# Mesh adaptation for the embedded boundary method in CFD

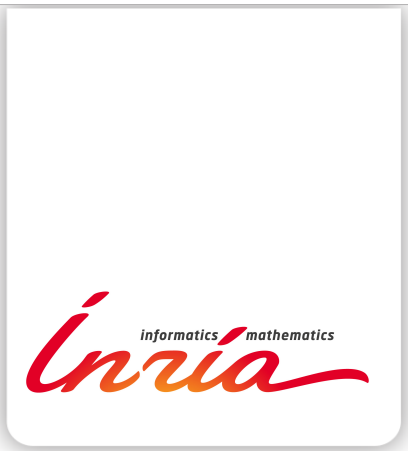
Rémi Feuillet, Adrien Loseille, Frédéric Alauzet


# Mesh adaptation for the embedded boundary method in CFD

**Rémi Feuillet, Adrien Loseille, Frédéric Alauzet**

**RESEARCH REPORT**
**N° 9305**
November 2019
Project-Team Gamma

ISSN 0249-6399 ISRN INRIA/RR--9305--FR+ENG




# Mesh adaptation for the embedded boundary method in CFD

Rémi Feuillet*, Adrien Loseille†, Frédéric Alauzet‡

Project-Team Gamma

Research Report n° 9305 — November 2019 — 58 pages

**Abstract:** The numerical simulation of complex physical phenomenons usually requires a mesh. In Computational Fluid Dynamics, it consists in representing an object inside a huge control volume. This object is then the subject of some physical study. In general, this object and its bounding box are represented by linear surface meshes and the intermediary zone is filled by a volume mesh. The aim of this report is to have a look on a way to represent the object. This approach, called embedded method, consists in fully meshing the bounding box volume without explicitly meshing the object in it. In this case, the presence of the object is implicitly simulated by the CFD solver. The coupling of this method with linear mesh adaptation is in particular discussed.

**Key-words:** Embedded methods, CFD, Mesh adaptation

* remi.feuillet@inria.fr
† adrien.loseille@inria.fr
‡ frederic.alauzet@inria.fr

RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

1 rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing
Campus de l'École Polytechnique
91120 Palaiseau

# Adaptation de maillage pour la méthode des frontières immergées en mécanique des fluides numérique

**Résumé :** La simulation numérique de phénomènes physiques complexes requiert généralement l'utilisation d'un maillage. En mécanique des fluides numérique, cela consiste à représenter un objet dans un gros volume de contrôle. Cet objet étant celui dont l'on souhaite simuler le comportement. Usuellement, l'objet et la boîte englobante sont représentés par des maillage de surface linéaires et la zone intermédiaire est remplie par un maillage volumique. L'objectif de ce rapport de recherche est de s'intéresser à une manière de représenter cet objet. Cette approche - dite immergée - consiste à mailler intégralement le volume de contrôle et ensuite à simuler le comportement autour de l'objet sans avoir à mailler explicitement dans le volume ladite géométrie. L'objet étant implicitement pris en compte par le schéma numérique. Le couplage de cette méthode avec de l'adaptation de maillage linéaire est notamment étudié.

**Mots-clés :** Méthodes immergées, Mécanique des fluides numérique, Adaptation de maillage

# Introduction

Computational Fluid Dynamics (CFD) is the art of using numerical analysis to study the fluid mechanics and to numerically solve problems that involve fluid flows. The physical phenomena are various as steady or unsteady flow that are modeled by the Euler or Navier-Stokes equations with applications to Fluid-Structure Interaction or Shape Optimization. While most of CFD techniques use a discretization mesh (or grid) to solve these physics, the choice of the nature of the discretization mesh remains an open question in the community. Structured, unstructured, isotropic, anisotropic, simplicial, non-simplicial or hybrid, ... the diversity of the used meshes is as big as the number of numerical schemes applied on it. Finally, a last question remains: how do we represent the object ? Three approaches mainly exist: the first one known as the body-fitted approach consists in exactly representing the object in the mesh. The second approach is the Chimera (or overset) method [12], in which the studied body has it own body-fitted sub-mesh, and the sub-mesh overlaps the global computational mesh. Finally, the last one is the embedded (or immersed) approach [44] where the object is implicitly represented (using a level-set function or a (discrete) CAD geometry) in the computational mesh.

In this report, we consider the last approach. The main interest is its ability perform numerical simulations on very complex objects without requiring as input a conformal mesh (*e.g.* a water-tight and non self-intersecting mesh) of the geometry. Actually, it would be almost impossible to obtain this mesh without manual intervention. This specific type of simulation is of particular interest for simulating shape-varying geometries like veins or aortic valves [47, 48], really complex geometries like helicopters or Formula One cars [14], or finally multi components bodies like in fragmentation cases [35, 36, 33, 46]. Another advantage relies in the simplicity of the generated volume mesh. As the constraint of the inner object is no longer there, the meshing of the volume part is not an issue and can be easily done, even in a structured fashion. However, following the law that there is constant balance between the work required in the solver and the mesh generation, it implies that the solver has more constraints to follow. Indeed, the solver has to be able to detect the boundary of the object, this boundary being represented by a set of edges or triangles, a CAD model or a level-set function, and to determine, if applicable, the inner and the outer part of the object. Then, the next step consists in applying the appropriate treatment to the elements of the mesh where it is intersected [53, 32, 22]. This step is the most crucial part of the process to get proper boundary conditions. Note that this approach has been developed for both compressible and incompressible Euler/Navier-Stokes discretized with either the Finite Volume Method [54, 55], the Finite Element Method [35] or a Discontinuous Galerkin Method [22].

In the case of simplicial (*e.g.* triangular and tetrahedral) meshing, the resolution of the flow solver is usually coupled with mesh adaptation. There exists mainly three type of adaptation: *r-adaptation* which consists in moving the vertices [30], *p-adaptation* which consists in locally enriching the functional space in the context of Finite Element Methods [8] and finally *h-adaptation* which consists in locally refining the discretization grid in a non-conformal [13] or conformal way [34]. The last approach appears to be interesting as it does not require to change the used solver (which, in our case, requires a conformal mesh) while highly refining the mesh if needed. In the context of body-fitted simulations, mesh adaptation has been successfully applied to various cases like Euler equations in the case of both steady [49, 37] and unsteady flows [2, 21] with possibility of moving geometries [42, 10, 26] and Fluid-Structure Interaction [52]. These methods have been extended for the steady case to the Navier-Stokes equations [41, 24] in the context of RANS simulations. Of course, other types of application can be done using mesh

adaptation like acoustics [17], electromagnetics [15], magnetohydrodynamics (MHD) [6] or solid mechanics [16] among many others. The coupling of mesh adaptation with the embedded boundary method has also been tackled in the case of Navier-Stokes equations [47, 48, 1] and more generally for non-simplicial meshes, the method is usually coupled with adaptive mesh refinement on cartesian grids [51].

In this report, we present the use of anisotropic mesh adaptation for the embedded boundary method applied to the Euler Equations. To this end, the first section deals with the basic principles used in mesh adaptation for the Euler equations and details the methodology used for the resolution of the Euler equations in `Wolf`. Then, the second section explains the approach used to implement the Embedded Boundary Method in the already existing resolution framework and shows some result that validate the implementation of the approach. Finally, the coupling between the method and mesh adaptation is performed.

# 1 Mesh adaptation for the Euler equations

The purpose of this section is to deal with mesh adaptation for Euler equations that govern adiabatic and inviscid flow. In the case of compressible flow, the Euler equations are conveniently solved by the Finite Volume Method. In the following, the implementation of the Finite Volume Method in the solver `Wolf` [4] is presented. It consists in a vertex-centered finite volume formulation where the finite volume cells are implicitly built on unstructured meshes. Second-order space accuracy is achieved through a piecewise-linear extrapolation based on the Monotonic Upwind Scheme for Conservation Law (MUSCL) procedure with a particular edge-based formulation. Both explicit and implicit approaches are available for the time integration. During implicit simulations, a linearized system is solved at each solver iteration using a Lower-Upper Symmetric Gauss-Seidel (LU-SGS) approach.
The general idea of anisotropic mesh adaptation is to modify the discretization of the computational domain so that it minimizes the error induced by the discretization. Some regions of the mesh are refined while others are coarsened, and stretched mesh elements are generated to follow the natural anisotropy of the physical phenomena. In general, the approximation error $(u - u_h)$ is decomposed into two types of error: (i) the interpolation error $(u - \Pi_h u)$, (ii) the implicit error $(\Pi_h u - u_h)$ , where $u$ is the exact solution, $u_h$ the numerical solution provided by the flow solver and $\Pi_h u$ the linear interpolate of $u$ on the discretization. In this chapter, we focus on controlling the interpolation error. It is used as starting point for the construction of proper size maps for the mesh where these sizes are based on the hessian of the solution. Then, the mesh adaptation is performed using the anisotropic local remesher `Feflo.a/AMG` [40].

## 1.1 The Finite Volume method for the Euler equations

Several methods exist to solve the Euler equations. In this section, we detail a vertex-centered Finite Volume Method. This type of numerical scheme is specifically tailored for the resolution of hyperbolic systems to whom the Euler equations are part of.

### 1.1.1 The Euler equations

The Euler equations in $\mathbb{R}^d_{(d=2,3)}$ read:

$$\begin{cases} \dfrac{\partial}{\partial t} W + \nabla \cdot (\mathbf{F}(W)) = 0, \quad \forall (\mathbf{x}, t) \in \Omega \times \mathbb{R}^+, \\ W(\mathbf{x}, 0) = W_0(\mathbf{x}), \\ W(\mathbf{x}, 0) = W_\infty(\mathbf{x}, t), \quad \forall (\mathbf{x}, t) \in \partial\Omega \times \mathbb{R}^+. \end{cases}$$

$\Omega$ is an open bounded domain of $\mathbb{R}^d$ of boundary $\partial\Omega = \Gamma$, $W$ is a vector function of $\mathbb{R}^{d+2}$ and $\mathbf{F}$ is an element $\mathbb{R}^{d+2} \times \mathbb{R}^{2d}$.
$W$ is the state variable and is defined by:

$$W = \begin{pmatrix} \rho \\ \rho \mathbf{U} \\ \rho E \end{pmatrix},$$

where $\rho$ denotes the density, $\mathbf{U} \in \mathbb{R}^d$ the fluid velocity and $E$ the total energy.
$\mathbf{F}(W) = (F_i(W))_{1 \leq i \leq d}$ with:

$$F_i(W) = \begin{pmatrix} \rho u_i \\ \rho u_i \mathbf{U} + p\delta_i \\ (\rho E + p) u_i \end{pmatrix},$$

where $\delta_i$ denotes the $i^{th}$ column vector of the identity matrix $I_d$ and $p$ the pressure, also satisfying the perfect gas equation:

$$p = (\gamma - 1)(\rho T),$$

where $\gamma$ is the ratio of the specific heats and is supposed constant. Note that following the International System of units, the pressure $p$ is in Pascals (Pa), the velocity $\mathbf{u}$ in meters per second ($\mathrm{m.s}^{-1}$), the density $\rho$ in kilograms per square ($d = 2$) or cube ($d = 3$) meters ($\mathrm{kg.m}^{-d}$) and the energy $E$ is in Joules (J).

### 1.1.2 Spatial discretization

The spatial discretization of the Euler equations relies on the Finite Volume method (a more complete survey of this method can be found in [41, 24], notably for the case of RANS equations). Let $\mathcal{H}$ be a mesh of domain $\Omega$, the vertex-centered finite volume formulation consists in associating with each vertex $P_i$ of the mesh a control volume or finite volume cell $C_i$ (Figure 1).

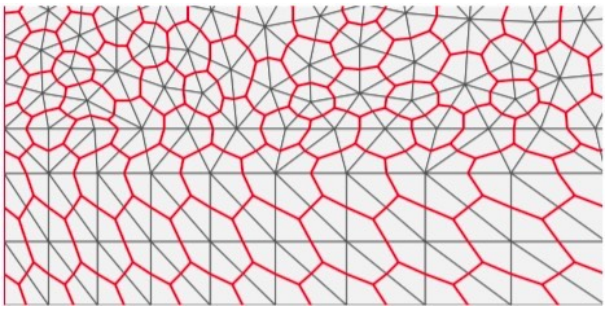


Figure 1: Median Finite Volume cells.

Discretized domain $\Omega_h$ can be written as:

$$\Omega_h = \bigcup_{i=1}^{N_K} K_i = \bigcup_{i=1}^{N_V} C_i,$$

where $N_K$ is the number of elements and $N_V$ the number of vertices. Note that unlike a cell-centered approach, the finite volume cells are computed on the fly and never stored. The only stored data is the input simplicial mesh.
The integration of the Euler equation for each cell $C_i$ (using Green Formula), gives:

$$|C_i| \frac{\mathrm{d}W_i}{\mathrm{d}t} + \mathbf{F}_i = 0,$$

where $W_i$ is the mean value of $W$ on cell $C_i$ and $\mathbf{F}_i$ is the numerical convective flux *e.g.*:

$$\mathbf{F}_i = \int_{\partial C_i} \mathbf{F}(W_i).\mathbf{n}_i \mathrm{d}\gamma,$$

where $\mathbf{n}_i$ is the outer normal to the finite volume cell surface $\partial C_i$.

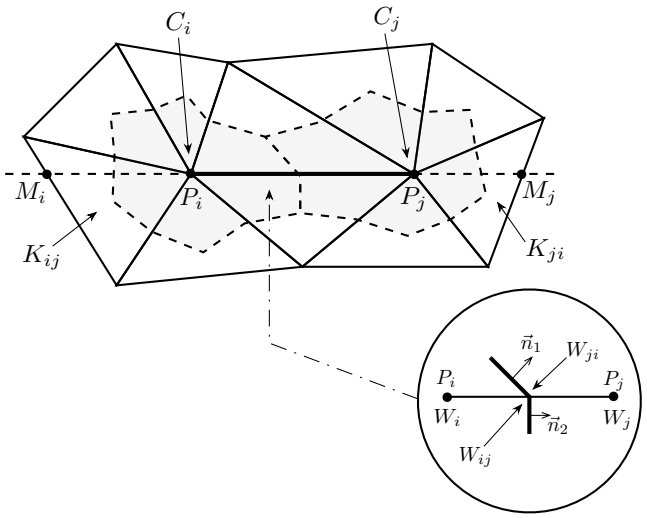



Figure 2: Construction of a cell and interface between two cells.

Now, if we note $\partial C_{i,j} = \partial C_i \cup \partial C_j$ (cf Figure 2), the integration of the numerical flux writes:

$$\mathbf{F}_i = \sum_{P_j \in \nu(P_i)} F_{|\partial C_{i,j}} \cdot \int_{\partial C_{i,j}} \mathbf{n}_i \mathrm{d}\gamma, \tag{1}$$

where $\nu(P_i)$ is the set of all neighboring vertices linked by an edge to $P_i$ and $F_{|\partial C_{i,j}}$ represents the constant value of $F(W)$ at the interface $\partial C_{i,j}$. It is then computed using a numerical flux function, denoted by $\Phi_{ij}$:

$$\Phi_{ij} = \Phi_{ij}(W_i, W_j, \mathbf{n}_{ij}) = F_{|\partial C_{i,j}} \cdot \int_{\partial C_{i,j}} \mathbf{n}_i \mathrm{d}\gamma, \tag{2}$$

where $\mathbf{n}_{ij} = \displaystyle\int_{\partial C_{i,j}} \mathbf{n}_i \mathrm{d}\gamma$. The purpose of approximated Riemann solvers is to compute this term. We employ the HLLC approximate Riemann solver [11].

### 1.1.3 HLLC approximate Riemann solver

The concept behind the HLLC flow solver [11] is to locally consider an approximated Riemann problem with two intermediate states depending on the local left and right states, Figure 3 .

The simplified solution to the Riemann problem consists of a contact wave with a velocity $S_M$ and two acoustic waves, which may be either shocks or expansion fans. The acoustic waves have the smallest and the largest velocities ($S_L$ and $S_R$ respectively) of all the waves present in the exact solution. if $S_L > 0$, then the flow is supersonic from left to right and the upwind flux is simply defined from $F(W_l)$ where $W_l$ is the state to the left of the discontinuity. Similarly, if $S_R < 0$, then the flow is supersonic from right to left and the flux is defined form $F(W_r)$

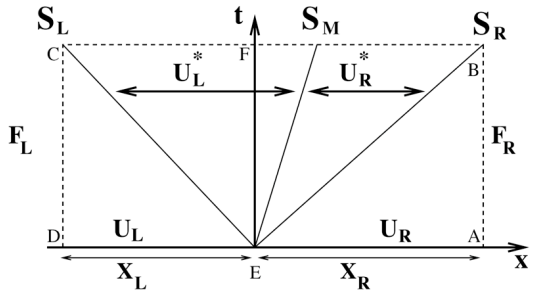



Figure 3: Simplified Riemann problem used for the computation of HLLC flux.

where $W_r$ is the state to the right of the discontinuity. In the more difficult subsonic case when $S_L < 0 < S_R$, we have to compute $F(W_l^*)$ or $F(W_r^*)$. Consequently, the HLLC flux is given by :

$$\Phi_{lr}^{hllc}(W_l, W_r, \mathbf{n}_{lr}) = \begin{cases} F(W_l).\mathbf{n}_{lr} & \text{if} \quad S_L > 0, \\ F(W_l^*).\mathbf{n}_{lr} & \text{if} \quad S_L \leq 0 < S_M, \\ F(W_r^*).\mathbf{n}_{lr} & \text{if} \quad S_M \leq 0 \leq S_R, \\ F(W_r).\mathbf{n}_{lr} & \text{if} \quad S_R < 0. \end{cases}$$

$W_l^*$ and $W_r^*$ are evaluated as follows. We denote by $\eta = \mathbf{u} \cdot \mathbf{n}$. Assuming (for simplification) that $\eta^* = \eta_l^* = \eta_r^* = S_M$, the following evaluations are proposed (the subscript $l$ or $r$ are omitted for clarity):

$$W^* = \frac{1}{S - S_M} \begin{pmatrix} \rho(S-\eta) \\ \rho\mathbf{u}(S-\eta) + (p^* - p)\mathbf{n} \\ \rho E(S-\eta) + p^* S_M - p\eta \end{pmatrix}, \qquad \text{with} \quad p^* = \rho(S-\eta)(S_M - \eta) + p.$$

A key feature of this solver is in the definition of the three waves velocity. For the contact wave, we consider

$$\begin{aligned} S_M &= \frac{\rho_r \eta_r (S_R - \eta_r) \rho_l \eta_l (S_L - \eta_l) + p_l - p_r}{\rho_r (S_R - \eta_r) - \rho_l (S_L - \eta_l)}, \\ S_L &= \min(\eta_l - c_l, \tilde{\eta} - \tilde{c}), \\ S_R &= \min(\eta_r + c_r, \tilde{\eta} + \tilde{c}), \end{aligned}$$

where $\sim$ denotes Roe average values [11]. With such waves velocities, the approximate HLLC Riemann solver has the following properties : (i) it satisfies the entropy inequality, (ii) it resolves isolated contacts exactly, (iii) it resolves isolated shocks exactly and (iv) it preserves positivity.

### 1.1.4 Second order accurate version

The Finite Volume Method as is, is a first order scheme. The MUSCL type reconstruction scheme [50] has been designed to increase the order of accuracy of the scheme. An extension of this approach to unstructured meshes has then been proposed in [20].

To this end, we use extrapolated values $W_{ij}$ and $W_{ji}$ (Fig.4) instead of $W_i$ and $W_j$ at the interface $\partial C_{i,j}$ to evaluate the flux. The numerical flux becomes:

$$\Phi_{ij} = \Phi_{ij}(W_{ij}, W_{ji}, \mathbf{n}_{ij}),$$

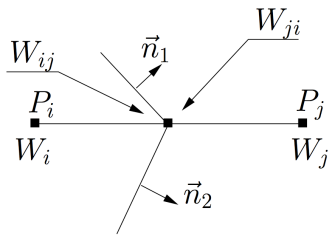



Figure 4: Extrapolation used for the MUSCL scheme.

with :

$$W_{ij} = W_i + \frac{1}{2}(\nabla W)_{ij} \cdot \overrightarrow{P_iP_j} \quad \text{and} \quad W_{ji} = W_j + \frac{1}{2}(\nabla W)_{ji} \cdot \overrightarrow{P_jP_i}. \tag{3}$$

The computation of gradients $(\nabla W)_{ij}$ and $(\nabla W)_{ji}$ is obtained using a combination of centered, upwind and nodal gradients.
The centered gradient is implicitly given by the relation:

$$(\nabla W)^C_{ij} \cdot \overrightarrow{P_iP_j} = W_j - W_i,$$

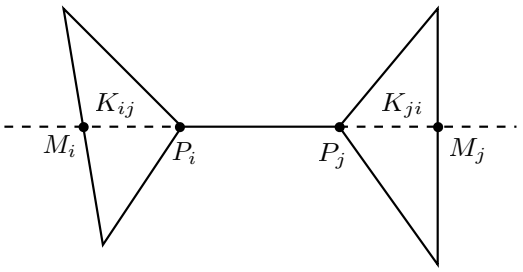



Figure 5: Upwind and downwind elements used for the MUSCL scheme.

