersections	1,397.17	1,803.33	2,747.23
Recomputation of distance to wall	550.76	812.76	1,466.2
Mesh repartitioning	887.61	1,047.54	1,853.94
Data redistribution	75.88	110.78	206.71
Total simulation	88,343.10	121,717.91	217,327.91

and the von Mises stress in the parachute. It shows that the boundary layer, bow shock in the front of the canopy, jet-like flow ejected through the canopy vent and gaps between the band/disk gores, and small flow features inside the canopy are captured during the inflation process by the proposed AMR approach. The parachute is fully inflated at $t = 0.05$ seconds, time after which it starts breathing. A noteworthy point for the design and analysis of the integrity of such a parachute is that the maximum von Mises stress, which can be related to material failure, occurs in this case before the parachute is fully inflated.

At the end of the three FSI simulations performed using three different specified mesh resolutions in the boundary layer region, the sizes of the adapted embedding meshes are as follows: 2,798,847 vertices for the FSI simulation performed using $y_1^{+*} = 500$ (first FSI simulation), 3,690,336 vertices for that performed using $y_1^{+*} = 250$ (second FSI simulation), and 7,355,367 vertices for that performed using $y_1^{+*} = 125$ (third FSI simulation). After the parachute is fully inflated, the mesh size remains roughly constant in each of the three simulations. Load balancing is observed to redistribute 10,000, 40,000, and 150,000 vertices during the first, second, and third FSI simulation, respectively. Table 4 reveals that in all cases, the elapsed time in AMR constitutes less than 5% of the total wall clock time of the performed FSI simulation, and load balancing, which incurs mesh repartitioning and data redistribution, accounts for less than 25% of the wall clock time elapsed in AMR.

7 | SUMMARY AND CONCLUSIONS

Immersed Boundary Methods or EBMs continue to gain popularity for the solution of highly nonlinear, FSI problems characterized by large structural displacements/rotations/deformations and/or by topological changes. For such problems, alternative methods based on Lagrangian or ALE approaches are at least impractical for high-speed flows in the first case, and unreliable in the second case because of the likely mesh entanglement issues. For viscous flow problems, however, EBMs necessitate AMR as they do not track the boundary layers that form around embedded obstacles and therefore cannot maintain them resolved. To address this issue, an AMR framework for EBMs is presented in this paper. This framework incorporates two different mesh adaptation criteria: One based on computing the distance to the wall and satisfying a specified y^+ requirement; and one based on the Hessian of one or several quantities of interest for tracking and resolving flow features such as shocks, vortices, and wakes. It is amenable to parallel processing and is equipped with load balancing for achieving parallel efficiency on a large number of processors. For this purpose, it relies on ParMETIS for performing dynamic repartitioning of the adapted embedding mesh. This AMR framework is sufficiently comprehensive to support many discretization methods. It is particularly attractive for vertex-centered finite volume schemes and FE methods, as it maintains mesh conformity during the adaptation process. Its potential for challenging, highly nonlinear, FSI simulations is demonstrated through its application to a difficult supersonic PID problem. For this FSI problem, it tracks and

resolves the boundary layer and expected shocks, wakes, vortices, and other flow features. Particularly noteworthy is the fact that on 256 cores, it accounts for less than 5% of the total wall clock time of a highly nonlinear FSI computation such as a PID simulation. Equally noteworthy is that, thanks to ParMETIS and a fast data redistribution scheme, dynamic load balancing on a distributed memory parallel processor accounts for less than 25% of the total cost of AMR. The described AMR framework is equally attractive for purely CFD problems, as it introduces a high degree of automation in mesh generation for EBMs. For CFD problems characterized by rough wall surface textures, as in, for example, heat shields of reentry vehicles, iced aircraft wings, feather-covered bird wings, and submarine hulls with biofouling, where the characteristic length scale of the roughness is much smaller than that of the wall surface, the highly automated meshing for EBMs enabled by this AMR framework becomes a significant advantage over body-fitted meshing.