Upwind and downwind gradients are computed according to the definition of upwind $(K_{ij})$ and downwind elements$(K_{ji})$, (Fig.5). These elements are the unique simplices of the ball of $P_i$ (resp. $P_j$) whose opposite facet to $P_i$ (resp. $P_j$) is crossed by the line defined by the edge $P_iP_j$ and that and do not contain $P_iP_j$.
Upwind and downwind gradients are then defines for vertices $P_i$ and $P_j$ as:

$$(\nabla W)^U_{ij} = (\nabla W)|_{K_{ij}} \quad \text{and} \quad (\nabla W)^D_{ij} = (\nabla W)|_{K_{ji}}$$

where $(\nabla W)|_K = \sum_{P\in K} W_P \nabla \phi_P|_K$ is the $P^1$-Galerkin gradient on the element K. Parameterized edge gradients are built by introducing the $\beta$-scheme:

$$(\nabla W)_{ij} \cdot \overrightarrow{P_iP_j} = (1-\beta)(\nabla W)^C_{ij} \cdot \overrightarrow{P_iP_j} + \beta(\nabla W)^U_{ij} \cdot \overrightarrow{P_iP_j}$$
$$(\nabla W)_{ji} \cdot \overrightarrow{P_jP_i} = (1-\beta)(\nabla W)^C_{ij} \cdot \overrightarrow{P_jP_i} + \beta(\nabla W)^D_{ij} \cdot \overrightarrow{P_jP_i}$$

where $\beta \in [0,1]$ is a parameter controlling the amount of upwinding. If $\beta = 0$, the scheme is centered and if $\beta = 1$ the scheme is fully upwind.

The most accurate $\beta$-scheme is obtained for $\beta = \frac{1}{3}$. On unstructured meshes, a second-order scheme with a fourth-order numerical dissipation is obtained. This scheme is denoted as the **V4-scheme**.
It is in fact possible to get an even less dissipative scheme. This scheme is denoted as the **V6-scheme**. It is a more complex linear combination of gradients using centered upwind and nodal gradients. The nodal $P^1$-Galerkin gradient of $P_i$ is related to cell $C_i$ and is computed by averaging the gradients of all the simplices containing the vertex $P_i$:

$$(\nabla W)_{P_i} = \frac{1}{(d+1)|C_i|} \sum_{K \in C_i} |K| (\nabla W)_K$$

A sixth-order dissipation scheme is then obtained by considering the following high-order gradient:

$$\begin{aligned}
(\nabla W)_{ij}^{V6} \cdot \overrightarrow{P_iP_j} &= [(\nabla W)_{ij}^{V4} \; - \; \frac{1}{30}\left((\nabla W)_{ij}^{U} - 2(\nabla W)_{ij}^{C} + (\nabla W)_{ij}^{D}\right) \\
&\quad - \frac{2}{15}\left((\nabla W)_{M_i} - 2(\nabla W)_{P_i} + (\nabla W)_{P_j}\right)] \cdot \overrightarrow{P_iP_j} \\
(\nabla W)_{ji}^{V6} \cdot \overrightarrow{P_jP_i} &= [(\nabla W)_{ji}^{V4} \; - \; \frac{1}{30}\left((\nabla W)_{ij}^{D} - 2(\nabla W)_{ij}^{C} + (\nabla W)_{ij}^{U}\right) \\
&\quad - \frac{2}{15}\left((\nabla W)_{M_j} - 2(\nabla W)_{P_j} + (\nabla W)_{P_i}\right)] \cdot \overrightarrow{P_jP_i}
\end{aligned} \tag{4}$$

where $(\nabla W)_{M_{i,j}}$ is the gradients at the points $M_{i,j}$, intersection of the line defined by $(P_iP_j)$ and upwind/downwind elements (see Fig. 5). These gradients are computed by linear interpolation of the nodal gradients of the facets containing $M_i$ and $M_j$.

**Limiter function.** MUSCL schemes are not monotone and can be a source of spurious oscillations. These oscillations can affect the accuracy of the final solution and its precision as it can crash the code by creating negative densities or pressures. One of the major concerns is to guarantee the LED (*Local Extremum Diminishing*) property of the scheme which ensures that the extrapolated values make sense. To this end, limiting functions are coupled with the previous high-order gradient evaluations. The gradient is substituted by a limited gradient denoted $(\nabla W)^{Lim}$. The choice of the limiting function is crucial as it directly affects the convergence and the accuracy of the solution. In this work, two limiters are used:

- Dervieux or Koren limiter [31]: This is a three entries limiter,

$$(\nabla W)_{ij}^{Lim}.\overrightarrow{P_iP_j} = Lim_{DE}((\nabla W)_{ij}^{C} \cdot \overrightarrow{P_iP_j} \; , \; (\nabla W)_{ij}^{D}.\overrightarrow{P_iP_j} \; , \; (\nabla W)_{ij}^{HO} \cdot \overrightarrow{P_iP_j}),$$

  with:

$$\begin{aligned}
&\text{if } uv \leq 0 \text{ then} \\
&\quad Lim_{DE}(u,v,w) = 0 \\
&\text{else} \\
&\quad Lim_{DE}(u,v,w) = Sign(u) min(2|u|, 2|v|, |w|).
\end{aligned}$$

  where $(\nabla W)_{ij}^{HO}$ is either $(\nabla W)_{ij}^{V4}$ or $(\nabla W)_{ij}^{V6}$.

- Piperno Limiter [45]: This limiter is expressed in a factorized form,

$$(\nabla W)_{ij}^{Lim}.\overrightarrow{P_iP_j} = (\nabla W)_{ij}^{C}.\overrightarrow{P_iP_j} \; Lim_{PI}\left(\frac{(\nabla W)_{ij}^{C}.\overrightarrow{P_iP_j}}{(\nabla W)_{ij}^{V4}.\overrightarrow{P_iP_j}}\right),$$

with

$$Lim_{PI}(R) = (\frac{1}{3} + \frac{2}{3}R) \begin{cases} \frac{3\frac{1}{R^2} - 6\frac{1}{R} + 19}{\frac{1}{R^3} - 3\frac{1}{R} + 18} & \text{if } R < 1 \\ 1 + (\frac{3}{2}\frac{1}{R} + 1)(\frac{1}{R} - 1)^3 & \text{if } R \geq 1 \end{cases}$$

where

$$R = \frac{(\nabla W)^C_{ij}.\overrightarrow{P_iP_j}}{(\nabla W)^{V4}_{ij}.\overrightarrow{P_iP_j}}.$$

The Piperno limiter can be only used with the V4 high-order gradient.

### 1.1.5 Boundary conditions

Euler equations are coupled with some boundary conditions. Slip boundary conditions are imposed for bodies and Steger-Warming flux is used to set-up free-stream (external flow) conditions.

**Slip condition (weak form)** This boundary condition consists in weakly imposing $\mathbf{U}.\mathbf{n} = 0$ on the boundary by imposing the following flux:

$$\Phi_{Slip} = F(W) = (0, p\mathbf{n}, 0)^T. \tag{5}$$

**Riemann slip condition** For this boundary condition, we also weakly impose: $\mathbf{U}.\mathbf{n} = 0$. To this end, the flux $\Phi$ between a state $W$ on the boundary and the mirror state $\overline{W}$ is computed:

$$W = \begin{pmatrix} \rho \\ \rho\mathbf{U} \\ \rho E \end{pmatrix} \quad \text{et} \quad \overline{W} = \begin{pmatrix} \rho \\ \rho\mathbf{U} - 2\rho(\mathbf{U}.\mathbf{n})\mathbf{n} \\ \rho E \end{pmatrix}, \tag{6}$$

If the slip condition holds, then $W = \overline{W}$ and $\Phi(W, \overline{W}) = \Phi_{Slip}$. Nevertheless, state $W$ on the boundary does not satisfy the slip condition unless it is strongly imposed. For these reason, the flux between these two states is computed which leads to compute this flux:

$$\Phi_{Slip\ Riemann} = (0, p^*\mathbf{n}, 0)^T, \tag{7}$$

where $p^* = p + \rho\mathbf{U}.\mathbf{n}\min(c, \tilde{c} + \mathbf{U}.\mathbf{n})$ in the case of a HLLC flux. This condition is numerically more stable but more dissipative (and consequently less accurate) than the previous one.

**Free-stream condition** This condition imposes a free-stream uniform state $W_\infty$ at the infinity:

$$W_\infty = \begin{pmatrix} \rho_\infty \\ (\rho\mathbf{U})_\infty \\ (\rho E)_\infty \end{pmatrix}.$$

This condition is imposed via the Steger-Warming flux which is completely upwind on solution $W_i$:

$$\Phi_\infty = A^+(W_i, \mathbf{n}_i)W_i + A^-(W_i, \mathbf{n}_i)W_\infty,$$

where $A$ is the jacobian matrix of F, $A^+ = \frac{|A|+A}{2}$ and $A^- = \frac{|A|-A}{2}$.

### 1.1.6 Time integration

Once the spatial discretization is performed, the temporal part is discretized and an advance in time is performed. Usually, two types of discretization exist for this part. An implicit and an explicit one.

**Explicit time integration** For the advance in time, it is convenient to rewrite the system under the form $\frac{\mathrm{d}W}{\mathrm{d}t} - L(W) = 0$ where:

$$\frac{\mathrm{d}W}{\mathrm{d}t} = L(W) = \frac{1}{|C_i|}\mathbf{R}_i^n,$$

and

$$\mathbf{R}_i^n = \sum_{j\in\nu(i)} \Phi_{ij}^{hllc}(W_i^n, W_j^n, \mathbf{n}_{ij}) + \sum_{f_{slip}|i\in f_{slip}} \Phi_{slip}(W_i^n, n_{f_{slip}}) + \sum_{f_\infty|i\in f_\infty} \Phi_\infty(W_i^n, n_{f_\infty}).$$

The discretization of this formula is done via the high-order multi-step Runge-Kutta scheme. The Runge-Kutta scheme of time-order 2 in with 2 steps is given by:

$$\begin{aligned} W^{(1)} &= W^n + \Delta t^n L(W^n), \\ W^{n+1} &= \frac{1}{2}W^n + \frac{1}{2}W^{(1)} + \frac{1}{2}\Delta t^n L(W^{(1)}). \end{aligned}$$

This scheme has a maximal CFL number of 1.
The Runge-Kutta scheme of time-order 2 in with 5 steps is given by:

$$\begin{aligned} W^{(1)} &= W^n + \frac{1}{4}\Delta t^n L(W^n), \\ W^{(k)} &= W^{(k-1)} + \frac{1}{4}\Delta t^n L(W^{(k-1)}), \quad \text{k=2,..,4}, \\ W^{n+1} &= \frac{1}{5}W^n + \frac{4}{5}W^{(4)} + \frac{1}{5}\Delta t^n L(W^{(4)}), \end{aligned}$$

and has a maximal CFL number of 4. It enables a gain of 60% in terms of CPU performance from the previous scheme.

**Implicit time integration** For an implicit time integration, we prefer to consider the following system:

$$L(W) = \frac{1}{|C_i|}\mathbf{R}_i^{n+1}.$$

Now, if we approximate $L(W)$ by $\frac{\delta W_i^{n+1}}{\Delta t_n^i} = \frac{W_i^{n+1} - W_i^n}{\Delta t_n^i}$, we obtain after linearization of the RHS:

$$\left(\frac{|C_i|}{\Delta t_i^n} I_d - \frac{\partial \mathbf{R}_i^n}{\partial W_i}\right) \delta W_i - \sum_{j\in\nu(i)} \left(\frac{\partial \mathbf{R}_i^n}{\partial W_j}\right) \delta W_j = \mathbf{R}_i^n, \tag{8}$$

where $j \in \nu(i)$ is the set of points $P_j$ connected via an edge to $P_i$. In th case of an HLLC solver, the linearization is given by:

$$\begin{aligned}\Phi_{ij}^{HLLC}(W_i^{n+1}, W_j^{n+1}, \mathbf{n}_{ij}) = \Phi_{ij}^{HLLC}(W_i^n, W_j^n, \mathbf{n}_{ij}) \\ + \frac{\partial \Phi_{ij}^{HLLC}(W_i^n, W_j^n, \mathbf{n}_{ij})}{\partial W_i} \delta W_i \Big\} (A) \\ + \frac{\partial \Phi_{ij}^{HLLC}(W_i^n, W_j^n, \mathbf{n}_{ij})}{\partial W_j} \delta W_j. \Big\} (B)\end{aligned} \tag{9}$$

Term $(A)$ contributes to matrix $D(i,i)$ and term $(B)$ contributes to upper matrix $U(i,j)$ (it is assumed that $i < j$). As $\Phi_{ji}^{HLLC} = -\Phi_{ij}^{HLLC}$, term $-(B)$ contributes to $D(j,j)$ and term $-(A)$ to lower matrix $L(j,i)$.
This linear system is solved at each flow solver iteration using an iterative Newton method. In practice, a maximal number $k_{max}$ of iterations of the Newton method and a targeted order of magnitude by which the residual of the system must be decreased are provided. The iteration is stopped when this targeted residual is reached. To solve the non-linear system, we follow the approach based on Symmetric Gauss-Seidel (SGS) implicit solver. Implementation details can be found in [41, 24].
As implicit schemes are unconditionally stable, CFL number could be arbitrarily large thus allowing large time steps. This is true if the solution is almost converged near the steady-state solution but not at the beginning. Indeed, the solution may blow up (a breakdown of the iteration process) if the CFL is too large when the flow field is not established. Therefore, to converge the Newton method, we generally start with low CFL number and its value is increased while the solution is converged. This process is ruled by CFL laws, see details in [41, 24].

**Time-step computation** The maximal time-step for this numerical scheme at $t^n$ is:

$$\Delta t_i^n = CFL \times \frac{h_i}{c_i^n + ||\mathbf{U}_i^n||}, \tag{10}$$

where $h_i$ is the smallest height of all the elements containing $P_i$, $\mathbf{U}_i$ the velocity and $c_i$ is the speed of sound at $P_i$, ($c_i = \sqrt{\gamma \frac{P_i}{\rho_i}}$). The global time step is found via: $\Delta t = \min_i(\Delta t_i)$. In the case of steady flow, the local time step is used to gain CPU time up to a factor 10. Each vertex goes on with its own time step and its own CFL number.

### 1.1.7 Aerodynamic coefficients

In this section, we define all the aerodynamic coefficients that will be used for the analysis of the solution. In particular, we will define drag, lift and pressure coefficient $C_p$.

**On the computation of reference data** The computation of these coefficients require three parameters:

- $S_{ref}$ the reference surface,
- $L_{ref}$ the reference length,
- $G_{ref}$ the reference (real) barycenter.

If they are not provided, we have to compute them. In this case, $G_{ref}$ is simply the barycenter of the object. For the computation of $L_{ref}$ and $S_{ref}$, we denote by $S_{xy}$ the area of the whole object projected in the plane $xy$ and by $E_w$ the wingspan. In this work, we consider:

$$S_{ref} = S_{xy} \quad \text{and} \quad L_{ref} = \frac{S_{xy}}{E_w} = \frac{S_{ref}}{E_w}\,.$$

**Local aerodynamic coefficient** Analyzing the flow solution on the studied geometry leads us to the visualization of some data on the wetted surface of the body. The more common coefficient is $C_p$ given by:

$$C_p(\mathbf{x}) = \frac{p(\mathbf{x}) - p_\infty}{\frac{1}{2}\rho_\infty \|\mathbf{u}_\infty\|^2},$$

where $\mathbf{u}_\infty$ the infinite inflow velocity, $\frac{1}{2}\rho_\infty\|\mathbf{u}_\infty\|^2$ is the infinite dynamic pressure.

**Drag, lift, momentum** The pressure coefficient vector for inviscid flow is then evaluated by integration of the $C_p$ on body $S$:

$$\begin{pmatrix} C_x \\ C_y \\ C_z \end{pmatrix} = \frac{1}{|S_{Ref}|}\int_S C_p(\mathbf{x})\mathbf{n}(\mathbf{x})\,\mathrm{d}\mathbf{x} = \frac{1}{|S_{Ref}|}\int_S \frac{p(\mathbf{x}) - p_\infty}{\frac{1}{2}\rho_\infty\|\mathbf{u}_\infty\|^2}\mathbf{n}(\mathbf{x})\,\mathrm{d}\mathbf{x},$$

with $\mathbf{n}$ the outward normalized normal and $S_{Ref}$ the reference surface area of body $S$. Finally, the aerodynamic pressure coefficient vector for inviscid flow is obtained considering the orientation of the body in space:

$$\begin{pmatrix} \text{Drag} \\ \text{Slip} \\ \text{Lift} \end{pmatrix} = Rot(\theta,\alpha,\sigma)\begin{pmatrix} C_x \\ C_y \\ C_z \end{pmatrix},$$

where $Rot(\theta,\alpha,\sigma)$ is the rotation matrix defined by the angle of attack $\alpha$, the angle of side slip $\sigma$ and the angle of roll $\theta$. For instance, if the span is along $y$ and the symmetry plane is $xz$, the rotation matrix is:

$$Rot(\theta,\alpha,\sigma) = R^x_\theta\, R^y_\alpha\, R^z_\sigma,$$

with:

$$R^x_\theta = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix}, \quad R^y_\alpha = \begin{pmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{pmatrix} \quad \text{and} \quad R^z_\sigma = \begin{pmatrix} \cos\sigma & \sin\sigma & 0 \\ -\sin\sigma & \cos\sigma & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

We deduce:

$$Rot(\theta,\alpha,\sigma) = \begin{pmatrix} \cos(\alpha)\cos(\sigma) & \cos(\alpha)\sin(\sigma) & -\sin(\alpha) \\ \sin(\alpha)\cos(\sigma)\sin(\theta) - \sin(\sigma)\cos(\theta) & \sin(\alpha)\sin(\sigma)\sin(\theta) + \cos(\sigma)\cos(\theta) & \cos(\alpha)\sin(\theta) \\ \sin(\alpha)\cos(\sigma)\cos(\theta) - \sin(\sigma)\sin(\theta) & \sin(\alpha)\sin(\sigma)\cos(\theta) - \cos(\sigma)\sin(\theta) & \cos(\alpha)\cos(\theta) \end{pmatrix}.$$

Similarly, the moment coefficient vector is given by:

$$\begin{pmatrix} C_l \\ C_m \\ C_n \end{pmatrix} = \frac{1}{|S_{Ref}||L_{Ref}|}\int_S C_p(\mathbf{x})\,(\mathbf{g}\mathbf{x}\times\mathbf{n}(\mathbf{x}))\,\mathrm{d}\mathbf{x},$$

where $\mathbf{g}$ is the body gravity center and $L_{Ref}$ the reference length of body $S$. Then, the aerodynamic moment coefficient vector is obtained considering the orientation of the body in space:

$$\begin{pmatrix} \text{Roll} \\ \text{Pitch} \\ \text{Yaw} \end{pmatrix} = Rot(\theta,\alpha,\sigma)\begin{pmatrix} C_l \\ C_m \\ C_n \end{pmatrix}.$$

### 1.1.8 An example

Here, we show an example that will be used as a reference problem for the next section. A body-fitted circle is considered with a uniform subsonic flow (Mach 0.1). The expected result is a potential flow that is analytically well-known. To compute this flow, an implicit HLLC solver is used along with a V6 scheme and no limiter. Two different boundary condition are employed: the slip boundary condition defined in Eq. (5), that gives the expected solution (Figure 6 left) and a Riemann slip boundary condition defined in Eq. (7) (see Figure 6 right). The result is obtained after 500 iterations of an implicit advance in time and both of them got a residual of magnitude $1.e-6$. It can be noted that the solution obtained with Riemann slip boundary condition is less accurate as it adds numerical dissipation.

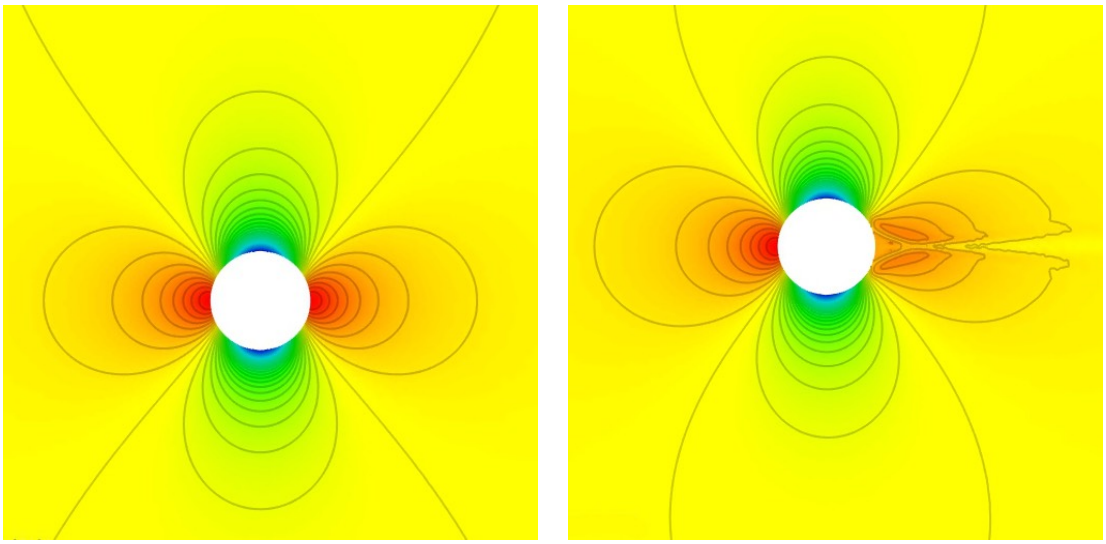


Figure 6: Density field of a potential flow around a circle with slip boundary condition (left) and Riemann slip boundary condition (right).

## 1.2 Continuous mesh framework

In what follows, we introduce the concept of continuous mesh that will be used to derive optimal anisotropic interpolation error estimates.