ACKNOWLEDGEMENTS

Raunak Borker, Daniel Z. Huang, Phil Avery, and Charbel Farhat acknowledge partial support by the Jet Propulsion Laboratory (JPL) under contract JPL-RSA No. 1590208, and partial support by the National Aeronautics and Space Administration (NASA) under Early Stage Innovations grant NASA-NNX17AD02G. Sebastian Grimberg acknowledges support by a Stanford Graduate Fellowship. Parts of this work were completed at the JPL, California Institute of Technology, under a contract with NASA.

NOMENCLATURE

t	time
$\bullet(t)$	designates a time-dependent quantity
$\bullet^{(n)}$	superscript designating the evaluation of a quantity at the time instance $t^{(n)}$
Γ_w	embedded surface
\mathcal{T}	Computational Fluid Dynamics (CFD) mesh
\mathcal{T}^i	i -th subdomain of \mathcal{T} when it is partitioned into subdomains
T	an element of the CFD mesh \mathcal{T}
n	a vertex of the CFD mesh \mathcal{T}
h	characteristic edge length of an element T in \mathcal{T}
\mathcal{E}	set of all edges of the mesh \mathcal{T}
\mathcal{E}_r	set of edges marked to be refined
\mathcal{E}_c	set of edges marked to be coarsened
\mathcal{E}_s	set of edges that should not be coarsened but may be refined
\mathcal{E}_b	set of edges located at the boundary of the domain
Z_i	inner zone of the CFD mesh
Z_t	transition zone of the CFD mesh
Z_{t_j}	j -th subzone in the transition zone
m	number of transition subzones
r	ratio of the geometric progression of a meshing procedure
d_z	distance to Γ_w of the boundary surface delimiting a zone Z_z that is the closest to the wall boundary
D_z	distance to Γ_w of the boundary surface delimiting a zone Z_z that is farthest to the wall boundary
S_{it}	surface defining the outer boundary of the inner zone and inner boundary of the transition zone
S_{to}	surface defining the inner boundary of the outer zone and outer boundary of the transition zone
h_i	maximum edge length in the inner zone
h_o	minimum edge length in the outer zone
h_t	characteristic edge length in the transition zone (potentially spatially varying)
y_1^+	dimensionless distance to the wall of the first layer of vertices away from the wall boundary
y_1^{+*}	user-specified value of y_1^+ in the mesh
$y_{1\min}^{+*}$	user-specified lower bound for y_1^{+*}
$y_{1\max}^{+*}$	user-specified upper bound for y_1^{+*}
u	exact solution

u_h	approximation of the exact solution
e_h	approximation error
\mathcal{E}_h	interpolation error
Π_h	interpolation operator
u_q	scalar quantity of interest
H_{u_q}	Hessian of u_q
e_h^{\max}	upper bound for the approximation error (refinement threshold for the Hessian criterion)
e_h^{\min}	lower bound for the approximation error (coarsening threshold for the Hessian criterion)
\mathcal{I}	transfer operator to the adapted mesh of the solution computed on the initial mesh
R	refinement edge of an element of the CFD mesh
s_i	i -th child of an element of the CFD mesh
\mathcal{T}_e	edge patch of an edge e of the CFD mesh
$N(\mathcal{T}, T)$	set of elements belonging to the CFD mesh \mathcal{T} containing $R(T)$
\mathcal{N}	set of vertices of the CFD mesh whose connected edges are marked for coarsening
$\hat{\mathcal{N}}_b$	set of vertices located on the boundary of the computational domain and whose connected edges are marked for coarsening
B	bounding box for a vertex of the CFD mesh
\vec{v}	velocity vector of the flow
ρ	density of the flow
p	pressure of the flow
μ	dynamic viscosity of the flow
M	Mach number of the flow
Re	Reynolds number of the flow
N_f	number of vertices in the CFD mesh
N_s	number of vertices in the discrete representation of an embedded surface
M_∞	free-stream Mach number
α_∞	free-stream angle of attack