### 1.2.1 Duality between discrete and continuous entities

In this section, some basic tools concerning the continuous mesh framework are recalled. More details about these techniques can be found in [37, 38, 39, 27] and the references therein. The main idea underlying the metric based mesh adaptation is to change the way distances are computed, via a continuous metric field $\mathbf{M} = (\mathcal{M}(x))_{x\in\Omega}$, where for all $x \in \Omega$, the matrix $\mathcal{M}(x)$ is a metric of $\mathbb{R}^d$, namely a symmetric definite positive matrix of $\mathbb{R}^d$. As it is explained below, the notion of unit mesh with respect to a metric field establishes a link between a metric field and a mesh, which reduces the problem of finding an optimal mesh to the problem of finding an optimal metric field. First of all, the scalar product induced by a constant metric $\mathcal{M}$ is given by:

$$\langle x,\, y\rangle_{\mathcal{M}} = x^T\,\mathcal{M}\,y, \text{ for all } x, y \in \mathbb{R}^d. \tag{11}$$

The following distance results from this scalar product

$$\|x\|_{\mathcal{M}} = \sqrt{x^T\,\mathcal{M}\,x}, \text{ for all } x \in \mathbb{R}^d. \tag{12}$$

Likewise, in the Banach space $\left(\mathbb{R}^d, \|.\|_{\mathcal{M}}\right)$, the distance between two points $a, b \in \mathbb{R}^d$ is given by:

$$\ell_{\mathcal{M}}\left(ab\right) = \sqrt{(ab)^T\,\mathcal{M}\,ab}, \tag{13}$$

where $ab$ stands for the vector linking $a$ to $b$. In the same way, if $\mathbf{M} = \left(\mathcal{M}(x)\right)_{x\in\Omega}$ is a smooth metric field on $\Omega$, the length of the segment between $a$ and $b$ is computed by using the integral formula:

$$\ell_{\mathcal{M}}\left(ab\right) = \int_0^1 \sqrt{(ab)^T\,\mathcal{M}\left(\gamma(t)\right)ab}\,dt, \tag{14}$$

where $\gamma(t) = (1-t)a + tb$.
In a same manner, the volume of an element under a metric field is given by:

$$|\Omega|_{\mathcal{M}} = \int_\Omega \sqrt{\det \mathcal{M}(x))}\mathrm{d}\Omega.$$

As the next definition shows, there is a strong correspondence between continuous metric spaces and meshes, through the notion of unit mesh with respect to a metric field.

**Definition 1.1** *An element $K$ of a mesh $\mathcal{H}$ is said to be unit with respect to a continuous metric field $\mathbf{M} = \left(\mathcal{M}(x)\right)_{x\in\Omega}$ if the lengths of its edges equal 1. That is to say, if $\{e_1, ...e_{m+1}\}$ are the edges of an element $K$ of $\mathcal{H}$, then we have:*

$$\ell_{\mathbf{M}}(e_i) = 1, \quad \textit{for all} \quad i \in \{1, ..., m+1\}\,.$$

**Definition 1.2** *A mesh $\mathcal{H}$ of a domain $\Omega \subset \mathbb{R}^d$ is said to be unit with respect to a continuous metric field $\mathbf{M} = \left(\mathcal{M}(x)\right)_{x\in\Omega}$ if all its elements are unit.*

In particular, if $K$ is a tetrahedron, then its volume is given by:

$$|K|_{\mathcal{M}} = \frac{\sqrt{2}}{12} \quad \text{and} \quad |K| = \frac{\sqrt{2}}{12}(\det(\mathcal{M}(x)))^{-\frac{1}{2}},$$

In fact, for a given element $K$ such that $|K| \neq 0$, there exists a unique metric tensor $\mathcal{M}$ for which element $K$ is unit with respect to $\mathcal{M}$. Consequently, the function *unit with respect to* defines classes of equivalences of discrete elements. A metric tensor $\mathcal{M}$ is therefore called a continuous element. Similarly, $\mathbf{M} = \left(\mathcal{M}(x)\right)_{x\in\Omega}$, is called a continuous mesh. Using this analogy, a correspondence between discrete and continuous entities can be set [38]. This is summarized in Table 1.

### 1.2.2 Optimal control of the interpolation error in $L^p$ norm

Mesh adaptation consists in finding the mesh $\mathcal{H}$ of a domain $\Omega$ that minimizes a given error for a given function $u$. For the sake of simplicity, we consider here the linear interpolation error $u - \Pi_h u$ controlled in $L^p$ norm and that $u$ is twice differentiable. The problem is thus stated in an *a priori* way:

| DISCRETE | CONTINUOUS |
|---|---|
| Element $K$ | Metric tensor $\mathcal{M}$ |
| Mesh $\mathcal{H}$ of $\Omega_h$ | Riemannian metric space $\mathbf{M}(\mathbf{x}) = (\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$ |
| Number of elements $N_e$ | Complexity $\mathcal{C}(\mathcal{M}) = 6\sqrt{2}\int_\Omega \sqrt{\det(\mathcal{M}(\mathbf{x}))}\mathrm{d}\Omega$ |
| Linear interpolate $\Pi_h u$ | Continuous linear interpolate $\Pi_{\mathcal{M}} u$ |

Table 1: The continuous mesh model draws a duality between the discrete domain and the continuous domain.

$$\text{Find } \mathcal{H}_{opt} \text{ having } N \text{ vertices such that } \mathbf{E}_{L^p}(\mathcal{H}_{opt}) = \min_{\mathcal{H}} ||u - \Pi_h u||_{L^p(\Omega_h)},$$

with $H_u$ the hessian of $u$. Such a formulation has several drawbacks, notably this is NP-complete. For this reason, its continuous counterpart is considered:

$$\text{Find } \mathbf{M}_{opt} \text{ having a complexity } \mathcal{N} \text{ such that } \mathbf{E}_{L^p}(\mathbf{M}_{opt}) = \min_{\mathbf{M}} ||u - \Pi_{\mathcal{M}} u||_{L^p(\Omega_h)},$$

where $\Pi_{\mathcal{M}} u$ is the continuous interpolate defined (in 3D) by:

$$\Pi_{\mathcal{M}} u(\mathbf{a}) = u(\mathbf{a}) + \nabla u(\mathbf{a}) + \frac{1}{20}\mathrm{trace}(\mathcal{M}(\mathbf{a})^{-\frac{1}{2}}|H_u(\mathbf{a})|\mathcal{M}(\mathbf{a})^{-\frac{1}{2}}).$$

The following problem is solved using a calculus of variations. Indeed, the previous problem can be rewritten into:

$$\begin{aligned}\text{Find } \mathbf{M}_{L^p} = \min_{\mathbf{M}} \mathbf{E}_{L^p}(\mathbf{M}) &= \left(\int_\Omega (u(\mathbf{x}) - \Pi_{\mathcal{M}} u\,(\mathbf{x}))^p \,\mathrm{d}\mathbf{x}\right)^{\frac{1}{p}} \\ &= \left(\int_\Omega \mathrm{trace}\left(\mathcal{M}(\mathbf{a})^{-\frac{1}{2}}|H_u(\mathbf{a})|\mathcal{M}(\mathbf{a})^{-\frac{1}{2}}\right)^p \mathrm{d}\mathbf{x}\right)^{\frac{1}{p}},\end{aligned}$$

under the constraint $\mathcal{C}(\mathbf{M}) = 6\sqrt{2}\int_\Omega \sqrt{\det(\mathcal{M}(\mathbf{x}))}\mathrm{d}\Omega = \mathcal{N}$. Using a calculus of variations, we prove that an unique solution exists on the space of continuous meshes. In [39], this unique solution is given by:

$$\mathcal{M}_{L^p}(\mathbf{x}) = \mathcal{N}^{\frac{2}{3}}\left(\int_\Omega \det(|H_u(\tilde{\mathbf{x}})|)^{\frac{p}{2p+3}}\mathrm{d}\tilde{\mathbf{x}}\right)^{-\frac{2}{3}} \det(|H_u(\mathbf{x})|)^{-\frac{1}{2p+3}} H_u(\mathbf{x}). \tag{15}$$

## 1.3 Feature-based anisotropic mesh adaptation for steady flows

In the previous section, we have presented the continuous mesh model, on which is based a powerful framework to handle mathematically adapted meshes and we have also seen its application to the optimal control of the interpolation error. However, the assumption that was made is

that the solution is analytically known. In this section, we tackle the subject in the case where the solution is discrete and only known at the vertices of the mesh. To this end, we present the classic mesh adaptation process for steady flows, which is a fixed-point algorithm where both solution and meshes are converged. The principle is to start from an initial coarse mesh and then to perform successive mesh adaptations. This way, the physics are progressively captured. At each stage of this fixed-point algorithm, the flow solver is called, an error estimate is then used to derive a metric. The latter is used as size map to perform mesh regeneration and adaptation. The considered estimate is a feature-based (or hessian-based) one.

### 1.3.1 Mesh adaptation algorithm for numerical simulations

Anisotropic mesh adaptation is a non-linear problem, therefore an iterative process is required to solve this problem. For steady simulations, an adaptive computation is carried out *via* a mesh adaptation loop inside which an algorithmic convergence of the mesh-solution couple is sought. This mesh adaptation loop is explained in **Algorithm 1** and in Figure 7 where $\mathcal{H}$, $\mathcal{S}$, $\mathcal{M}$ denote respectively meshes, solutions and metrics.

**Algorithm 1:** Mesh adaptation for Steady flows

**init:** Initial mesh and solution $(\mathcal{H}_0, \mathcal{S}_0)$ and target complexity $\mathcal{N}$

**for** $i = 0, n_{adap}$ **do**

1. $(S_i)$ = Compute Solution with the flow solver from pair $(\mathcal{H}_i, \mathcal{S}_i^0)$;
   **if** $i = n_{adap}$ **then** break;
2. $(\mathcal{M}_{L^p,i})$ = Compute metric $\mathcal{M}_{L^p}$ according to selected error estimate from $(\mathcal{H}_i, \mathcal{S}_i)$;
3. $(\mathcal{H}_{i+1})$ = Generate a new adapted mesh from pair $(\mathcal{H}_i, \tilde{\mathcal{M}}_{L^p,i})$;
4. $(\mathcal{S}_{i+1}^0)$ = Interpolate new initial solution from $(\mathcal{H}_{i+1}, \mathcal{H}_i, \mathcal{S}_i)$

Note that this loop can be applied several times for a sequence of given mesh complexities, for instance $\mathcal{N}$, $2\mathcal{N}$, ...

### 1.3.2 Hessian-based anisotropic mesh adaptation

Unlike the case where $u$ is known continuously, two major difficulties occur when applying mesh adaptation to numerical simulations:

- the continuous solution of the problem $u$ is not known, only its numerical approximation $u_h$ is available (it is provided by the flow solver),
- a control of the approximation error is expected, $u - u_h$ instead of $u - \Pi_h u$.

**Control of the approximation error** Here, the interpolation theory is applied in the case when $u_h$ is only known as a piecewise linear approximation of the solution. Indeed, in this case, the interpolation error estimates (15) cannot be readily applied to $u$ nor $u_h$.
Let $\bar{V}_h^n$ be the space of piecewise polynomials of degree $n$ (possibly discontinuous) and $V_h^k$ be the space of continuous piecewise polynomials of degree $n$ associated with a given mesh $\mathcal{H}$ of

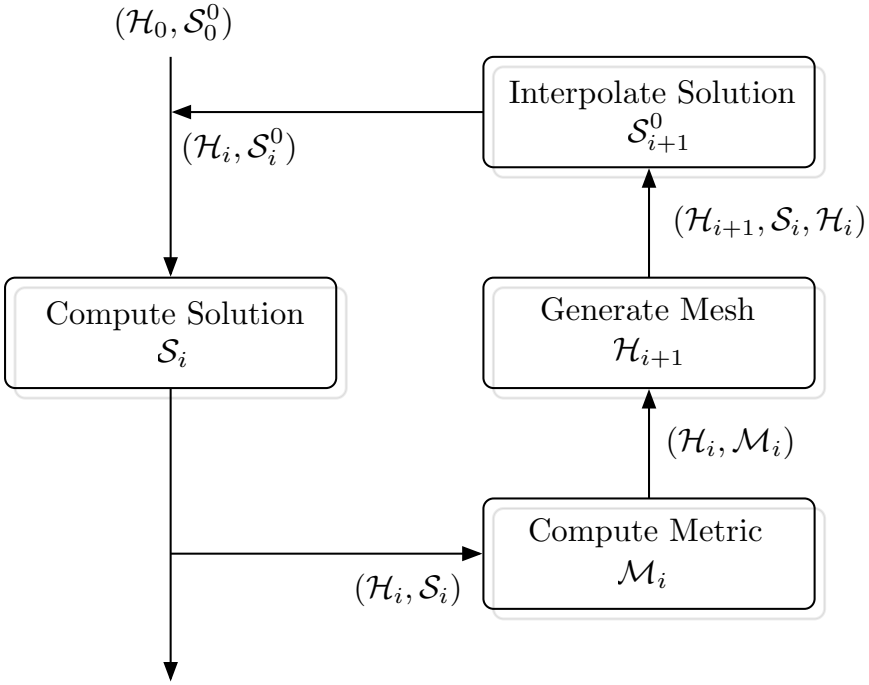



Figure 7: Mesh adaptation algorithm.

domain $\Omega_h$. We denote by $R_h$ a reconstruction operator applied to numerical approximation $u_h$. This reconstruction operator can be either a recovery process [56], a hierarchical basis [9], or an operator connected to an *a posteriori* estimate [30]. We assume that the reconstruction $R_h u_h$ is more accurate than $u_h$ for a given norm $\|.\|$ in the sense that:

$$\|u - R_h u_h\| \leq \alpha \|u - u_h\| \quad \text{where} \;\; 0 \leq \alpha < 1 \,.$$

From the triangle inequality, we deduce:

$$\|u - u_h\| \;\leq\; \frac{1}{1-\alpha} \|R_h u_h - u_h\| \,.$$

$R_h$ has the property: $\Pi_h R_h \phi_h = \phi_h \,,\; \forall \phi_h \in V_h^1$ , the approximation error of the solution can be bounded by the interpolation error of reconstructed function $R_h u_h$:

$$\|u - u_h\| \leq \frac{1}{1-\alpha} \|R_h u_h - \Pi_h R_h u_h\| \,.$$

From previous section, if $\mathcal{H}_{L^p}$ is an optimal mesh to control the interpolation error in $L^p$ norm of $R_h u_h$, then the following upper bound of the approximation error can be exhibited:

$$\|u - u_h\|_{L^p(\Omega_h)} \leq \frac{3\, N^{-\frac{2}{3}}}{1-\alpha} \left( \int_\Omega \det \left( |H_{R_h u_h}(\mathbf{x})| \right)^{\frac{p}{2p+3}} \mathrm{d}\mathbf{x} \right)^{\frac{2p+3}{3p}} \,.$$

This upper bound is true for any $n \geq 1$ but is too large for $n > 1$. Though, the same analysis can be performed in general for derivatives of order $n+1$ and similar relations can be found in [19].

In the context of numerical simulations, $u_h$ lies in $V_h^1$ and its derivatives $\nabla u_h$ in $\bar{V}_h^0$. We propose a reconstruction operator from $V_h^1$ into $V_h^2$ based on $P^2$ Lagrange finite element test functions. As approximate solution $u_h$ is only known at mesh vertices, we need to reconstruct mid-edge values. To this end, we consider the $L^2$-projection operator $\mathcal{P} \ : \ \bar{V}_h^0 \rightarrow V_h^1$ defined by [18]:

$$\nabla_R u_h = \mathcal{P}(\nabla u_h) = \sum_{\mathbf{p}_i \in \mathcal{H}} \nabla_R\, u_h(\mathbf{p}_i)\phi_i \quad \text{where} \quad \nabla_R\, u_h(\mathbf{p}_i) = \frac{\sum_{K_j \in S_i} |K_j| \nabla(u_{h|K_j})}{\sum_{K_j \in S_i} |K_j|}\,,$$

where $\mathbf{p}_i$ denotes the $i^{th}$ vertex of mesh $\mathcal{H}$, $S_i$ is the stencil of $\mathbf{p}_i$, $\phi$ the basis function of $V_h^1$ and $|K_j|$ denotes the volume of element $K_j$. These nodal recovered gradients are used to evaluate mid-edge values. For edge $\mathbf{e} = \mathbf{pq}$, the mid-edge value $u_h(\mathbf{e})$ is given by:

$$u_h(\mathbf{e}) = \frac{u_h(\mathbf{p}) + u_h(\mathbf{q})}{2} + \frac{\nabla_R u_h(\mathbf{p}) - \nabla_R u_h(\mathbf{q})}{8} \cdot \mathbf{pq}\,,$$

which corresponds to a cubic reconstruction. The reconstructed function $R_h u_h$ of $V_h^2$ writes:

$$R_h u_h = \sum_{\mathbf{p}_i} u_h(\mathbf{p}_i)\psi_{\mathbf{p}_i} + \sum_{\mathbf{e}_j} u_h(\mathbf{e}_j)\psi_{\mathbf{e}_j}\,,$$

where $\psi_{\mathbf{p}} = \phi_{\mathbf{p}}\,(2\,\phi_{\mathbf{p}} - 1)$ and $\psi_{\mathbf{e}} = 4\,\phi_{\mathbf{p}}\,\phi_{\mathbf{q}}$ are the $P^2$ Lagrange test functions. This reconstructed function can be rewritten $R_h u_h = u_h + z_h$ and by definition verifies:

$$\Pi_h R_h u_h = u_h \quad \text{thus} \quad \Pi_h z_h = 0\,.$$

Therefore, we deduce:

$$\|R_h u_h - \Pi_h R_h u_h\| = \|u_h + z_h - u_h\| = \|z_h - \Pi_h z_h\|\,.$$

Finally, the approximation error can be estimated by evaluating the interpolation error of $z_h$:

$$\|u - u_h\| \ \leq \ \frac{1}{1-\alpha}\|z_h - \Pi_h z_h\| \ \leq \ \frac{3\,N^{-\frac{2}{3}}}{1-\alpha}\left(\int_\Omega \det\left(|H_{z_h}(\mathbf{x})|\right)^{\frac{p}{2p+3}} \mathrm{d}\mathbf{x}\right)^{\frac{2p+3}{3p}}\,.$$

Note that the Hessian of $z_h$ lies in $\bar{V}_h^0$. If nodal values are needed to build $\mathcal{M}_{L^p}$, then the $L^2$-projection operator can be applied to these hessians [18]. This recovery procedure is somehow similar to the ones of [56, 57]. Other reconstruction operators can be applied such as the double $L^2$-projection, the least square method or eventually the Green formula based approach [25].

## 1.4 Conclusion

In this section, we have recalled all the basic tools and algorithmic operations used to solve the Euler equations and then perform mesh adaptation on it. The purpose of the next section is to extend these tools to the case of the embedded boundary method applied to the Euler equations.

# 2 Embedded boundary method and applications

Several strategies are used in CFD to deal with embedded geometries. Once the boundary is detected and the inner part of the object is set, two approaches exist. The first approach consists in applying a so-called penalization method [7, 1] by weakly imposing a given value in the inner part of the geometry. This way, the presence of the object is simulated and the appropriate boundary condition is applied. These methods are specifically used for Navier-Stokes equations with Dirichlet boundary conditions (*e.g.* u=0 or u=v on the boundary). Another approach consists in imposing the value in a strong way and apply the wished boundary condition at the interface. This method is used in particular for the Euler equations. The interface is then created in a specific way. It can either be the surface induced by the resulting finite volume cells [53] or a result of a local modification of the scheme [22, 23, 43]. This is the last strategy, called cut-cell method, that is used in this work. Finally, it can be noted that the coupling of embedded methods with anisotropic mesh adaptation strategies has been made in several contexts. First, it is used in the finite element resolution of the incompressible Navier-stokes equations with objects represented by a level-set function [47, 48], then this approach can be completed with the possibility of simulating fluid-structure interactions problems [28] and finally it has been tackled in the context of the compressible Navier-Stokes equations with penalization [1].
In this chapter, we present the principle of the embedded boundary method applied to the Euler equations. The processing of an embedded geometry is done in three steps: the intersection between the geometry and the background mesh is performed, then the dual mesh is modified by means of a cut-cell method and finally, modifications are brought to the numerical scheme with the implementation of a specific boundary condition that simulates the presence of the object. In particular, we show the implementation details needed in the vertex-centered finite volume CFD solver `Wolf` and highlight it with numerical examples that show that the method gives the expected results. Then, a coupling of this method with anisotropic mesh adaptation using `Feflo.a/AMG` is performed to improve the accuracy of the results.

## 2.1 Geometrical intersection

In this section, we detail how to deal with the embedded boundary from a geometrical point of view and in particular how the intersection between the embedded geometry and the background mesh is performed. The detection of the boundary is crucial as it will be later useful for the modification of the dual scheme and the implementation of a boundary condition to model the presence of the boundary.

### 2.1.1 Intersection algorithm

The detection of the embedded boundary relies on an intersection algorithm. In 2D, the used process is an edge/edge intersection algorithm and in 3D it is a triangle/edge intersection algorithm.

**2D case** The formula to compute an edge/edge intersection is the following [5]:
Let $e_P = [P_0P_1]$ and $e_Q = [Q_0Q_1]$ be two edges of $\mathbb{R}^2$ and let us note $\mathbf{n}_{e_P}$ and $\mathbf{n}_{e_Q}$ their counterclockwise oriented normals.
If $P \in \mathbb{R}^2$, its *power* with respect to an edge $e_Q = \overrightarrow{Q_0Q_1}$ writes:

$$\mathcal{P}(P, e_Q) = \overrightarrow{Q_0P}.\frac{\mathbf{n}_{e_Q}}{||\mathbf{n}_{e_Q}||} = \overrightarrow{Q_1P}.\frac{\mathbf{n}_{e_Q}}{||\mathbf{n}_{e_Q}||}.$$

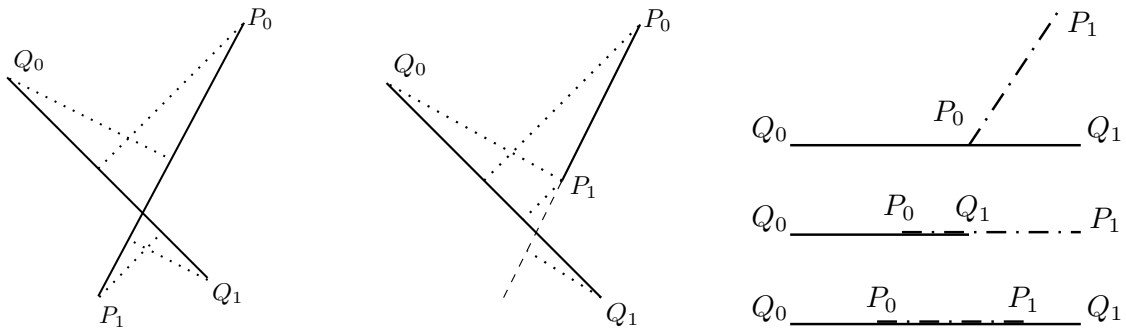



Figure 8: Edge/edge intersection case: intersection (left), no intersection (middle) and degenerate cases (right).