ORCID

Sebastian Grimberg  <https://orcid.org/0000-0002-9188-0562>

Charbel Farhat  <https://orcid.org/0000-0003-2563-8820>

REFERENCES

1. Sengupta A, Roeder J, Kelsch R, Wernet M, Kandis M, Witkowski A. Supersonic disk gap band parachute performance in the wake of a viking-type entry vehicle from mach 2 to 2.5. Paper presented at: AIAA Atmospheric Flight Mechanics Conference and Exhibit; 2008; Honolulu, HI.
2. Karagiozis K, Kamakoti R, Cirak F, Pantano C. A computational study of supersonic disk-gap-band parachutes using large-eddy simulation coupled to a structural membrane. *J Fluids Struct.* 2011;27(2):175-192.
3. Huang Z, Avery P, Farhat C, Rabinovitch J, Derkevorkian A, Peterson LD. Simulation of parachute inflation dynamics using an Eulerian computational framework for fluid-structure interfaces evolving in high-speed turbulent flows. Paper presented at: 2018 AIAA Aerospace Sciences Meeting; 2018; Kissimmee, FL.
4. Wu P, Ifju P, Stanford B, et al. A multidisciplinary experimental study of flapping wing aeroelasticity in thrust production. Paper presented at: 50th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference; 2009; Palm Springs, CA.
5. Shyy W, Aono H, Chimakurthi SK, et al. Recent progress in flapping wing aerodynamics and aeroelasticity. *Prog Aerosp Sci.* 2010;46(7):284-327.
6. 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. *Comput Fluids.* 2003;32(1):3-29.
7. Geuzaine P, Brown G, Harris C, Farhat C. Aeroelastic dynamic analysis of a full F-16 configuration for various flight conditions. *AIAA J.* 2003;41(3):363-371.
8. Chen X, Zha GC, Yang MT. Numerical simulation of 3-D wing flutter with fully coupled fluid-structural interaction. *Comput Fluids.* 2007;36(5):856-867.
9. Maire PH, Abgrall R, Breil J, Ovadia J. A cell-centered Lagrangian scheme for two-dimensional compressible flow problems. *SIAM J Sci Comput.* 2007;29:1781-1824.