If the case is not degenerated ($\mathcal{P}(P_0, e_Q) \neq 0$ and $\mathcal{P}(P_1, e_Q) \neq 0$), then there is edge/edge intersection if an only if:

$$\mathcal{P}(P_0, e_Q)\mathcal{P}(P_1, e_Q) < 0 \quad \text{and} \quad \mathcal{P}(Q_0, e_P)\mathcal{P}(Q_1, e_P) < 0.$$

The intersection point $X$ is then deduced:

$$X = P_0 + \frac{\mathcal{P}(P_0, e_Q)}{\mathcal{P}(P_0, e_Q) - \mathcal{P}(P_1, e_Q)}\overrightarrow{P_0P_1}.$$

The three degenerate cases are the following:

- A power equals 0, for instance $\mathcal{P}(P_0, e_Q) = 0$, then there is intersection if and only if $\mathcal{P}(Q_0, e_P)\mathcal{P}(Q_1, e_P) < 0$ and $X = P_0$.
- Two power equal 0, one for each edge, for instance $\mathcal{P}(P_0, e_Q) = 0$ and $\mathcal{P}(Q_0, e_P) = 0$. In this case, there is intersection and $X = P_0 = Q_0$
- All the power equal 0, the edges are in fact aligned. There is intersection if and only if both edges overlap, which leads to the following two sub-cases:
  - One intersection point that is the common point of both edges.
  - Two intersection points that correspond to the extremities of one edge totally embedded in the other one or an extremity of each.

**3D case** Like in the 2D case, degenerated cases appear. In this work, the strategy is to treat them in a first step as it is done in [3].

**Dealing with degenerated cases.** We first introduce the 3D version of the *power*, of point $P$ with respect to face $F = [P_0, P_1, P_2]$, whose outward normal is depicted by $\mathbf{n}_F$:

$$\mathcal{P}(P, F) = \overrightarrow{P_kP} \cdot \frac{\mathbf{n}_F}{\|\mathbf{n}_F\|} \text{ where } k \text{ is either } 0, 1 \text{ or } 2\,. \tag{16}$$

The distance of point $P$ with respect to edge $e_i = \overrightarrow{P_0P_1}$ is:

$$\mathcal{P}(P, e_i) = \frac{\|\overrightarrow{P_0P_1} \times \overrightarrow{P_0P}\|}{\|\overrightarrow{P_0P_1}\|} = \frac{\|\overrightarrow{P_0P} \times \overrightarrow{P_1P}\|}{\|\overrightarrow{P_0P_1}\|}\,.$$

All possible vertex degenerated cases are checked according to $\mathcal{P}(P_j, F)$ values:

- is a vertex inside a face ?
- is a vertex on a edge ?
- are two vertices coinciding ?

Afterwards, edge/edge intersections - which are degenerated cases - are tested. Let $P_0P_1$ and $Q_0Q_1$ be the two considered lines. A necessary condition is that the two lines are coplanar. If it is the case, the intersection point $\mathbf{x}$ is evaluated following Hill's approach [29]:

$$\begin{aligned} \mathbf{x} &= P_0 + \frac{\left(\overrightarrow{P_0Q_0} \times \overrightarrow{Q_0Q_1}\right) \cdot \left(\overrightarrow{P_0P_1} \times \overrightarrow{Q_0Q_1}\right)}{\|\overrightarrow{P_0P_1} \times \overrightarrow{Q_0Q_1}\|^2} \overrightarrow{P_0P_1} = P_0 + s\,\overrightarrow{P_0P_1}\,, \\ \text{or} \quad \mathbf{x} &= Q_0 + \frac{\left(\overrightarrow{P_0P_1} \times \overrightarrow{Q_0P_0}\right) \cdot \left(\overrightarrow{P_0P_1} \times \overrightarrow{Q_0Q_1}\right)}{\|\overrightarrow{P_0P_1} \times \overrightarrow{Q_0Q_1}\|^2} \overrightarrow{Q_0Q_1} = Q_0 + t\,\overrightarrow{Q_0Q_1}\,. \end{aligned}$$

Now, to check if the intersection point is an intersection between the two segments, we have to check that:

$$0 \leq s \leq 1 \quad \text{and} \quad 0 \leq t \leq 1\,.$$

**Edge/face intersection.** Dealing with degenerated cases first simplifies the following edge-face intersection procedure. Indeed, the computation of all the above geometric degeneracy treats in particular all the possible coplanar edge-face intersections as depicted in Figure 9.

Now, only non-coplanar edge-face intersection remains where the intersection point lies inside the face. The algorithm checks the 48 possible edge-face intersections. Let us denote the considered edge by $e = [P_0, P_1]$ and face by $F = [Q_0, Q_1, Q_2]$. As it is a non-coplanar case, the edge's vertex powers with respect to the face are not zero: $\mathcal{P}(P_0, F) \neq 0$ and $\mathcal{P}(P_1, F) \neq 0$. A necessary condition for the edge-face intersection is:

$$\mathcal{P}(P_0, F)\,\mathcal{P}(P_1, F) \;<\; 0\,.$$

If this condition is satisfied, the point of intersection between the edge and the plane defined by the face is computed:

$$\mathbf{x} = P_0 + \frac{\mathcal{P}(P_0, F)}{\mathcal{P}(P_0, F) - \mathcal{P}(P_1, F)} \overrightarrow{P_0P_1}\,.$$

Finally to verify if the edge-face intersection effectively occurs, *i.e.,* point $\mathbf{x}$ lies inside face $F$, the signs of barycentrics of point $\mathbf{x}$ with respect to face $F$ are analyzed. If there is intersection, point $\mathbf{x}$ is added to the intersection points list.

### 2.1.2 Detection of the embedded boundary in 2D

Let us give a mesh $\mathcal{H}$ and the discretization of the embedded boundary $I_h$ as a set of edges. Let us note $\mathcal{D}_h$, a sub-domain of $\mathcal{H}$ whose boundary is $I_h$ and whose inner part is the approximation of the interior of the embedded object.
The first step is to determine which vertices of $\mathcal{H}$ are covered by $\mathcal{D}_h$ and to find the set of edges of $\mathcal{H}$ denoted as $\mathcal{E}_h$ that intersect $I_h$. The edges of $\mathcal{H}$ and $I_h$ are respectively denoted by $e_{\mathcal{H}}$ and $e_{I_h}$. A core concept in all the following algorithms is the notion of mark whose principle is detailed here. This concept is the key to avoid algorithms with a "searching" of quadratic complexity.

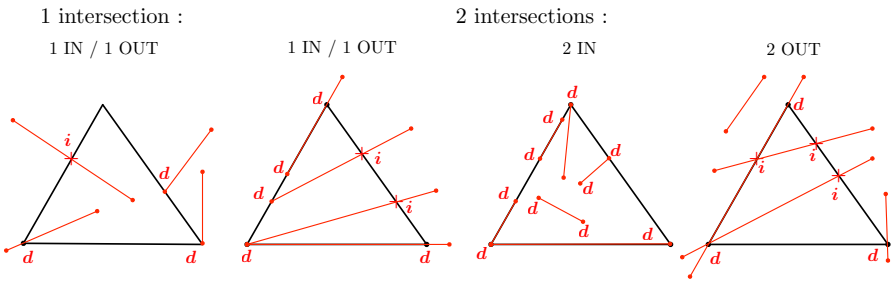



Figure 9: *All possible intersection configurations for coplanar pair edge-face (all kinds of intersection are gathered by edge-face configurations). Four configurations can occur. One intersection with an edge's vertex in the face and the other vertex out. Two intersections with zero (resp. one or two) edge's vertex in the face and two (resp. one or zero) edge's vertices outside of the face. In the pictures, an intersection marked by "i" or "d" represents a pure or a degenerated intersection, respectively.*

**Principle of the mark.** When deploying an algorithm, it is common to create a list or stack of entities to be analyzed. To create this type of lists, a naive strategy would be to check if an entity is in the list before adding it. However, doing this creates an algorithm of quadratic complexity. A solution to overcome this issue is to create a flag table. For each of the entities is associated a flag. If, for instance, the considered object is added to the list its flag is activated. Thus, if its flag is already activated, this means that the element is already in the list and consequently and there is no need in adding it to the list. This way, the quadraticity of the algorithm vanishes. However, the flag table has to be reset before each of its use and this could be costly, in particular on large-size meshes. To this end, the mark is introduced. It consists in an integer value that is attached to the mesh and initialized to 0. The flag tables are then replaced by mark tables, that are also initialized to 0. Before each use of any of the mark tables, the mark is incremented by 1 and then, the procedure is similar as with the flag table: if an object is added to the list, its mark (*i.e.* its corresponding value in the appropriate mark table) is set to the current mark. However, if its mark already equals the current mark, it is not added. Afterwards, the procedure is quite similar.

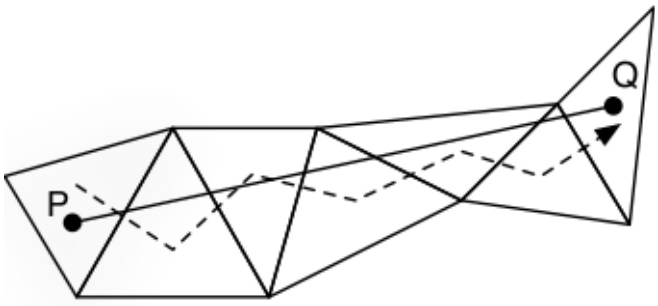



Figure 10: Pipe of an embedded edge.

For instance, to detect the intersection between a triangle of $\mathcal{H}$ and $e_{I_h}$, the used procedure is as follows for a given edge, we choose one of the vertices of the edge, then we locate the triangle

of $\mathcal{H}$ containing the vertex. Afterwards, all neighboring triangles of the previous triangles are checked in order to find if the embedded edge goes through this triangle or not *e.g* if at least one of the edges of the triangle is intersected. The mark is used to tag all the visited vertices and triangles along the path of the embedded edge. Note that the obtained set of tagged triangles is called a "pipe".
If for a triangle $K_h$, we note $K_h^1, K_h^2, K_h^3$ the three triangles respectively sharing the edge $(e_1, e_2, e_3)$ with it, and if we note $\mathcal{K}_h$ the sort list of triangles to be analyzed for an intersection. Then, the intersection algorithm is given in **Algorithm 2**.

**Algorithm 2:** Two-dimensional geometry/mesh intersection algorithm

```
init: E_v = ∅
for e_{I_h} = [P,Q] ∈ I_h do
    mark = mark +1
    init: K_h | P ∈ K_h; i = 0; K_h = ∅; markTri(K_h) = mark
    while (i ≤ |K_h|) do
        if (i ≠ 0) then K_h = K_h[i];
        for j = 1..3 do
            if e_{I_h} ∩ e_j then
                E_v = E_v ∪ {e_j}
                if markTri(K_h^j) ≠ mark then
                    markTri(K_h^j) = mark ; K_h = K_h ∪ {K_h^j}
        i = i + 1
```

### 2.1.3 Generalization to 3D

The three-dimensional procedure is quite similar to the 2D algorithm. The embedded geometry is represented as a triangular surface mesh and the mesh $\mathcal{H}$ is a tetrahedral mesh. Since the embedded boundary method is studied for Finite Volumes purposes, we only need to handle the intersections of the embedded geometry discretization with the edges of $\mathcal{H}$. Basically, the idea is to locate the tetrahedra containing the 3 vertices of a given triangle of the embedded mesh and then to go through the neighboring tetrahedra containing the geometry thanks to an edge/triangle intersection algorithm. For the needs of the algorithm, let us note $(e_i)_{i\in[1,6]}$, the 6 edges of tetrahedron $K_h$ and $(F_i)_{i\in[1,4]}$, the faces of $K_h$. Let us also note $\mathcal{K}_h^{e_i}$, the set of tetrahedra sharing the edge $e_i$ with $K_h$ and $K_h^{F_i}$ the tetrahedron sharing the face $F_i$ with $K_h$. Finally the triangles of the surface mesh $I_h$ are noted $T_{I_h}$. Finally, we note $\mathcal{K}_h$ the sort list of triangles to be analyzed for an intersection. The 3D intersection algorithm is given in **Algorithm 3**.

### 2.1.4 Detection of the covered vertices

Once the intersection is performed, the next step is to make sure if the intersection leads to an opened or closed embedded geometry. Indeed, if some vertices are covered by the geometry (*e.g.* it is a closed geometry), a constant solution is attached at this vertex and specific changes need to be done. The determination of the covered vertices is a consequence of the intersection algorithm: thanks to the mark and the set of connected neighbors to $P_i$ noted $\nu(P_i)$, it is easy

**Algorithm 3:** Three-dimensional geometry/mesh intersection algorithm

**init:** $\mathcal{E}_v = \emptyset$
**for** $T_{I_h} = [P, Q, R] = [e^1_{T_{I_h}}, e^2_{T_{I_h}}, e^3_{T_{I_h}}] \in I_h$ **do**
  mark = mark +1
  **init:** $K_h \mid P \in K_h$; $i = 0$; $\mathcal{K}_h = \emptyset$; markTet($K_h$) = mark
  **while** $i \leq |\mathcal{K}_h|$ **do**
    **if** $i \neq 0$ **then** $K_h = \mathcal{K}_h[i]$ ;
    **for** $j = 1..6$ **do**
      **if** $T_{I_h} \cap e_j$ **then** $\mathcal{E}_v = \mathcal{E}_v \cup \{e_j\}$;
      **for** $K_h^{shell} \in \mathcal{K}_h^{e_i}$ **do**
        **if** markTet($K_h^{shell}$) $\neq$ mark **then**
          mark($K_h^{shell}$) = mark ; $\mathcal{K}_h = \mathcal{K}_h \cup \{K_h^{shell}\}$
    **for** $(j = 1..6)$ and $(k = 1..3)$ **do**
      **if** $(e^k_{T_{I_h}} \cap F_j)$ and (markTet($K_h^j$) $\neq$ mark) **then**
        markTet($K_h^j$) = mark ; $\mathcal{K}_h = \mathcal{K}_h \cup \{K_h^j\}$
    $i = i + 1$

to determine them. In the following, the uncovered vertices will be marked and then the covered vertices are found. Let us note $V_h$ as the set of the already checked vertices. Note this algorithm is independent of the dimension.

**Algorithm 4:** Algorithm for the detection of $\mathcal{D}_h$

**init:** $P \notin \mathcal{D}_h$; $V_h = \emptyset$; i = 0
mark = mark +1
**while** $i \leq |V_h|$ **do**
  **for** $Q \in \nu(P)$ **do**
    **if** $[P, Q] \in \mathcal{E}_v$ and markVer($Q$) $\neq$ mark **then**
      markVer($Q$) = mark
      $V_h = V_h \cup \{Q\}$
$\mathcal{D}_h = \mathcal{H} \setminus V_h$

## 2.2 Modification of the dual mesh via a cut-cell method

In this section, we detail the modification required to the finite volume dual mesh (see Section 1.1.2) when an embedded object is detected. Several strategies exist to deal with this problem. It is possible to directly apply slipping mirror boundary conditions, the obtained interface is the so-called *surrogate interface* [53]. It is defined using the finite volume cells associated to the vertices that are not covered by the embedded geometry. Albeit its simplicity, this strategy is not really accurate. A possibility to remedy with that is to use a *cut-cell method* [54, 55, 22, 23, 43]. It consists in locally modifying the finite volume cells defined in Section 1.1.2 so that they match with the embedded interface.

### 2.2.1 Finite volume median cells

In the used solver, the finite volume cells are the median cells. In 2D, this methods consists in building the cells triangle by triangle. Inside each of the triangles, the three medians that link the middle $X_i$ of an edge $e_i$ with the opposite vertex $P_i$ are considered. Their intersection is the gravity center of the triangle $X_G$ and using it with middle of the edges, we are able to divide a triangle into 3 quadrilaterals, one for each vertex of the triangle (see Figure 11 on the left). The 2D median finite volume cell of a point $P$ consists in the gathering of all its associated quadrilaterals of the triangles containing $P$. In 3D, each tetrahedron is split into four hexahedra. To construct them for a point $P_i$ of the tetrahedron, we use the middles $(X_j^i)_{j\in[1,3]}$ of the three incident edges to $P_i$ the gravity centers $(XF_j^i)_{j\in[1,3]}$ of the 3 faces containing $P_i$, the gravity center $X_G$ of the tetrahedron and the considered vertex $P_i$ (see Figure 11 on the right). The 3D median finite volume cell of a point $P$ consists then in the gathering of all its associated hexahedra of the tetrahedra containing $P$.

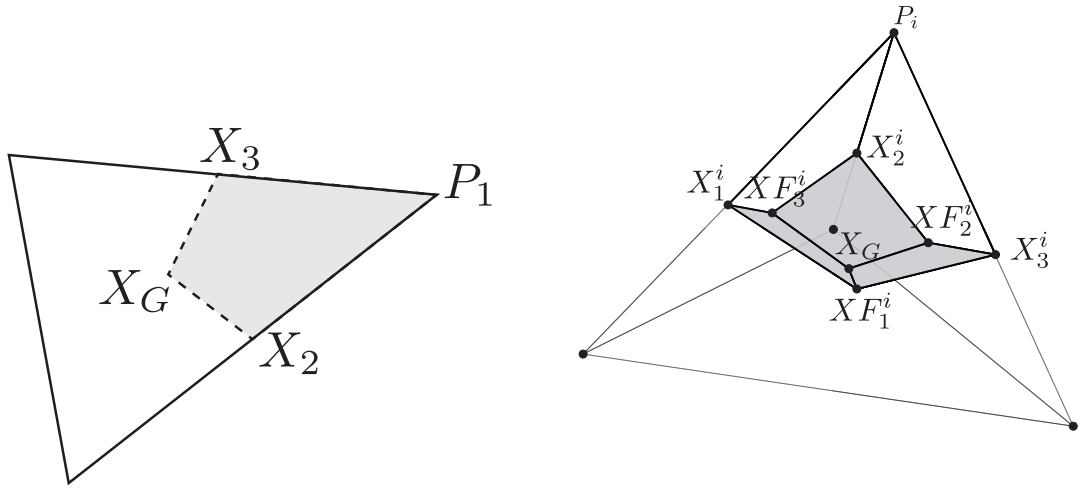



Figure 11: Finite volume median cells.

### 2.2.2 Two-dimensional cut-cell

First, we tackle the two-dimensional case. Let us note $K_h$ the triangles of $\mathcal{H}$ and let us reuse the notations of the previous section. In the most common case, two edges are intersected once. Let us assume it is $e_1$ and $e_2$. Let us note $X_1$, $X_2$, $X_3$ as the intersection points of the cell interfaces with the edges of the same number, and $X_G$ the gathering point of the three cells of $K_h$. Let us finally note $XI_1$, $XI_2$, the intersection point of the geometry with the edges $e_1$ and $e_2$. The principle of the cut-cell method is to modify the already existing finite volume-cells so that the intersection points are present in the dual mesh while keeping consistency with the classic strategy for the areas where no intersection occur. The process is explained in **Algorithm** 5 and an example of cut-cell is given in Figure 12. $X_1$, $X_2$ and $X_G$ are modified so that the finite volume cells match with the embedded boundary while $X_3$ is kept intact to be consistent with a classic cut-cell in the neighboring triangle. Note that this algorithm is the same if only one ($P_3$,

for instance) or two ($P_1$ and $P_2$ for instance) point are covered by the geometry as it does not need to know *a priori* which ones are covered.

**Algorithm 5:** Two-dimensional cut-cell algorithm

**for** $K_h \in \mathcal{H}$ **do**
  **if** $(e_1, e_2) \in \mathcal{E}_h \times \mathcal{E}_h$ **then**
    $X_1 = XI_1$ ; $X_2 = XI_2$ ; $X_3 = \frac{1}{2}(P_1 + P_2)$ ; $X_G = (X_3P_3) \cap (X_1X_2)$

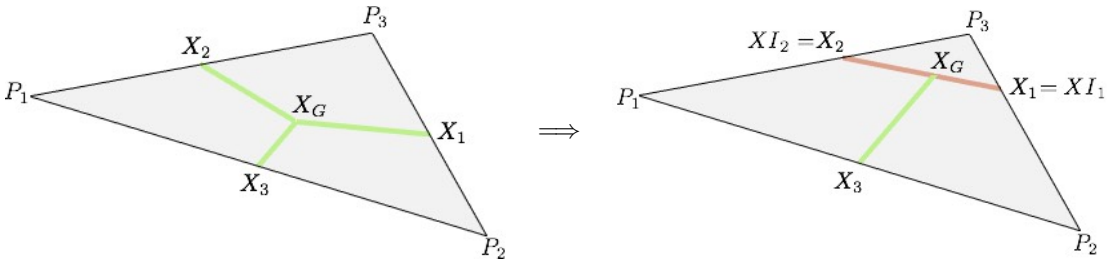



Figure 12: Modification of the finite volume cells in 2D with a cut-cell method in the classic case. In green, the cell interfaces. In red, the embedded geometry interfaces.

Then several particular cases exist. The problem of the corners has to be tackled as well as the problem of the double intersection on an edge. In these cases, the cut-cell is also particular and explained in Figures 13 and 14.

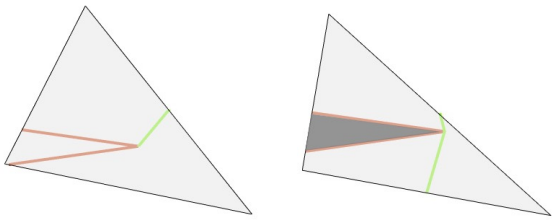


Figure 13: Two particular cases of cut-cell in 2D dealing with the corners. In green, the cell interfaces. In red, the embedded geometry interfaces.

In Figure 13, the case of the corner is handled. On the left, there are intersections with two different edges. The procedure simply consists in setting the corner of the embedded surface as a new $X_G$. This procedure is the same if one (convex corner) or two (concave corner) points are covered by the geometry as it does not need to know *a priori* which ones are covered. On the right, there is a corner issued from a double intersections. If all the three vertices of the triangle are not covered by the geometry, the finite volume cells are modified as depicted in Figure 13 (right) and some area of the triangle is erased. On the contrary, if the three vertices are covered, the corner is ignored.

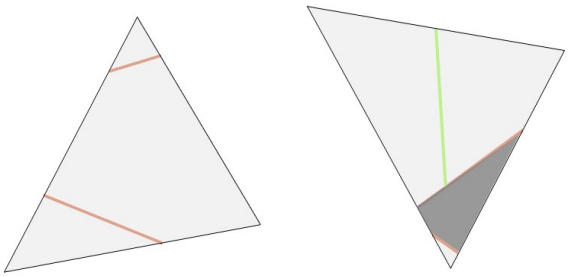


Figure 14: Two particular cases of cut-cell in 2D dealing with the double intersections. In green, the cell interfaces. In red, the embedded geometry interfaces.

In Figure 14, the case of the double intersections is handled. On the left, there is one intersections on two different edges and a double intersection on the last one. The procedure simply consists setting the finite volume cells according to the cut part of the triangle a vertex belong. This procedure is the same if one or two points are covered by the geometry as it does not need to know *a priori* which ones are covered. On the right, there are two double intersections. If all the three vertices of the triangle are not covered by the geometry, the finite volume cells are modified as depicted in Figure 14 (right) and some area of the triangle is erased. On the contrary, if the three vertices are covered, double intersections are ignored.

### 2.2.3 Three-dimensional cut-cell

In 3D, the idea behind the cut-cell is quite the same. First of all, a classical plane/tetrahedron intersection gives us two possibilities: a quadrilateral (this means that two points are covered by the geometry) or a triangle (this means that one or three points are covered). Like in 2D, the finite volume cells of the tetrahedron are locally modified to fit with the surface of intersection. For each face of the tetrahedron, there is either 2 or 0 intersections on the edges. Basically, for each face, it is possible to reproduce either the 2D classic cut-cell or the 2D classic median finite volume cells (in order to fit with the neighbors where no cut-cell has been done). Then, the gravity center of the tetrahedron is replaced by the gathering point of the four modified cells.
In the following algorithm, let us note $(e_i^j)_{i\in[1,3]}$, the three adjacent edges to a vertex $P_j$ and $X_i^j$ their middle. Let us also note $(F_i^j)_{i\in[1,3]}$ the three faces of the tetrahedron containing $P_j$ so that $F_1^j$ does not contain $e_i^j$ and $XF_i^j$, their gravity center and let us note $X_G$ the gravity center of the tetrahedron. Let us finally note $XI_i^j$, the intersection point (if any) of the geometry with the edges $e_i^j$. The principle of the algorithm is to set new values for $X_G$, $XF_j^i$ and $X_j^i$ in the tetrahedron so that the embedded boundary is well represented in the finite volume cells of the $P^i$. This process is summarized in **Algorithm** 6 and is illustrated in Figure 15. Like in the standard 2D case, this algorithm does not need to know *a priori* which vertices have been covered by the geometry.

In the same way as in 2D, some particular cases need to be treated. It might be interesting to take into account the corners, the required edges/ridges and the double intersections. These particular cases were not difficult to take into account in 2D, but it turns out to be pretty difficult to implement in 3D. That's why, it has been decided for the sake of simplicity to simplify these particular cases into a non intersection case or a classic intersection case.
From a more general point of view, the question of taking into the singularities of the geometry

**Algorithm 6:** Three-dimensional cut-cell algorithm

**for** $K_h \in \mathcal{H}$ **do**
  **for** $P_j \in K_h$ **do**
    **for** $F_i^j \in K_h$ **do**
      **if** $F_i^j \cap I_h = \emptyset$ **then** $XF_i^j = bary(F_i^j)$;
      **else**
        $XF_i^j = X_G^{2D\ cut-cell}(F_i^j)$
    **for** $e_i^j \in K_h$ **do**
      **if** $e_i^j \cap I_h = \emptyset$ **then**
        $X_i^j = XI_i^j$
      **else**
        $X_i^j = middle(e_i^j)$
    $\text{Cell}(P_j) = [P_j, X_1^j, XF_1^j, X_2^j, X_3^j, XF_2^j, X_G, XF_3^j]$

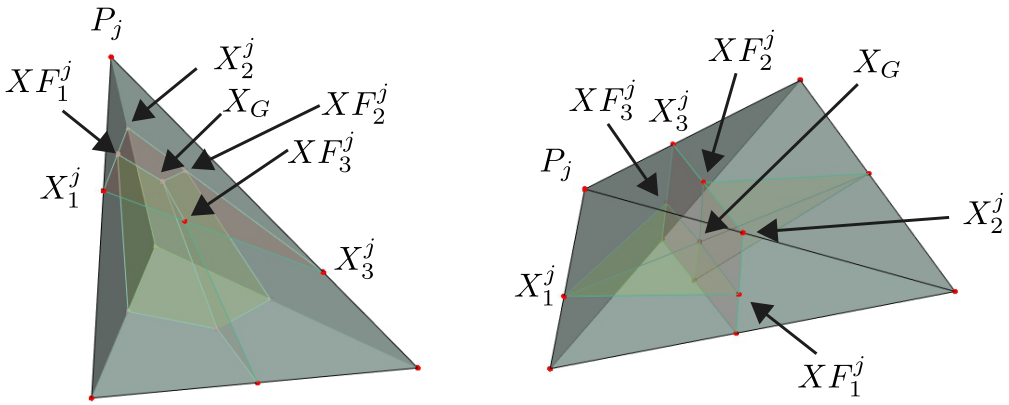



Figure 15: Two classic cases of cut-cell in 3D.

is more a meshing-related problem than a solver-related. Thus, it should not be dealt with by a solver.

## 2.3 Modification of the numerical scheme

In this section, we give the modifications required to make the Embedded Boundary Method compatible with the numerical scheme of an already existing vertex-centered Finite-Volume solver. In particular, we show how the embedded intersection impacts the already existing scheme and then we explain how we modify the numerical flux to create an *embedded* boundary condition.

### 2.3.1 Impact on the vertices and edges covered by the geometry

As we are in the case of the Euler equations, we can directly enforce the solution vector $W$ attached the vertices covered by the geometry. It is forced to reference value which is the value at the infinity in all our cases. This implies a change in the flux computation for an edge totally

covered by the geometry (we recall that $\mathcal{D}_h$ is the set of vertices covered by the embedded geometry):

$$\Phi_{ij} = 0 \text{ if } (P_i, P_j) \in \mathcal{D}_h.$$

In the case of an implicit advance in time, a change in the differentiation of the flux is required for the edges where at least one vertex is covered:

$$\frac{\partial \Phi_{ij}(W_i^n, W_j^n, \mathbf{n}_{ij})}{\partial W_j} = 0 \text{ if } P_j \in \mathcal{D}_h.$$

In addition, the identity matrix is forced in the implicit linear system for the covered vertices. This means that the interactions of these points are canceled.

### 2.3.2 Impact on the gradient computation

The implementation of a finite volume scheme along with a MUSCL extrapolation relies on the computation of gradients. Nonetheless, gradient computation can be impacted by the embedded intersection. Gradient computation is based on some informations provided by the upwind and downwind elements. If one of these elements is intersected by the embedded geometry, it is required to ignore the classical gradient computation and to compute a surrogate gradient as it is already the case with the boundary vertices. In the same way, nodal gradients, required for a V6-scheme, are modified as follows ($d$ is the dimension):

$$(\nabla W)_{P_i} = \frac{1}{(d+1)|C_i|} \sum_{\substack{K \in C_i \\ K \notin \mathcal{D}_h}} |K| (\nabla W)_K.$$

### 2.3.3 Implementation of the embedded boundary condition

To simulate, the presence of the embedded boundary, we induce two types of slip conditions: a Riemann slip condition and a classic slip condition.

**Embedded Riemann slip condition** The numerical flux is changed for the edges intersected by the geometry. In the first hand, a Riemann slip condition is weakly imposed. For an edge $e = [P_i, P_j]$ intersected by the embedded geometry, it writes[1]:

$$\Phi_{Embed\ Slip\ Riemann} = \Phi_{ij}(W_i, \overline{W_i}, \mathbf{n}_{ij}) \text{ if } P_j \in \mathcal{D}_h \text{ and } P_i \notin \mathcal{D}_h \text{ (for instance)}, \qquad (17)$$

The state $\overline{W}_i$ is the mirror state of $W_i$, associated to $P_i$. Thanks to this flux, a Riemann slip boundary condition is induced on the embedded surface.

**Differentiation for implicit advance in time.** In order to drastically reduce CPU time, the embedded boundary method has been extended to the implicit solver for the Euler equations. The jacobian of the flux for an intersected edge $e = [P_i, P_j]$ is modified as follows:

$$\frac{\partial \Phi_{ij}(W_i^n, W_j^n, \mathbf{n}_{ij})}{\partial W_i} = \frac{\partial \Phi_{ij}(W_i^n, \overline{W_i^n}, \mathbf{n}_{ij})}{\partial W_i} \text{ if } P_j \in \mathcal{D}_h \text{ and } P_i \notin \mathcal{D}_h \text{ (for instance)}.$$

This way, a Riemann slipping boundary condition is induced.

[1] Note that as the embedded interface $\Gamma$ has become the interface of $e$ by means of the cut-cell approach, we have $\mathbf{n}_\Gamma = \mathbf{n}_{ij}$. Consequently, the boundary is naturally recovered with the intersected edges.

**A more accurate implementation** Mirror boundary condition is a first order approximation of the numerical flux done in a very critical area. To improve the accuracy, it seems natural to extrapolate the solution at the intersection between the embedded interface and the edge. To this end, a MUSCL type reconstruction method [50, 20] is used. To improve the accuracy, it seems natural to extrapolate. If we note $\overline{P}_i$ the symmetric point of $P_i$ with respect to the intersection point $XI$ on the edge $P_iP_j$ (see Figure 16), we have for an intersected edge:

$$W_{XI} = W_i + \frac{1}{2}(\nabla W)_{ij}.\overrightarrow{P_i\overline{P_i}} = W_i + (\nabla W)_{ij}.\overrightarrow{P_iXI}, \tag{18}$$

and

$$\Phi_{Embed\ Slip\ Riemann} = \Phi_{ij}(W_{XI}, \overline{W_{XI}}, \mathbf{n}_{ij}).$$

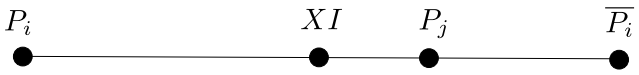



Figure 16: Construction of the symmetric point used for the MUSCL extrapolation for an intersected edge.

The centered gradient computation is made by replacing $W_j$ by $\overline{W_i}$ and $P_j$ by $\overline{P_i}$ in every equation. Then, this second order extrapolation is locally coupled with a Dervieux or Piperno (if applicable) limiter. In Figure 17, we can see two simulations of the embedded boundary method in the case of a potential flow around a circle at Mach 0.1 on a mesh of 5 000 vertices. An HLLC solver is used along with a V6 scheme and no limiter. The implicit advance in time is used. This improved *embedded* boundary condition is applied around the detected circle and the result is shown after 500 iterations for two simulations, one without a MUSCL reconstruction on the boundary (left) and one with a MUSCL reconstruction on the boundary (right). A slight improvement of the results can be noticed when using the MUSCL, particularly with the contour lines.

**Embedded slip condition (weak form)** In the previous section, the induced slip condition is a Riemann slip condition. This boundary condition is more stable but less accurate than a slip boundary condition (where $\mathbf{U}.\mathbf{n} = 0$ is enforced weakly), that is why another flux corresponding to a slip condition in weak form is proposed:

$$\Phi_{Embed\ Slip} = (0, p_{XI}\mathbf{n}, 0)^T,$$

where $p_{XI}$ is the pressure related to $W_{XI}$ computed thanks to Equation (18). For the implicit advance in time, the differentiation of the embedded Riemann slip condition is used.

## 2.4 Numerical results

In this section, we show the numerical results of flow simulations performed using the embedded boundary method.

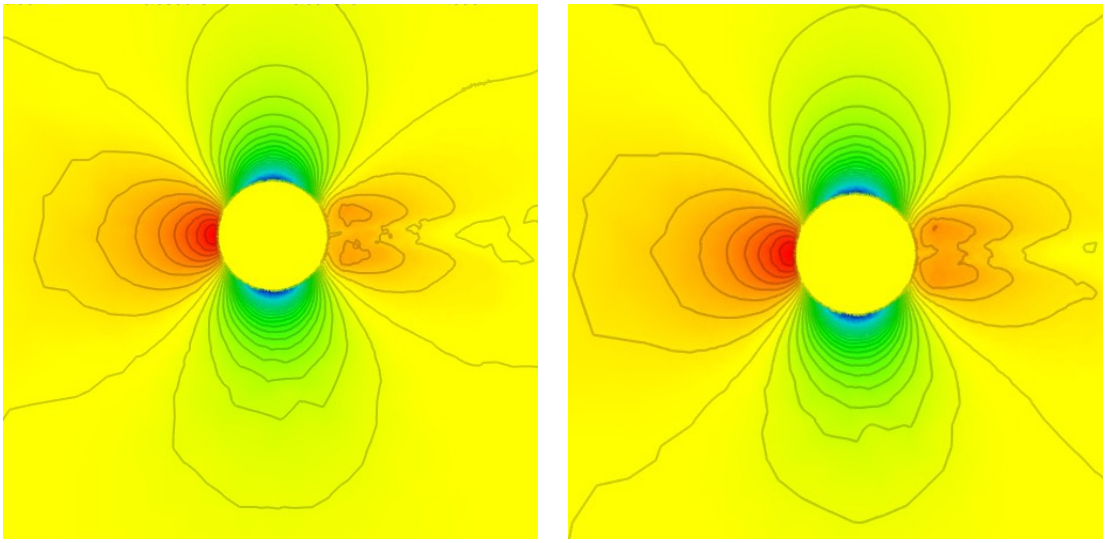


Figure 17: Comparison of the solution without MUSCL on the boundary (left) and with MUSCL on the boundary (right).

### 2.4.1 On the geometrical accuracy of the embedded geometry

One of the major concerns of numerical computations is to give geometries that are consistent with the analytical definition of the initial geometry. In particular, the mesh size $h_{I_h}$ from $I_h$ needs to be locally consistent with the mesh size $h_{\mathcal{H}}$ from the computing mesh $\mathcal{H}$. Moreover, size $h_{I_h}$ should be such that the curvature of the objects is preserved. For instance, a circle does not have any discontinuity on its outward normal whereas a regular polygon, one of its possible discretization, has some. More precisely, if $h_{I_h} \leq h_{\mathcal{H}}$, there is not any effect of the discretization of the embedded boundary on the numerical solution (see Figure 18 bottom) but if $h_{I_h} \geq h_{\mathcal{H}}$, the geometric representation brings some undesired effects in the solution (see Figure 18 top). In the example Figure 18, the embedded geometry is seen by the flow solver as a piecewise linear wing whereas, the wing geometry is $C^\infty$. Consequently, every discontinuity (in the tangent) creates vortices that are propagated in the domain. This is a pure artifact due to an inconsistent resolution of the true geometry.
As we will see in the following sections, mesh adaptation can highly increase the resolution of the mesh next to the boundary, therefore a reliable discretization of the boundary is needed for these simulations.

### 2.4.2 Steady flow simulations

To test the reliability of the code, several standard steady cases have been tested with this method. Each case compares a body-fitted simulation with its embedded counterpart.

**Circle** As a validation test case, the potential flow over a circle presented in Section 1.1.8 is used. To compute this flow, an HLLC solver is used along with a V6 scheme and no limiter. An implicit advance in time is used. The studied boundary condition on the circle is a slipping boundary condition. The results are shown after 1000 iterations. In Figure 20, two embedded cases are studied. The first one is a toy problem as it consists in explicitly representing the geom-

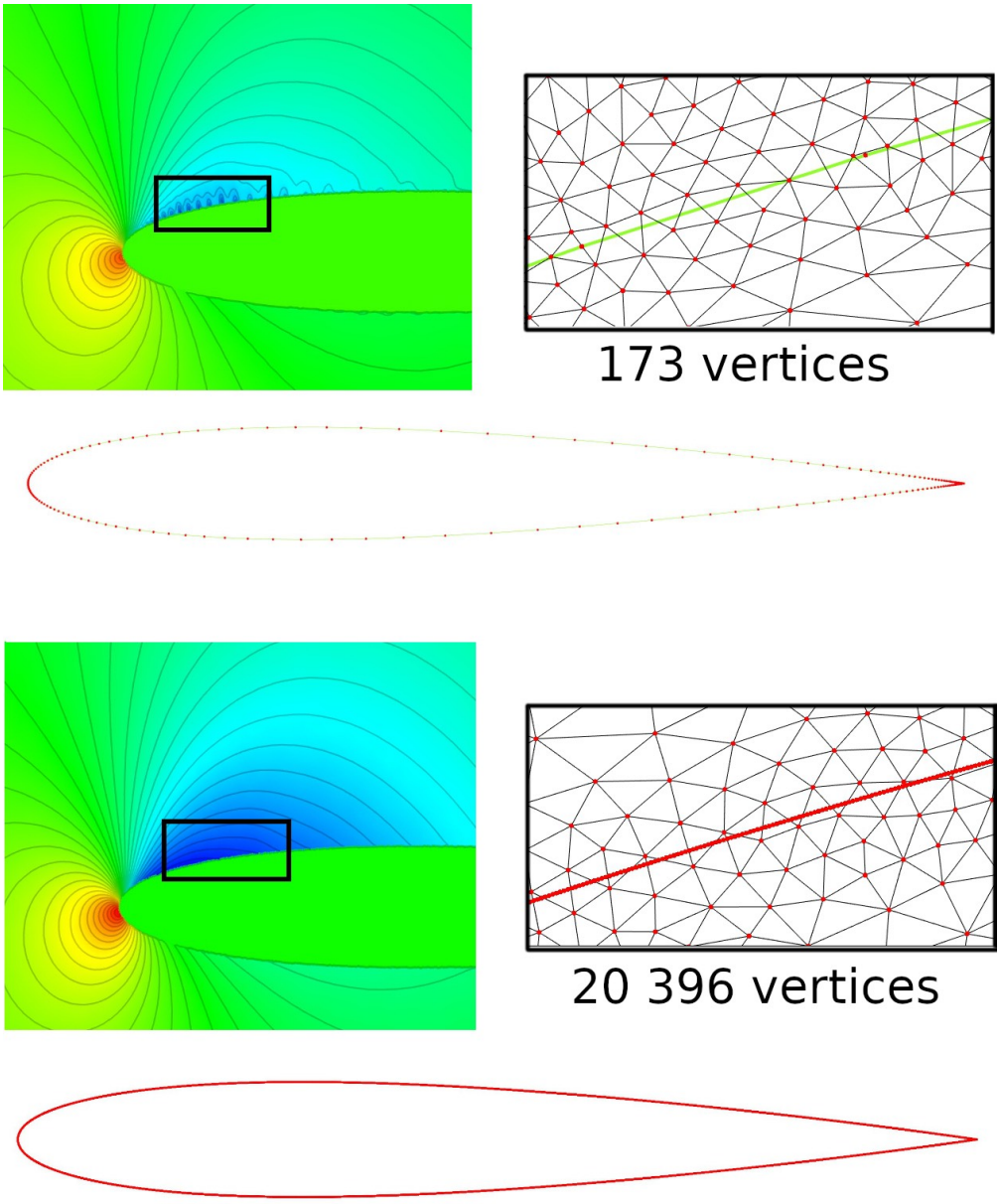



Figure 18: On the left, computed solution after a mesh adaptation process with a coarse (top) and fine (bottom) representation of the geometry. On the right, zoom on the black square region. The extremities of the edges are the red points and the embedded boundary is in green. The whole embedded mesh along with the number of vertices are also given.

etry in the background mesh. This is done so that we can make sure that the flow solver behaves as expected like in the body-fitted case. The second one is a standard embedded case where the geometry is not explicitly defined in the mesh. The size of the meshes is of approximately

36 000 vertices and the embedded circle is defined with a uniform distribution of 189 vertices. The results show that the embedded boundary method degenerates into the body-fitted one when the edges of the embedded and the background mesh match. This is confirmed by the plot of $C_p$ curves (Figure 19) where both cases cannot be distinguished. However, the other simulation shows the drawback of the embedded approach for meshes not fitted for the geometry. Both solution and $C_p$ curves show that the behavior is the one expected but that a loss of accuracy in the vicinity of the geometry is present. It can be noted that the solution is highly impacted by the point distribution in the background mesh. Indeed, the induced intersected geometry is less well represented where the mesh is coarse which provides the loss of symmetry in the profile around the circle.