10. Georges G, Breil J, Maire PH. A 3D GCL compatible cell-centered Lagrangian scheme for solving gas dynamics equations. *J Comput Phys*. 2016;305:921-941.
11. Hirt C, Amsden AA, Cook J. An arbitrary Lagrangian-Eulerian computing method for all flow speeds. *J Comput Phys*. 1974;14(3):227-253.
12. Donea J, Giuliani S, Halleux JP. An arbitrary Lagrangian-Eulerian finite element method for transient dynamic fluid-structure interactions. *Comput Methods Appl Mech Eng*. 1982;33(1-3):689-723.
13. Farhat C, Geuzaine P, Grandmont C. The discrete geometric conservation law and the nonlinear stability of ALE schemes for the solution of flow problems on moving grids. *J Comput Phys*. 2001;174(2):669-694.
14. Geuzaine P, Grandmont C, Farhat C. Design and analysis of ALE schemes with provable second-order time-accuracy for inviscid and viscous flow simulations. *J Comput Phys*. 2003;191(1):206-227.
15. Farhat C, Geuzaine P. Design and analysis of robust ALE time-integrators for the solution of unsteady flow problems on moving grids. *Comput Methods Appl Mech Eng*. 2004;193(39-41):4073-4095.
16. Peskin CS. Numerical analysis of blood flow in the heart. *J Comput Phys*. 1977;25(3):220-252.
17. Colella P, Graves DT, Keen BJ, Modiano D. A Cartesian grid embedded boundary method for hyperbolic conservation laws. *J Comput Phys*. 2006;211(1):347-366.
18. Mittal R, Iaccarino G. Immersed boundary methods. *Annu Rev Fluid Mech*. 2005;37(1):239-261.
19. Farhat C, Degand C, Koobus B, Lesoinne M. Torsional springs for two-dimensional dynamic unstructured fluid meshes. *Comput Methods Appl Mech Eng*. 1998;163(1-4):231-245.
20. Li R, Tang T, Zhang P. Moving mesh methods in multiple dimensions based on harmonic maps. *J Comput Phys*. 2001;170(2):562-588.
21. Degand C, Farhat C. A three-dimensional torsional spring analogy method for unstructured dynamic meshes. *Comput Struct*. 2002;80(3-4):305-316.
22. Rendall T, Allen C. Unified fluid-structure interpolation and mesh motion using radial basis functions. *Int J Numer Methods Eng*. 2008;74(10):1519-1559.
23. Wang K, Lea P, Farhat C. A computational framework for the simulation of high-speed multi-material fluid-structure interaction problems with dynamic fracture. *Int J Numer Methods Eng*. 2015;104(7):585-623.
24. Bridson R, Fedkiw R, Anderson J. Robust treatment of collisions, contact and friction for cloth animation. Paper presented at: ACM SIGGRAPH 2005 Courses (SIGGRAPH); 2005; Los Angeles, CA.
25. Wang K, Gr  tarsson J, Main A, Farhat C. Computational algorithms for tracking dynamic fluid-structure interfaces in embedded boundary methods. *Int J Numer Meth Fluids*. 2012;70(4):515-535.
26. Farhat C, Lakshminarayan VK. An ALE formulation of embedded boundary methods for tracking boundary layers in turbulent fluid-structure interaction problems. *J Comput Phys*. 2014;263:53-70.
27. Berger MJ. *Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations*. [PhD thesis]. Stanford, CA: Department of Computer Science, Stanford University; 1982.
28. Berger MJ, Colella P. Local adaptive mesh refinement for shock hydrodynamics. *J Comput Phys*. 1989;82(1):64-84.
29. Hornung RD, Kohn SR. Managing application complexity in the SAMRAI object-oriented framework. *Concurrency Computat Pract Exper*. 2002;14(5):347-368.
30. Vanella M, Posa A, Balaras E. Adaptive mesh refinement for immersed boundary methods. *J Fluids Eng*. 2014;136(4):040909.
31. Hachem E, Feghali S, Codina R, Coupez T. Immersed stress method for fluid-structure interaction using anisotropic mesh adaptation. *Int J Numer Methods Eng*. 2013;94(9):805-825.
32. Abgrall R, Beaugendre H, Dobrzynski C. An immersed boundary method using unstructured anisotropic mesh adaptation combined with level-sets and penalization techniques. *J Comput Phys*. 2014;257:83-101.
33. Borker R, Grimberg S, Avery P, Farhat C, Rabinovitch J. An adaptive mesh refinement concept for viscous fluid-structure computations using Eulerian vertex-based finite volume methods. Paper presented at: 2018 AIAA Aerospace Sciences Meeting; 2018; Kissimmee, FL.
34. Mitchell WF. *Unified Multilevel Adaptive Finite Element Methods for Elliptic Problems* [PhD thesis]. Urbana, IL: University of Illinois at Urbana-Champaign; 1988.
35. Rivara M-C. Mesh refinement processes based on the generalized bisection of simplices. *SIAM J Numer Anal*. 1984;21(3):604-613.
36. B  nsch E. Local mesh refinement in 2 and 3 dimensions. *IMPACT Comput Sci Eng*. 1991;3(3):181-191.
37. Kossaczky I. A recursive approach to local mesh refinement in two and three dimensions. *J Comput Appl Math*. 1994;55(3):275-288.
38. Maubach JM. Local bisection refinement for N -simplicial grids generated by reflection. *SIAM J Sci Comput*. 1995;16(1):210-227.
39. Traxler CT. An algorithm for adaptive mesh refinement in n dimensions. *Computing*. 1997;59(2):115-137.
40. Arnold DN, Mukherjee A, Pouly L. Locally adapted tetrahedral meshes using bisection. *SIAM J Sci Comput*. 2000;22(2):431-448.
41. Stevenson R. The completion of locally refined simplicial partitions created by bisection. *Math Comput*. 2008;77(261):227-241.
42. Zhang LB. A parallel algorithm for adaptive local refinement of tetrahedral meshes using bisection. *Numer Math Theory Methods Appl*. 2009;2:65-89.
43. 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. *J Comput Phys*. 2008;227(16):7674-7700.
44. Wang K, Rallu A, Gerbeau JF, Farhat C. Algorithms for interface treatment and load computation in embedded boundary methods for fluid and fluid-structure interaction problems. *Int J Numer Meth Fluids*. 2011;67(9):1175-1206.

45. Farhat C, Wang KG, Main A, et al. Dynamic implosion of underwater cylindrical shells: experiments and computations. *Int J Solids Struct*. 2013;50(19):2943-2961.
46. Lakshminarayan V, Farhat C, Main A. An embedded boundary framework for compressible turbulent flow and fluid-structure computations on structured and unstructured grids. *Int J Numer Meth Fluids*. 2014;76(6):366-395.
47. Huang DZ, De Santis D, Farhat C. A family of position- and orientation-independent embedded boundary methods for viscous flow and fluid-structure interaction problems. *J Comput Phys*. 2018;365:74-104.
48. Bartels S, Schreier P. Local coarsening of simplicial finite element meshes generated by bisections. *BIT Numer Math*. 2012;52(3):559-569.
49. C  a J. Approximation variationnelle des probl  mes aux limites. *Ann Inst Fourier*. 1964;14(2):345-444.
50. Wong ML, Lele SK. Multiresolution feature detection in adaptive mesh refinement with high-order shock- and interface-capturing scheme. Paper presented at: 46th AIAA Fluid Dynamics Conference; 2016; Washington, DC.
51. L  hner R. Adaptive mesh refinement. In: *Applied Computational Fluid Dynamics Techniques: An Introduction Based on Finite Element Methods*. 2nd ed. Chichester, UK: John Wiley and Sons Ltd; 2001:269-297.
52. Peraire J, Vahdati M, Morgan K, Zienkiewicz OC. Adaptive remeshing for compressible flow computations. *J Comput Phys*. 1987;72(2):449-466.
53. Frey PJ, Alauzet F. Anisotropic mesh adaptation for CFD computations. *Comput Methods Appl Mech Eng*. 2005;194(48-49):5068-5082.
54. Ragusa JC. A simple Hessian-based 3D mesh adaptation technique with applications to the multigroup diffusion equations. *Ann Nucl Energy*. 2008;35(11):2006-2018.
55. Kahn AB. Topological sorting of large networks. *Commun ACM*. 1962;5(11):558-562.
56. Alk  mper M, Kl  fkor  n R. Distributed newest vertex bisection. *J Parallel Distributed Comput*. 2017;104:1-11.
57. Chen L, Zhang C. A coarsening algorithm on adaptive grids by newest vertex bisection and its applications. *J Comput Math*. 2010;28:767-789.
58. Spalart PR, Allmaras SR. A one-equation turbulence model for aerodynamic flows. *La Recherche A  rosp*. 1994;5-21.
59. Sethian JA. A fast marching methods. *SIAM Review*. 1999;41(2):199-235.
60. Zhao H. A fast sweeping method for eikonal equations. *Math Comput*. 2005;74(250):603-627.
61. Jeong WK, Whitaker RT. A fast iterative method for eikonal equations. *SIAM J Sci Comput*. 2008;30(5):2512-2534.
62. Grimberg S, Farhat C. Fast computation of the wall distance in unsteady Eulerian fluid-structure computations. *Int J Numer Meth Fluids*. 2019;89:143-161. <https://doi.org/10.1002/fld.4686>
63. Farhat C, Rallu A, Wang K, Belytschko T. Robust and provably second-order explicit-explicit and implicit-explicit staggered time-integrators for highly non-linear compressible fluid-structure interaction problems. *Int J Numer Methods Eng*. 2010;84(1):73-107.
64. L  hner R. *Applied Computational Fluid Dynamics Techniques: An Introduction Based on Finite Element Methods*. Chichester, UK: John Wiley and Sons Ltd; 2008.
65. Karypis G. METIS and ParMETIS. In: *Encyclopedia of Parallel Computing*. New York, NY: Springer Science+Business Media; 2011:1117-623.
66. Sod GA. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *J Comput Phys*. 1978;27(1):1-31.
67. Brogniez S, Rajasekharan A, Farhat C. Provably stable and time-accurate extensions of Runge-Kutta schemes for CFD computations on moving grids. *Int J Numer Meth Fluids*. 2012;69(7):1249-1270.
68. Geuzaine C, Remacle JF. Gmsh: a 3-D finite element mesh generator with built-in pre- and post-processing facilities. *Int J Numer Methods Eng*. 2009;79(11):1309-1331.
69. Hida K. An approximate study on the detached shock wave in front of a circular cylinder and a sphere. *J Phys Soc Jpn*. 1953;8(6):740-745.
70. Heberle JW, Wood GP, Gooderum PB. *Data on Shape and Location of Detached Shock Waves on Cones and Spheres*. Technical note. Washington, DC: National Aeronautics and Space Administration; 1950.
71. Nagata T, Nonomura T, Takahashi S, Mizuno Y, Fukuda K. Investigation on subsonic to supersonic flow around a sphere at low Reynolds number of between 50 and 300 by direct numerical simulation. *Phys Fluids*. 2016;28(5). 056101.
72. Lin JK, Shook LS, Ware JS, Welch JV. *Flexible Material Systems Testing*. Technical report. Hampton, VA: NASA; 2010.
73. Militello C, Felippa CA. The first ANDES elements: 9-dof plate bending triangles. *Comput Methods Appl Mech Eng*. 1991;93(2):217-246.
74. Alk  mper M, Gaspoz F, Kl  fkor  n R. A weak compatibility condition for newest vertex bisection in any dimension. arXiv preprint arXiv:1711.03141. 2017.