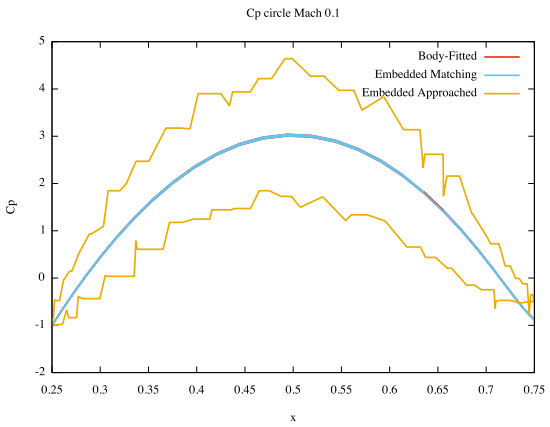



Figure 19: Computed $C_p$ of a potential flow around a circle. In red, the body-fitted case. In blue, the embedded matching case. In yellow, the standard embedded case.

**NACA0012 airfoils** Using this method, flow simulations at three different regimes around NACA0012 airfoils have been done. The used meshes are shown in Figure 21 and the displayed field is the flow density. To compute these flows, an HLLC solver is again used along with a V4 scheme and a Piperno limiter. An implicit advance in time is used. The studied boundary condition on the airfoil is a slipping boundary condition. For all these simulations, the same meshes are used for all the regimes. A mesh of 4812 vertices for the embedded case and a mesh of 5041 vertices for the body-fitted case. The embedded geometry consists in a mesh of 20 396 vertices. For these meshes the treatment of the embedded geometry requires a CPU time during the preprocessing of the same order as one iteration of the flow solver.

**Subsonic case.** The first case is a subsonic flow (Mach 0.63) around NACA0012 airfoil with an angle of attack of 2 degrees. For this case, the solver is run until convergence is reached with a residual lower than $10^{-8}$. For the body-fitted case, it is reached after 280 iterations whereas it is reached 1165 iterations for the embedded case. This phenomenon can be explained by a simple reason (among others): the minimal time step of the flow solver in the body-fitted case is of 0.00103 whereas it is of 0.00023 in the embedded case. Indeed, the cut-cell performed in the preprocessing step usually reduces the size of the finite volume cells. This reduction thus provokes an increase of the number of time steps.

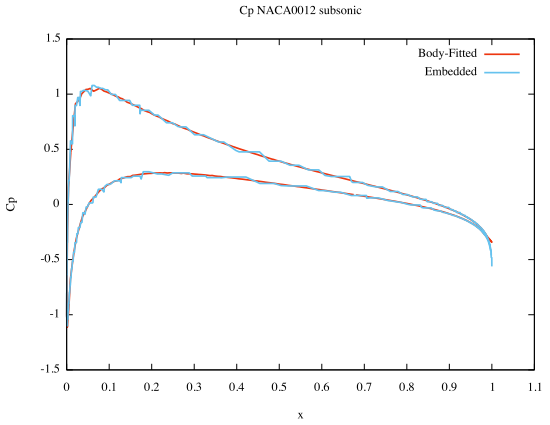



Figure 22: Computed $C_p$ of a subsonic flow around a NACA0012 airfoil.

The obtained results are quite similar in trend (see Figure 23) and this can be confirmed by the

$C_p$ (Figure 22) curve around the airfoil even if we can see that the curve oscillates a lot in the embedded case. However, note that a relative error of 54% is obtained for drag computation which is not surprising as it is based on an integration over the boundary of the object.

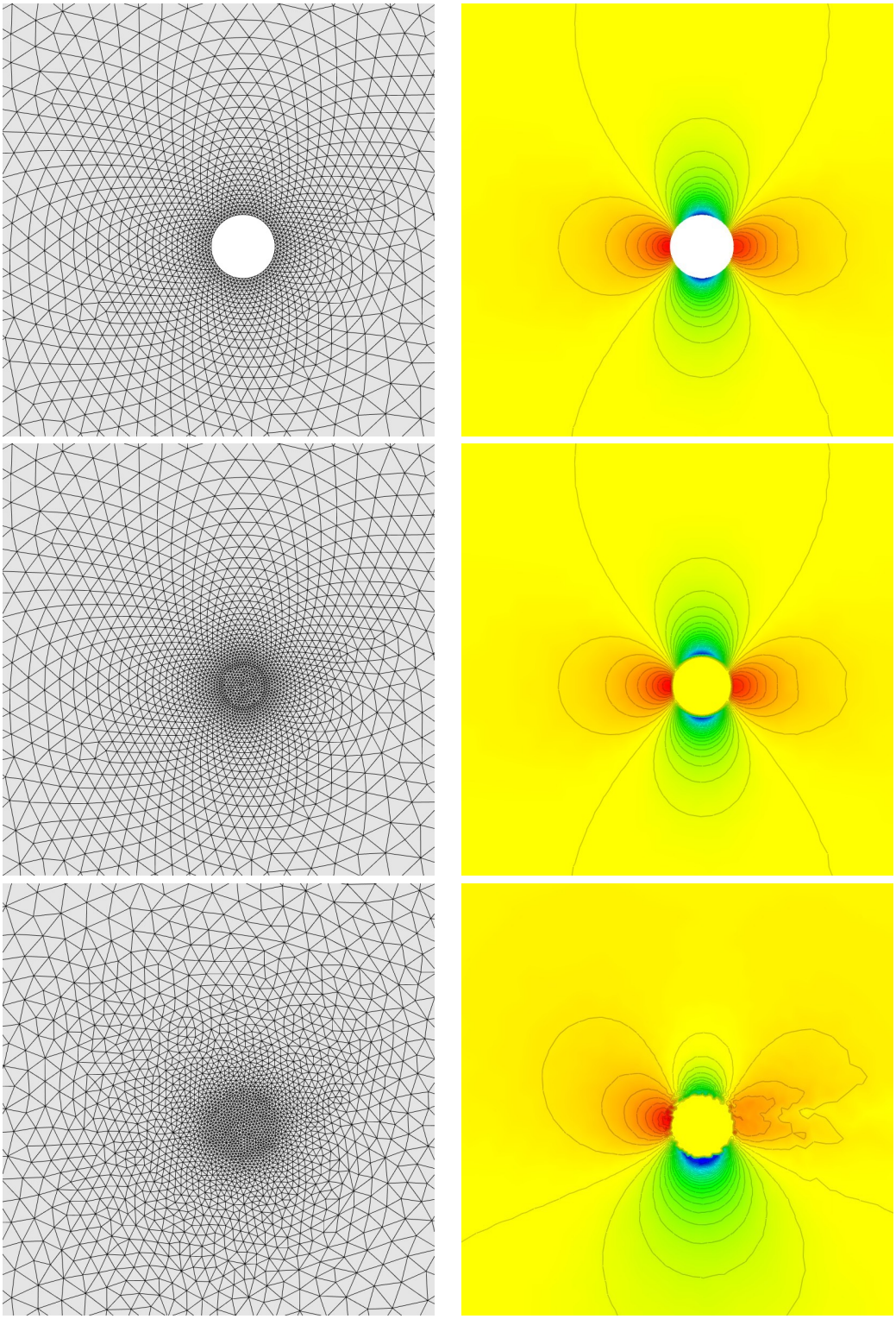


Figure 20: Simulation of potential flow around a circle. Mesh (left) and density field (right) for three cases: body-fitted (top), embedded with exact representation of the circle (middle) and embedded without explicit representation of the circle (bottom).

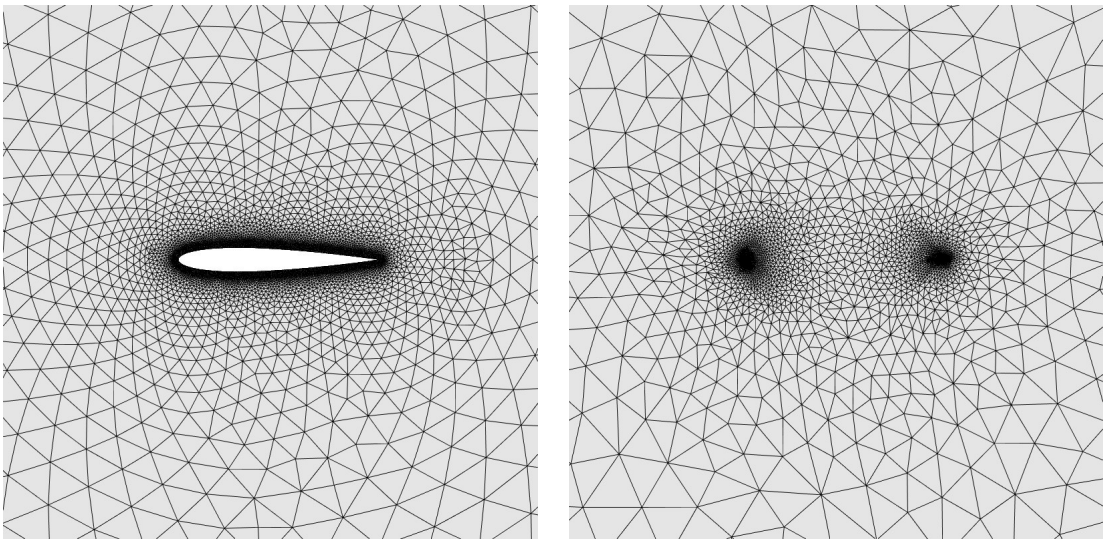


Figure 21: Body-fitted (left) and embedded (right) meshes used for the simulations of the flow around a NACA0012 airfoil.

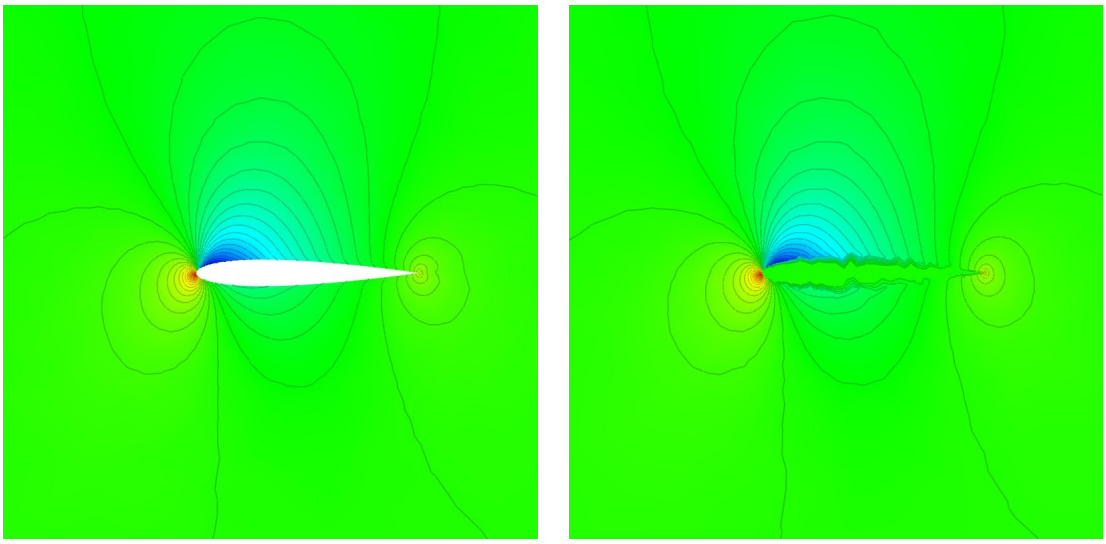


Figure 23: Body-fitted (left) and embedded (right) simulation of a subsonic flow around a NACA0012 airfoil.

**Transonic case.** The second case is a transonic flow (Mach 0.85) around a NACA0012 airfoil with an angle of attack of 1.25 degrees. For this case, the solver is stopped after 500 iterations. The minimal time step of the flow solver in the body-fitted case is of 0.00012 whereas it is of 0.000029 in the embedded case. This is again a consequence of the cut-cell performed in the preprocessing step.
The obtained results are quite similar in trend (see Figure 25) and this can be confirmed by the $C_p$ (Figure 24) curve around the airfoil. Again, an oscillation is seen with embedded case. Note that a relative error of 3% is obtained for the drag computation.

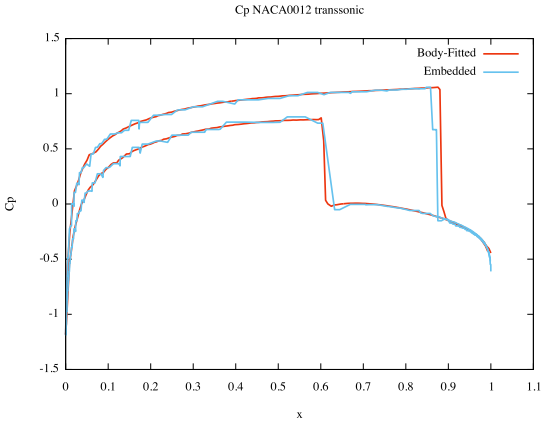



Figure 24: Computed $C_p$ of a transonic flow around a NACA0012 airfoil.

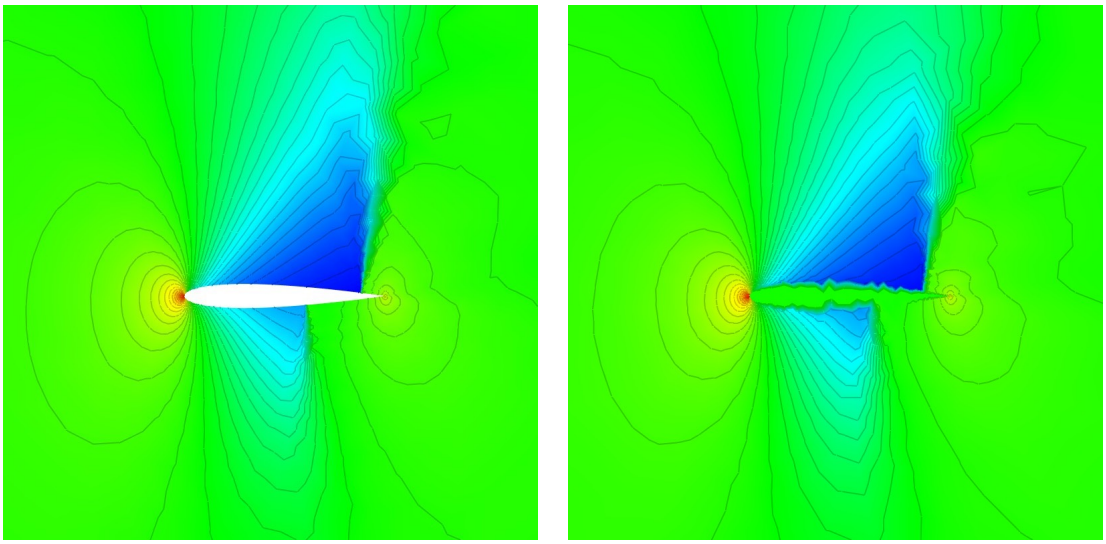


Figure 25: Body-fitted (left) and embedded (right) simulation of a transonic flow around a NACA0012 airfoil.

**Supersonic case.** The last case is a supersonic flow (Mach 2) around a NACA0012 airfoil with an angle of attack of 0 degree. For this case, the solver is stopped after 500 iterations. The minimal time step of the flow solver in the body-fitted case is of 0.000185 whereas it is of 0.000037 in the embedded case. This is again consequence of the cut-cell performed in the preprocessing step.

The obtained results are quite similar in trend (see Figure 26) and this can be confirmed by the $C_p$ (Figure 27) curve around the airfoil. Again, an oscillation is seen with embedded case as well as a greater dissipation of the supersonic shockwaves. Note that a relative error of 0.1% is obtained for the drag computation.

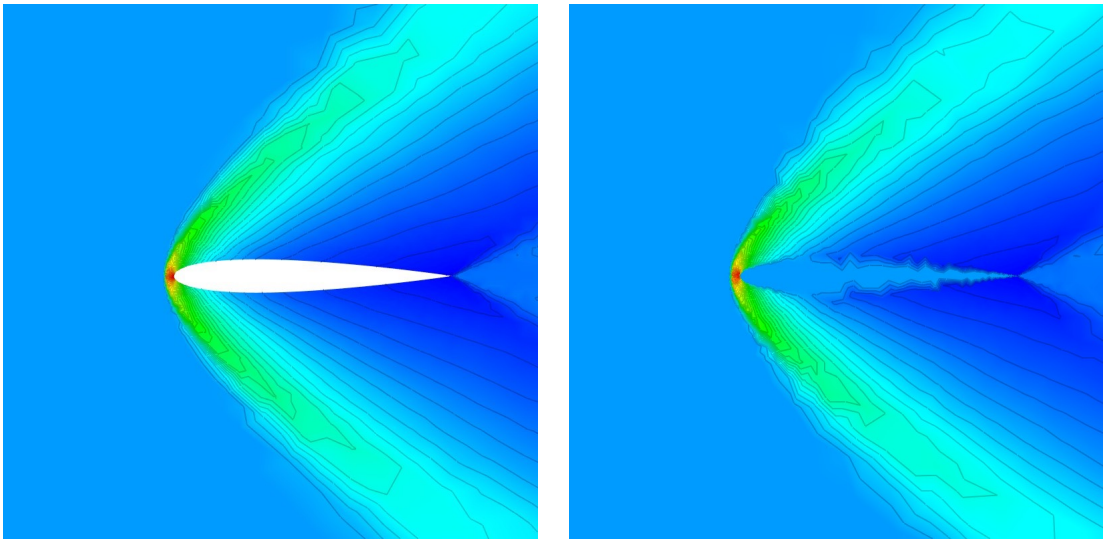


Figure 26: Body-fitted (left) and embedded (right) simulation of a supersonic flow around a NACA0012 airfoil.

In all the cases, the global trend of solution and contour lines are quite similar even if a lot of spurious oscillations occur all along the $C_p$ curve which leads to high differences in drag and lift computations. Also, other problems can be listed. First, the boundary of the airfoils is not well represented. Second, the point distribution of the mesh used for the embedded simulations is done to make sure something is captured by the tail of the airfoil in the first place, but it requires to know a bit the embedded object in the mesh generation process. One solution to overcome both of these issues is to use the iterative process of mesh adaptation. This will be done in one of the following sections.

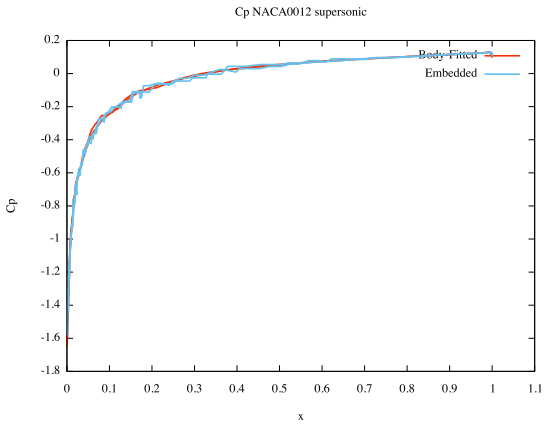



Figure 27: Computed $C_p$ of a supersonic flow around a NACA0012 airfoil.

**Supersonic notional missile** In the three-dimensional case, we simulate a supersonic (Mach 1.6) flow around a notional missile (see Figure 28) with a zero incidence (angle of attack of 0 degrees).The main difficulty with this geometry is the very thin winglets that are challenging for the embedded method. The used meshes with the results are shown in Figures 29 and 30. The displayed solution field is the flow density. An implicit advance in time is used. To compute these flows, an HLLC solver is again used along with a V6 scheme and a Dervieux limiter. The studied boundary condition on the airfoil is a slipping boundary condition. Two meshes of 90 078 and 784 847 vertices for the embedded case and a mesh of 199 342 vertices for the body-fitted case are considered. The embedded geometry is composed of 10 707 vertices in both cases. An analysis of the meshes shows a difference in the mesh density. This difference is explained because

small sizes have to be satisfied near the missile in the body-fitted case whereas, the generated mesh in the embedded case does not have *a priori* knowledge of the location of the object. The analysis of the obtained solution shows that a trend of the global behavior is visible but the embedded approach does not capture the physics due to the lack of accuracy in the final representation and even if the used mesh is relatively fine (784 847 vertices). By analyzing the result of the detection by the CFD solver, we notice that in both cases the shape of the missile is totally lost, especially its winglets. The use of mesh adaptation seems inevitable to recover the geometry in the embedded case without huge costs in preprocessing. Note that like in 2D, the treatment of the embedded geometry with this mesh requires a CPU time of the same order as one iteration of the flow solver. Finally, the analysis of the time-steps in this case is not relevant as the size maps are not consistent between the embedded and the body-fitted case.

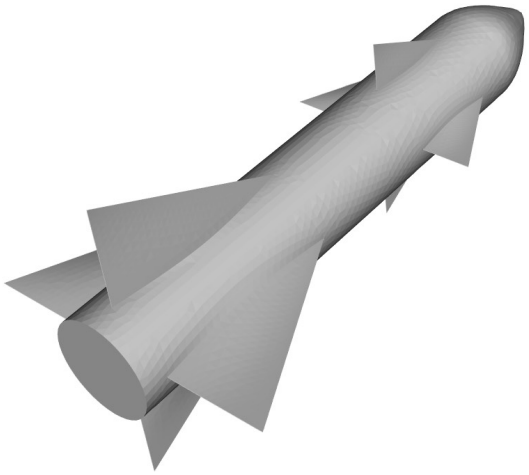


Figure 28: Geometry of the studied notional missile.