How to cite this article: Borker R, Huang D, Grimberg S, Farhat C, Avery P, Rabinovitch J. Mesh adaptation framework for embedded boundary methods for computational fluid dynamics and fluid-structure interaction. *Int J Numer Meth Fluids*. 2019;90:389-424. <https://doi.org/10.1002/fld.4728>

APPENDIX A

NUMBERING AND TAGGING OF AN INITIAL MESH

Let $T = (\alpha_0, \alpha_1, \alpha_2, \alpha_3)_\gamma$ be a tetrahedron of an initial tetrahedral mesh \mathcal{T} :

- The element T_R defined by $T_R = (\alpha_3, \alpha_2, \alpha_1, \alpha_0)_\gamma$, and therefore by a particular reordering of T , is called the *reflected element* of T .
- Two neighboring elements T and T' are said to be *reflected neighbors* if the vertex ordering of T or that of T_R coincides with the vertex ordering of T' , except for one position in the sequence of the α_i s.

Then, as noted in Section 4.4.1, the finite termination of Algorithm 5 is guaranteed if the ordering and tagging of \mathcal{T} satisfy certain conditions. Specifically, these conditions are:

- All elements of \mathcal{T} have the same type γ .
- Two neighboring elements $T = (\alpha_0, \alpha_1, \alpha_2, \alpha_3)$ and $T' = (\alpha'_0, \alpha'_1, \alpha'_2, \alpha'_3)$ sharing a face S are reflected neighbors if the refinement edge $\overline{\alpha_0\alpha_3} \in S$, or the refinement edge $\overline{\alpha'_0\alpha'_3} \in S$; otherwise, the pair of neighboring children of T and T' are reflected neighbors.

Various approaches for generating an initial mesh that satisfies the above conditions have been proposed in the literature.^{37,38,41} Three of them are overviewed below.