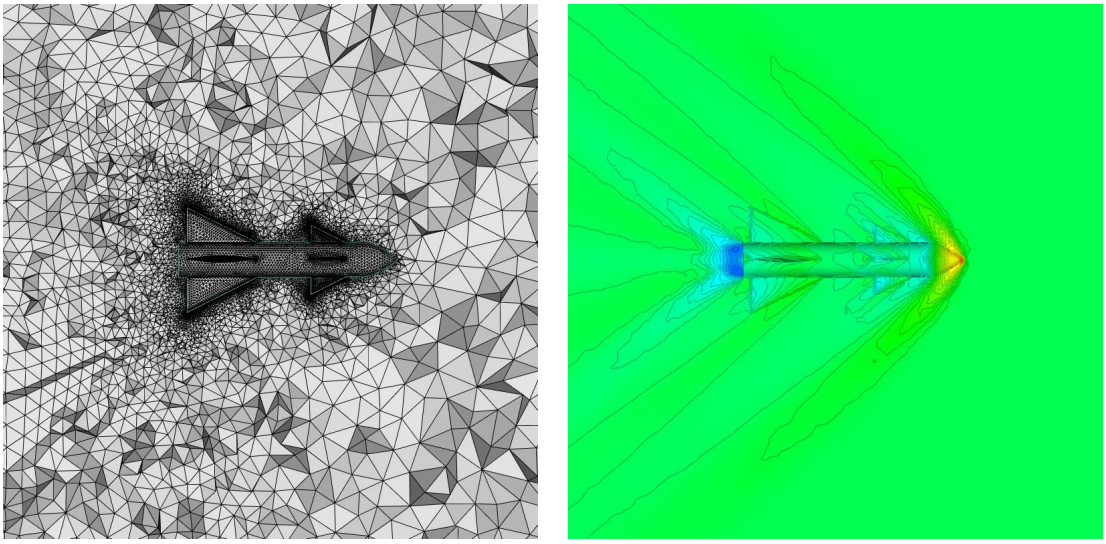


Figure 29: Supersonic flow around a notional missile. Mesh (left) and density solution field (right) for the body-fitted simulation

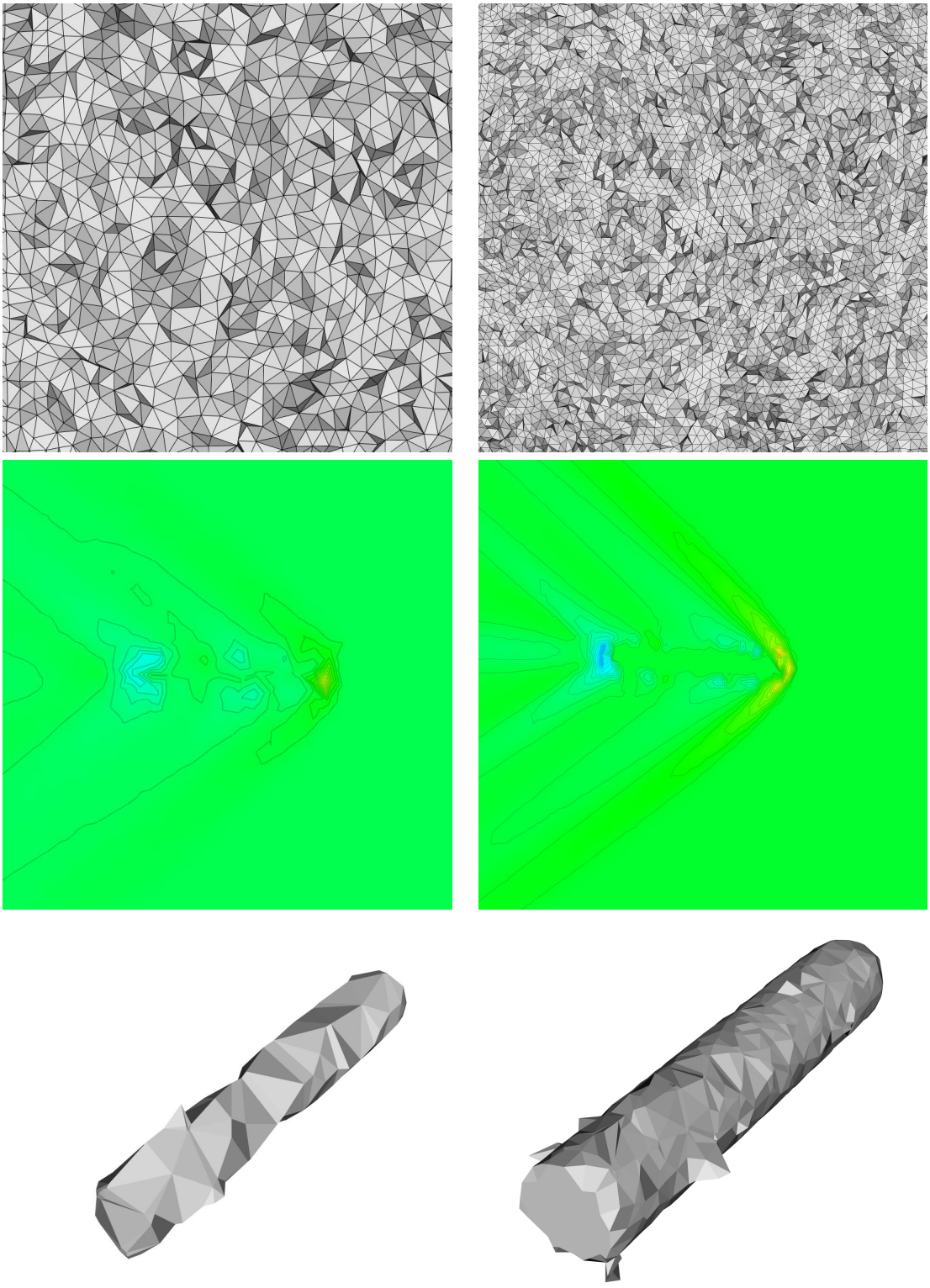


Figure 30: Supersonic flow around a notional missile. On the left, embedded simulation on a coarse mesh of 90 078 vertices and on the right, simulation on a mesh twice finer of 784 847 vertices. Top, volume meshes used, middle, solutions computed on these meshes and bottom, the geometry as it is seen by the CFD solver in both cases.

### 2.4.3 Unsteady flow simulations with fixed geometry

Using the same geometrical approach as with steady flows, a blast simulation has been done as unsteady flow simulation. The considered domain is a square of size $[-10, 10] \times [-10, 10]$ with sixteen circle of radius 0.1. The blast is ignited like a Sod shock problem inside a circle of radius 0.5 located in the middle of the square: the initial (dimensionless) density is of 1 inside the circle and 0.125 outside and the initial (dimensionless) pressure is of 1 inside the circle and 0.1 outside. For the resolution of the unsteady Euler equations, an HLLC solver is used and is coupled with an explicit advance in time by means of a Runge-Kutta scheme of order 2. Wall boundary conditions are applied on the edge of the domain and slipping boundary condition is applied on the circles inside the domain. Finally, the total physical time of the blast is of 6s. The body-fitted mesh contains 4418 vertices and the mesh used for the embedded simulation is composed of 5050 vertices. Note that the mesh of the embedded case is non uniform and adapted to the position of the embedded circles. The results are shown in Figure 32 where the used meshes and two snapshots at $t = 1.5$ and $t = 4.0$ are given in both embedded and body-fitted cases. With only a dozen of points covered by each embedded circle, the solver is able to detect them and apply properly the boundary condition like in the body-fitted case. A zoom around one of the circles is done in Figure 31. In the embedded case the CPU time required in preprocessing is approximately of the same time of one solver iteration and is clearly negligible beyond the total time required for the simulation. In the body-fitted case, the simulation requires 412 iterations with time steps that are approximately of order $10^{-2}$ and in the embedded case, the simulation requires 352 iterations with time steps that are approximately of order $10^{-2}$. In other words, the embedded simulation has an equivalent behavior as its body-fitted counterpart.

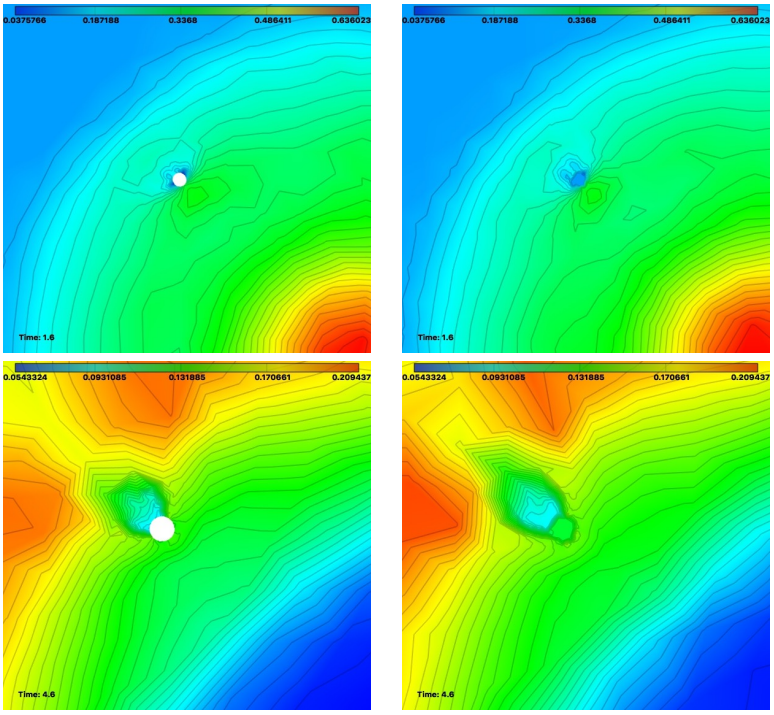



Figure 31: Zoom of the density of field around one of the circles involved in the blast of Figure 32. Left, the body-fitted case and right, the embedded case. The solutions are at two different times: $t = 1.8$ and $t = 4.6$.

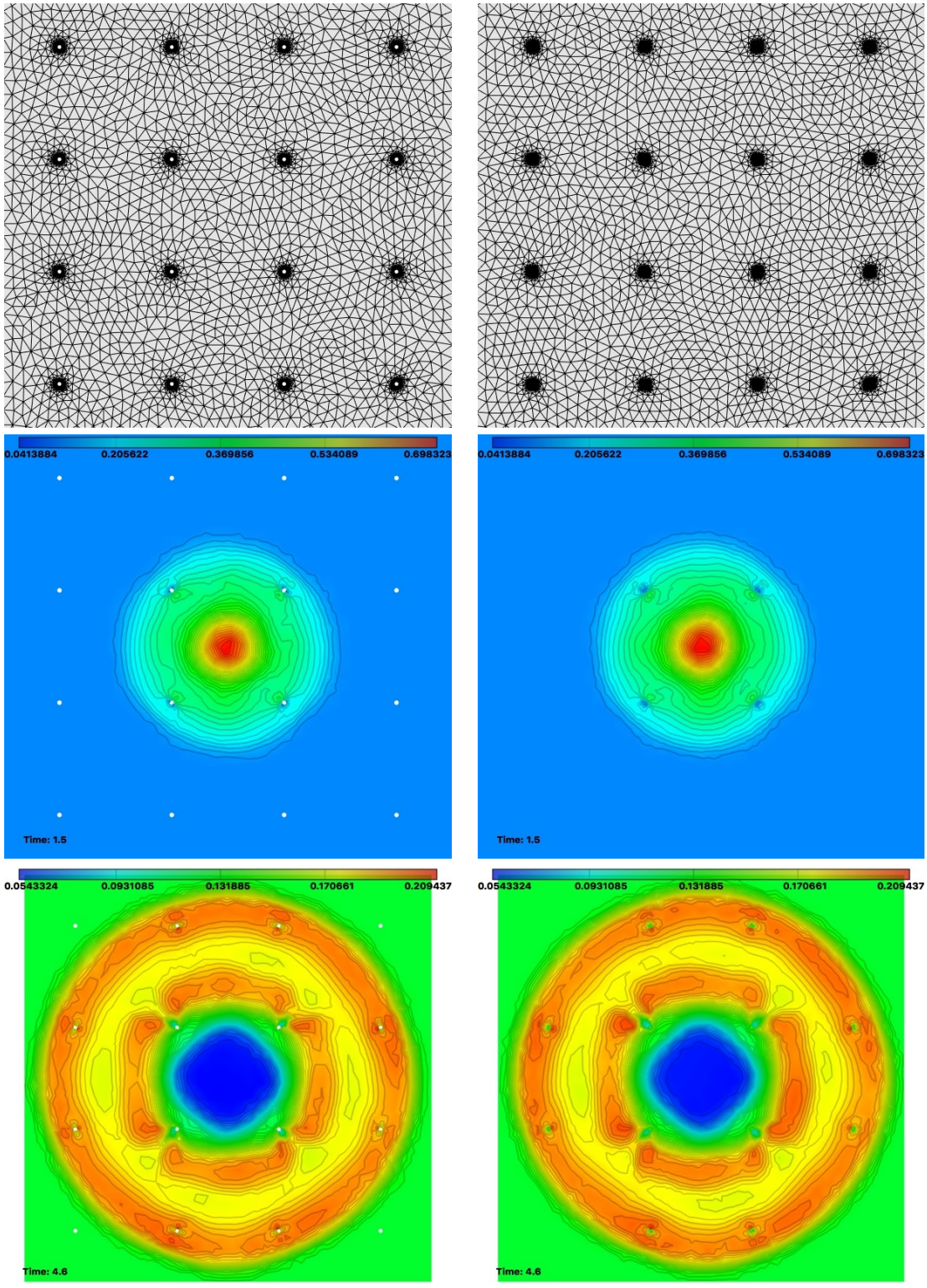
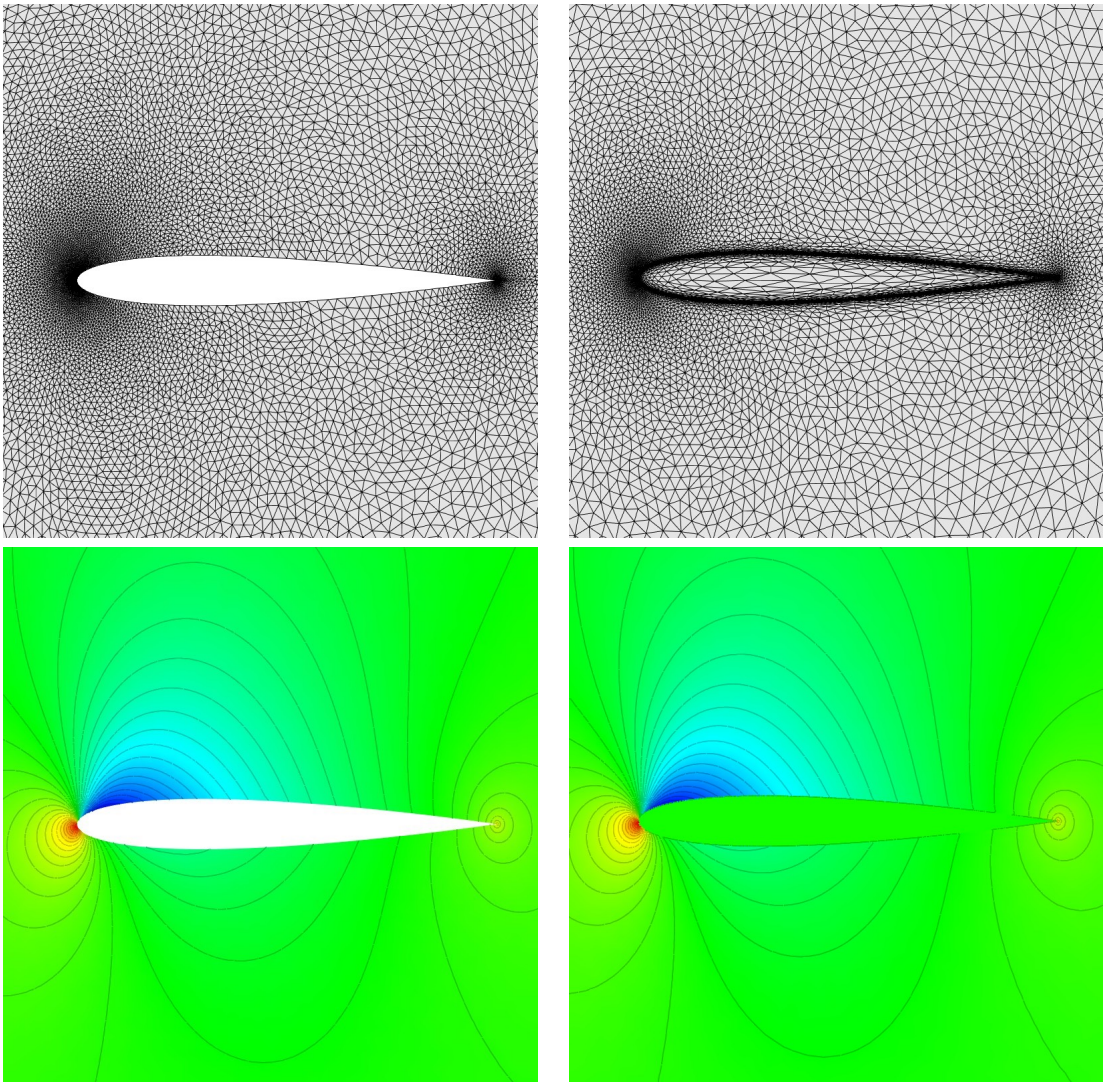


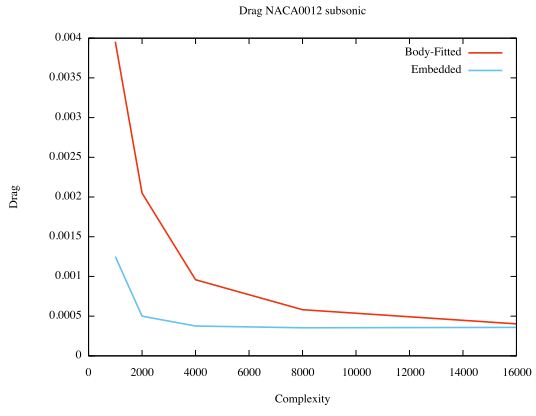

Figure 32: Blast simulation with a fixed geometry. Left, the body-fitted case and right, the embedded case. Top, the considered meshes and thereafter, the associated solutions at two different times: $t = 1.8$ and $t = 4.6$ (density fields).

### 2.4.4 Mesh adaptation for steady flow simulations with embedded geometries

Steady flow simulations with embedded geometries captures naturally the embedded geometry in the mesh thanks to the boundary conditions. Indeed, if a vertex is detected as covered by the geometry, a constant value, the value at infinity, is assigned to this vertex. Most of the time, this creates a discontinuity (unless the flow is at rest) of the solution around the embedded geometry. Also in CFD, the computation of aerodynamical coefficients around geometries (drag, lift, ...) is of major interest as they quantify the quality of a design from an engineering point of view. Consequently, an accurate and reliable mesh that captures the geometry well is desirable. As the position of the geometry is not necessarily known in the first hand, mesh adaptation becomes highly interesting. The idea is to start from a uniform and/or coarse mesh and to apply the adaptation loop several times. Thanks to the gap naturally created around the geometry by the solver, the mesh is refined around the location of the embedded geometry. Once adaptation process is done, the geometric error due to embedding process is reduced, residual converges faster and solution looks really close to the body-fitted one.
On some of the previous cases, adaptation process has been performed. The general process is the following: for a given complexity, the flow solver is called 5 times and after each call, a feature-based metric using the sensor field is deduced (see Section 1.2). Using this and the input complexity, a new mesh is generated and the process continues. The method is applied in 2D on two different types of simulations around a NACA0012 airfoil : a supersonic and a subsonic flow as each of them highlights various phenomena involved in CFD. Then the three-dimensional test-case of the notional missile is performed in the supersonic case. In all cases, the sensor used for the mesh adaptation is the local mach number.

**Subsonic and supersonic NACA0012** For the case of the NACA0012 airfoil, the considered complexities are 1 000, 2 000, 4 000 8 000, and 16 000. In these simulations, the studied flow and the solver characteristics are exactly the same as in Section 2.4.2. The results are given in Figures 33 and 34 where the last results at complexity 16 000 are shown as well as the comparison of both curves and the evolution of the computed drag all along the adaptation process.
In both cases, the observations are the same: the adaptation process captures the same physical features in the body-fitted case and in the embedded one. This is confirmed by the plot of $C_p$ curves which shows no real difference between both curves. It is also corroborated by the plot of the computed drag all along the process which shows that both body-fitted and embedded case converge to the same result. Moreover, for the case of the embedded simulation, it is clear that the geometry is quickly and well captured in the adaptation process. This explains notably that for a given complexity, the computed drag seems more accurate than the body-fitted one as larger number of degrees of freedom lies on the boundary.





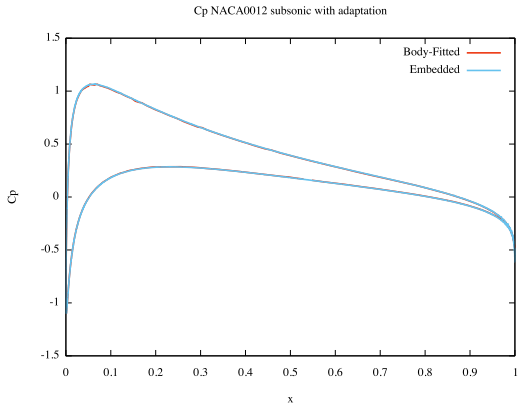




Figure 33: Subsonic flow around NACA0012 airfoils. Left, the adapted body-fitted case. Right, the adapted embedded case. Top, the adapted meshes obtained at the last complexity. Middle, the associated solutions density fields. Bottom, the evolution of the drag all along the adaptation process (left) and the $C_p$ curves at the last complexity (right).