APPENDIX B

KUHN'S SIMPLEX METHOD

The Kuhn triangulation, also known as the Kuhn decomposition, is a decomposition of a cube into six tetrahedral elements (see Figure B1). Assuming, without any loss of generality, that the cube is $[0, 1]^3$, the six tetrahedra are:

$$\begin{aligned}
 T_{(1,2,3)} &= \{(0, 0, 0), (1, 0, 0), (1, 1, 0), (1, 1, 1)\} \\
 T_{(1,3,2)} &= \{(0, 0, 0), (1, 0, 0), (1, 0, 1), (1, 1, 1)\} \\
 T_{(2,1,3)} &= \{(0, 0, 0), (0, 1, 0), (1, 1, 0), (1, 1, 1)\} \\
 T_{(3,1,2)} &= \{(0, 0, 0), (0, 1, 0), (0, 1, 1), (1, 1, 1)\} \\
 T_{(3,2,1)} &= \{(0, 0, 0), (0, 0, 1), (0, 1, 1), (1, 1, 1)\} \\
 T_{(2,3,1)} &= \{(0, 0, 0), (0, 0, 1), (1, 0, 1), (1, 1, 1)\}.
 \end{aligned} \tag{B1}$$

A tetrahedral mesh satisfying the conditions for finite termination of Algorithm 5 overviewed above can be obtained by first dividing the initial computational domain into a structured mesh of cubical elements, then splitting each element into Kuhn tetrahedra defined by the ordering given in Equation (B1) and $\gamma = 0$. Although this approach is limited to simple geometries such as hexahedral domains, such geometries are not uncommon in the context of EBM.

Next, two alternative methods that are applicable to arbitrary geometries are overviewed.

B.1 | Prerefinement method

Given any triangulation in three dimensions, each tetrahedron $T = (\alpha_0, \alpha_1, \alpha_2, \alpha_3)$ may be subdivided into four tetrahedra by inserting a new vertex P at the center of gravity of T . Then, each of these four tetrahedra can be divided again into three tetrahedra by inserting a vertex Q at the center of gravity of each face of the original tetrahedron. Each of the resulting 12 tetrahedra consists of the vertex P , one vertex Q located at the center of gravity of a face of T , and two vertices from T (see Figure B2). The ordering of each child tetrahedron is as follows:

$$T(\alpha_i, \alpha_j, P, Q) = \begin{cases} (\alpha_i, P, Q, \alpha_j), & \text{if the global ID number of the vertex } \alpha_i \text{ is smaller than that of the vertex } \alpha_j. \\ (\alpha_j, P, Q, \alpha_i), & \text{if the global ID number of the vertex } \alpha_i \text{ is larger than that of the vertex } \alpha_j. \end{cases}$$

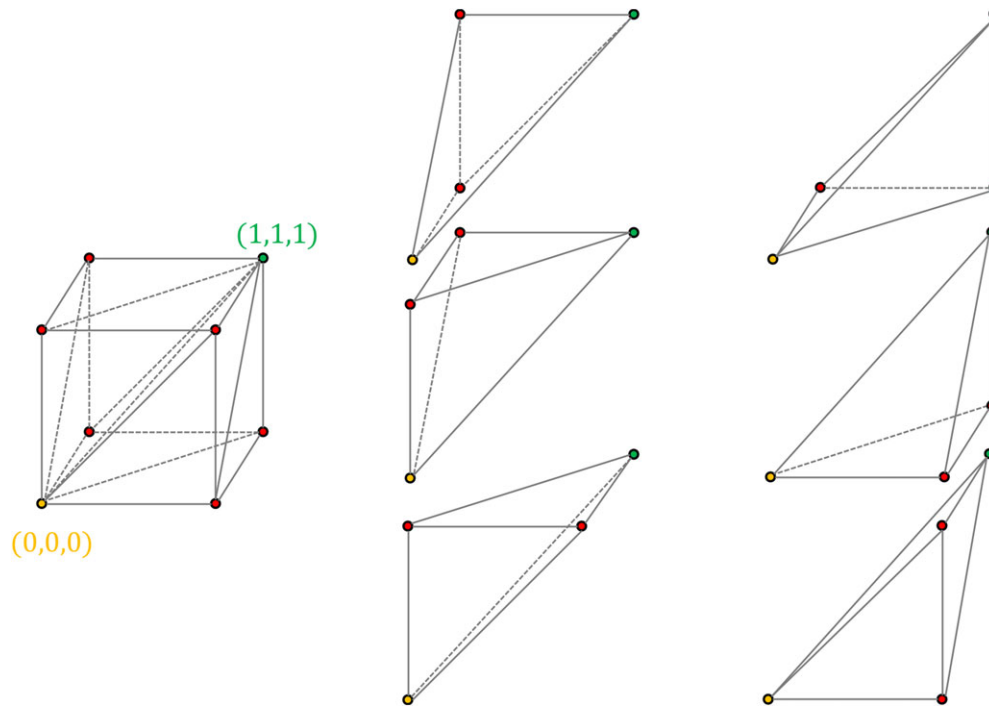


FIGURE B1 Kuhn's simplex method for ordering and tagging an initial mesh [Colour figure can be viewed at wileyonlinelibrary.com]

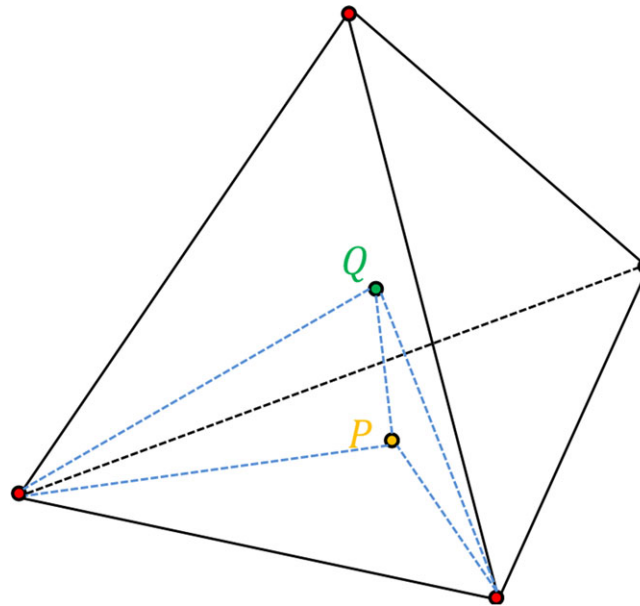


FIGURE B2 Prerefinement method for ordering and tagging an initial mesh [Colour figure can be viewed at wileyonlinelibrary.com]

If the type γ of all elements of an initial mesh constructed using a coarse triangulation of the computational domain of interest and the prerefinement and ordering approach described above is set to two, the resulting mesh satisfies the finite termination conditions stated above. Nevertheless, this approach for generating an initial mesh that satisfies these conditions has two main disadvantages: the aspect ratios of the elements in the prerefined mesh are higher than those of the elements of the original mesh; it increases the number of elements of the given triangulation by an order of magnitude, and therefore can constitute an excessive prerefinement.

An alternative approach for ordering and tagging an initial mesh that does not require pre-refinement but only reordering is described next.

B.2 | Reordering method

Alkämper et al⁷⁴ proposed a weak compatibility condition that is a sufficient condition for Algorithm 5 to terminate. They also proposed several reordering algorithms to achieve the weak compatibility condition for any initial triangulation. One of their algorithms is described below.

Algorithm 11 Reordering algorithm⁷⁴ for satisfying the conditions of termination of Algorithm 5

- 1: globally order all vertices of the initial triangulation
 - 2: for each tetrahedron T , set $\gamma = 0$ and sort its vertices in ascending order based on the global numbering of the tetrahedral mesh
-

In the above algorithm, the global numbering is arbitrary. However, it must be carefully chosen as it can affect the performance of the refinement algorithm and the quality of the refined mesh. Hence, the following greedy procedure is used in this work to address these issues:

- For each tetrahedron of the initial triangulation, locate its longest edge (if there are multiple longest edges of equal length, choose any one of them).
- Insert the vertices of each tetrahedron into the global numbering sequence. If possible, place the two vertices that are not on the longest edge into the sequence between the vertices of the longest edge.