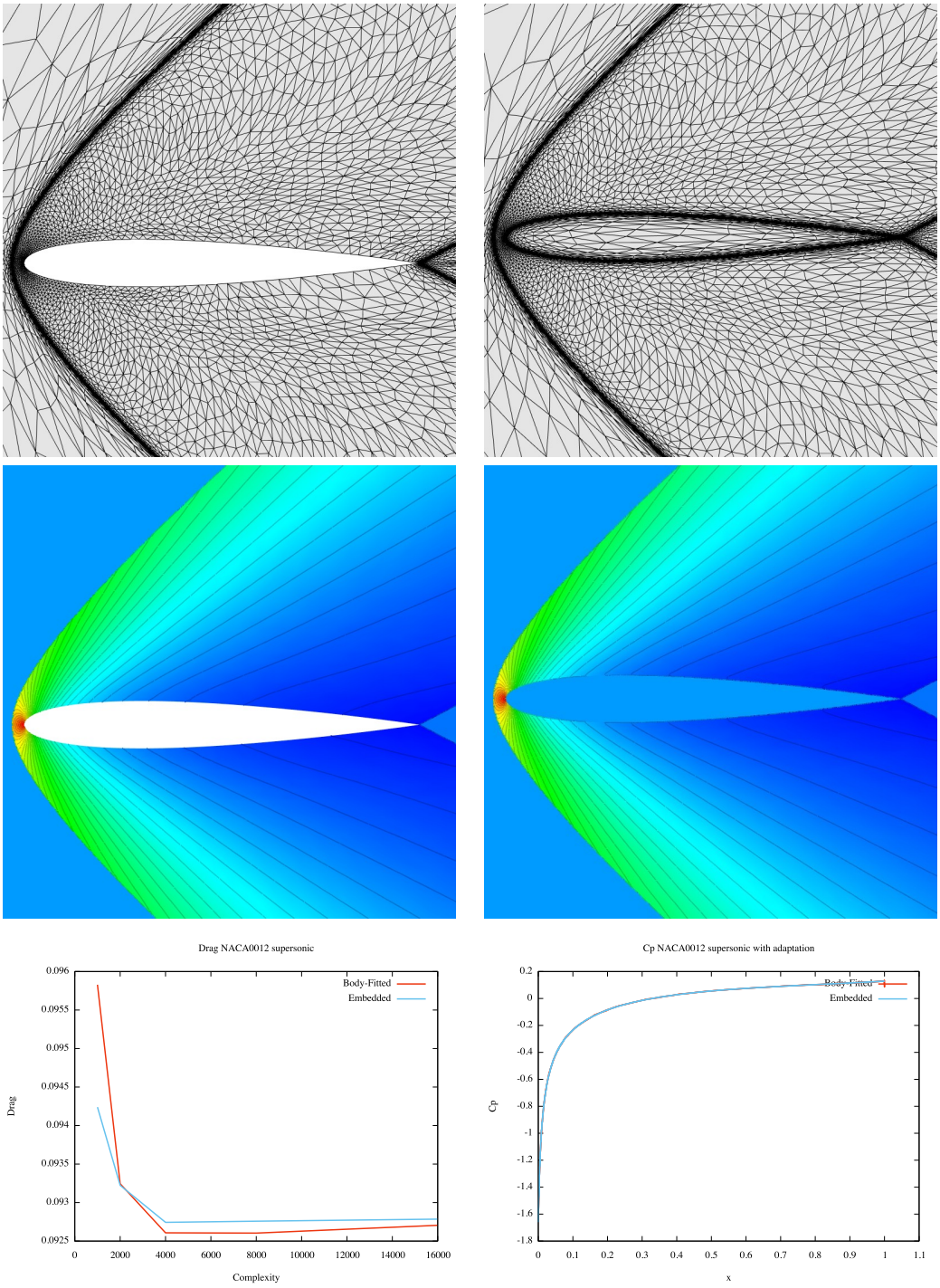



Figure 34: Supersonic flow around NACA0012 airfoils. Left, the adapted body-fitted case. Right, the adapted embedded case. Top, the adapted meshes obtained at the last complexity. Middle, the associated solutions density fields. Bottom, the evolution of the drag all along the adaptation process (left) and the $C_p$ curves at the last complexity (right).

**Supersonic notional missile** For the case of the notional missile, the used complexities are 1 000, 2000, 4 000 8 000, 16 000, 32 000, 64 000 and 128 000. In this process, the studied flow and the solver characteristics are exactly the same as in Section 2.4.2. The numerical results are given in Figures 36 and 37 where the last results at complexity 128 000 are shown with two views. In both cases, the adaptation process captures the same physical features in the body-fitted case and in the embedded case. In the last case, the embedded geometry is well captured by the CFD solver as it is shown in Figures 38 and 39, where the detected geometry is shown after 5 iterations at each of the studied complexities. This process is able to recover even the small features of the object like the winglets as it is shown with the zoom provided.
Finally, a convergence study in $L^2$ norm (using the result at complexity 128 000 as reference) has been performed to compare the behavior of the numerical scheme when using the embedded approach with respect to the body-fitted counterpart. The results are given in Figure 35 and show that the approach damages the order of convergence of the method. This shows one of the limitations of the embedded approach. This can be explained by several factors: the local modification of the dual mesh, the lack of accuracy of the input geometry that is defined with a $P^1$ mesh, ...

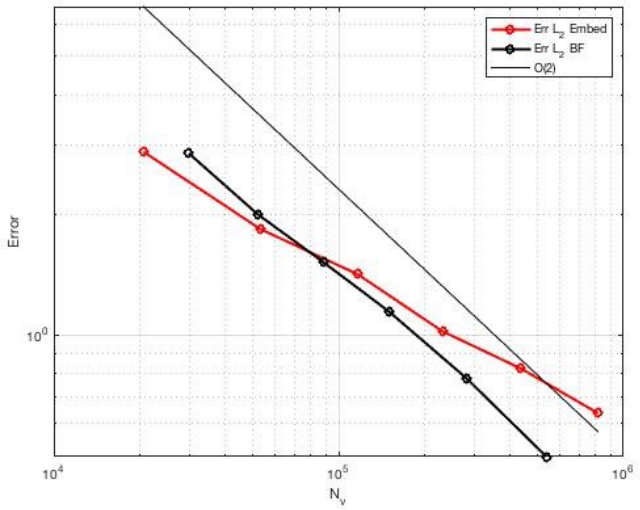



Figure 35: Convergence study in $L^2$ norm of the mesh adaptation process in the case of the supersonic notional missile. Embedded (red line) and body-fitted (black line) approaches are shown.

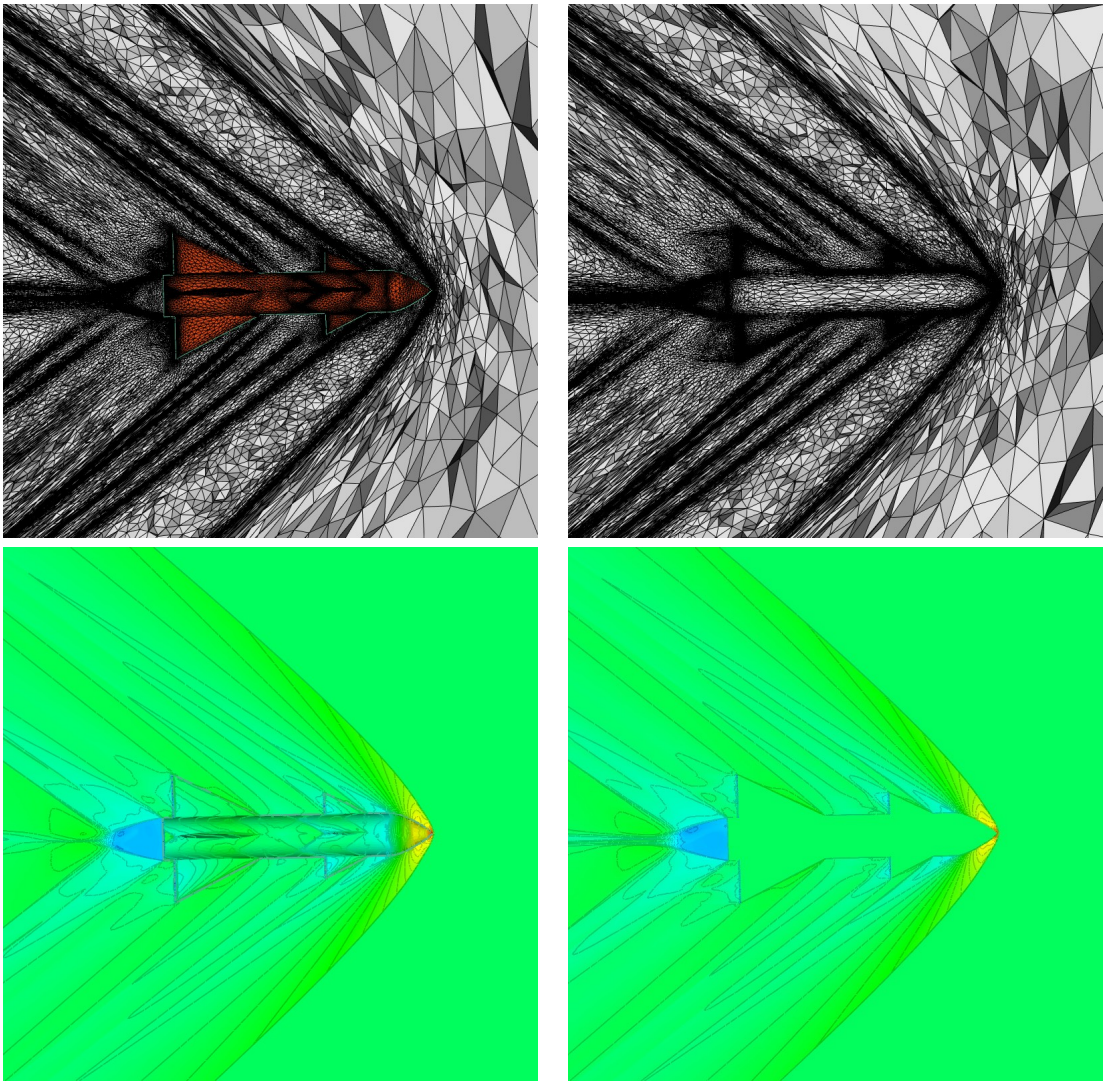


Figure 36: Supersonic flow around a notional missile. Left, adapted body-fitted case. Right, adapted embedded case. View along $y = 0$ axis. Top, the adapted meshes. Bottom, the associated solutions.

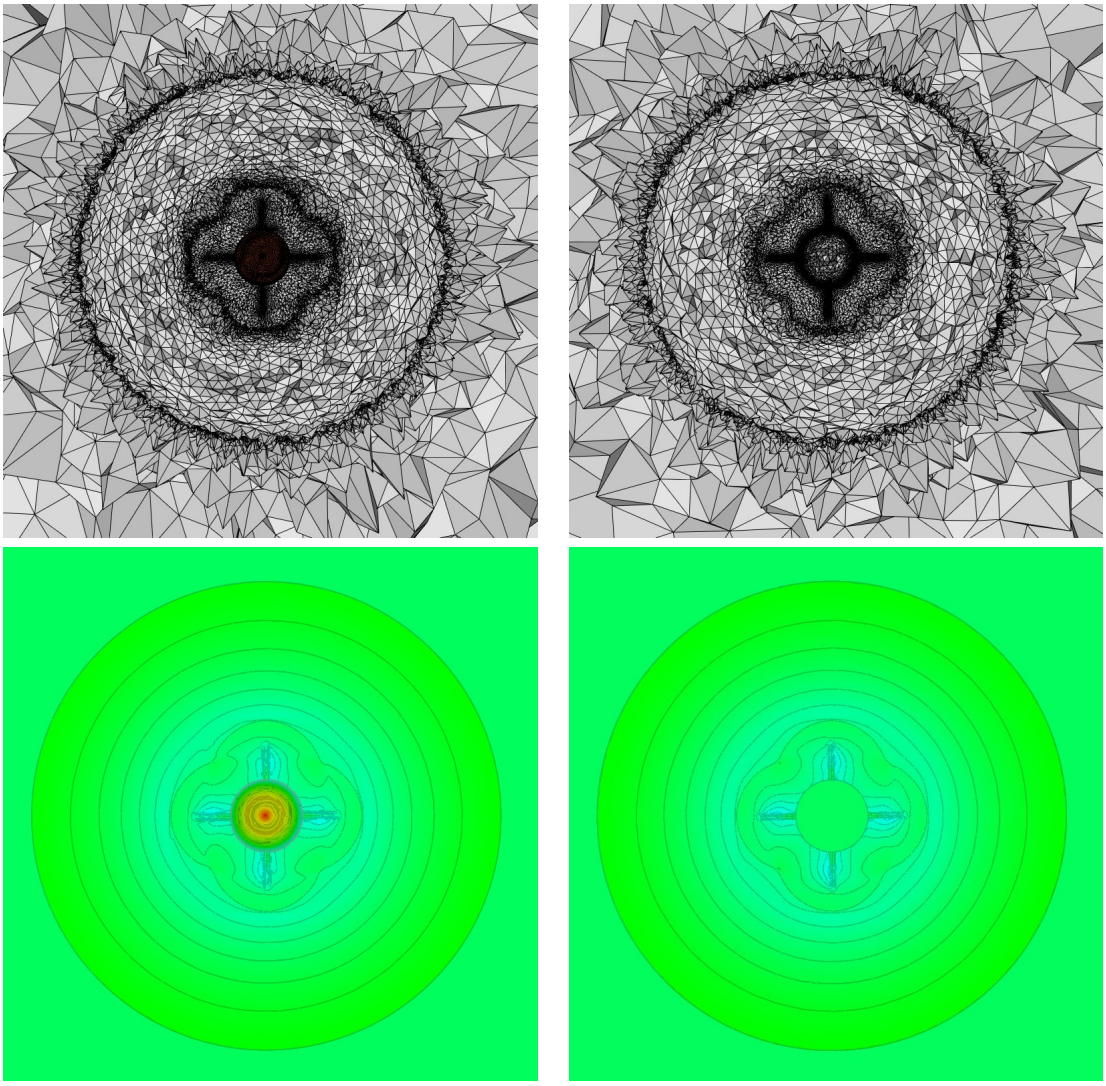


Figure 37: Supersonic flow around a notional missile. Left, adapted body-fitted case. Right, adapted embedded case. View along $x = 0$ axis. Top, the adapted meshes. Bottom, the associated solutions.

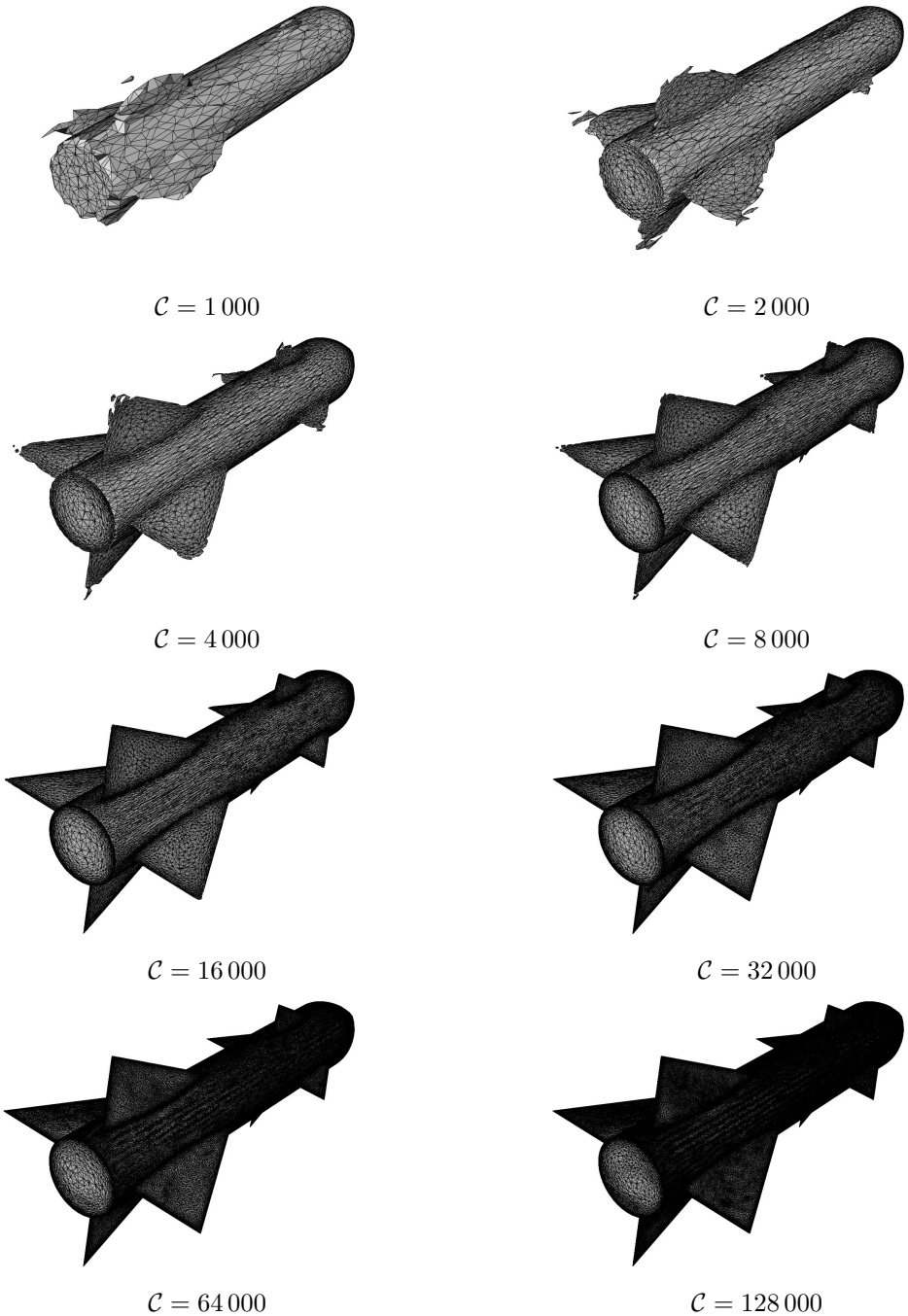



Figure 38: Evolution of the detection of the embedded geometry by the CFD solver all along the mesh adaptation process. Each of the 8 cases shows the result obtained at the last iteration of the adaptation process for a given complexity.

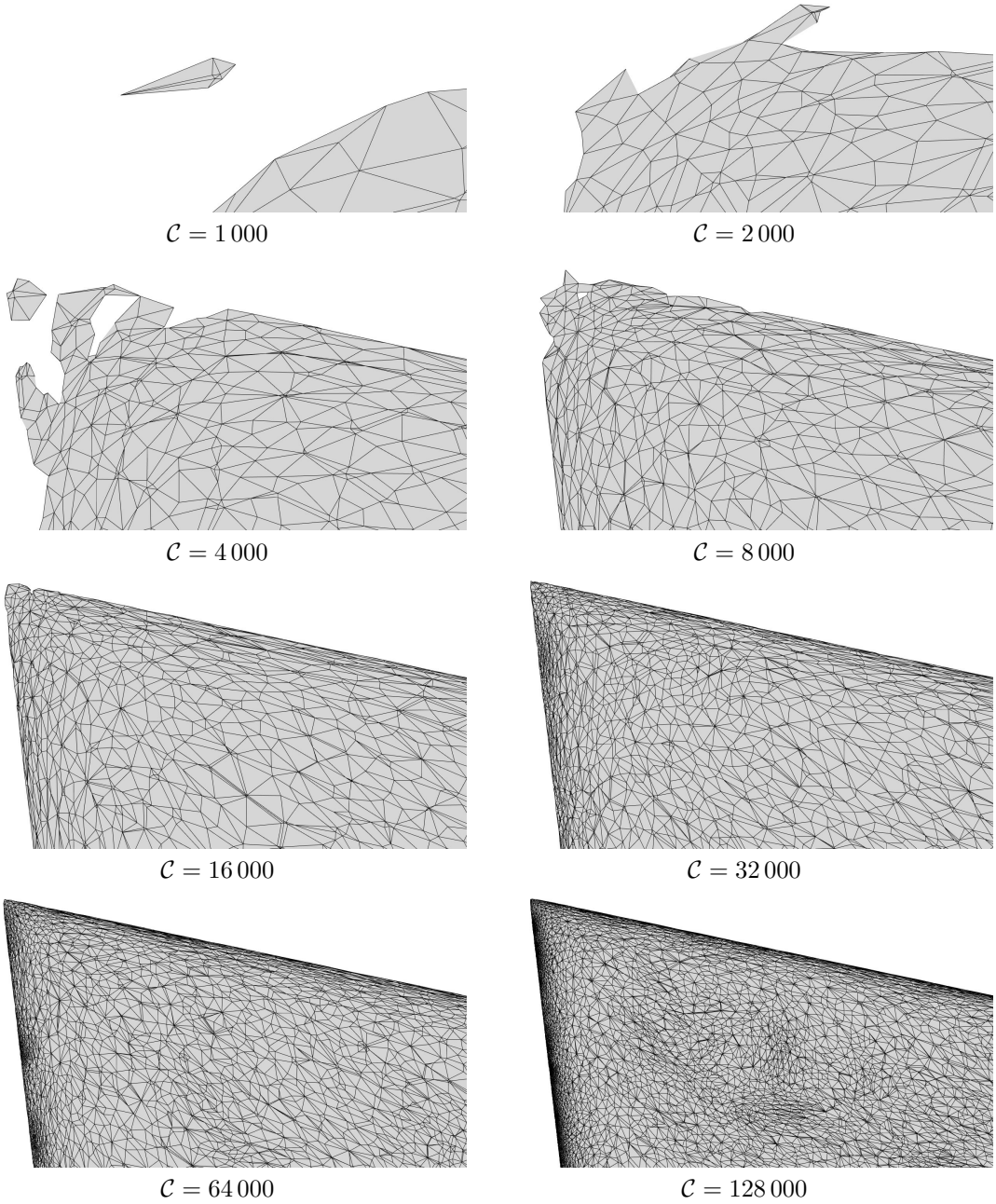



Figure 39: Zoom around the top winglet of the embedded geometries detected by the solver in Figure 38. Each of the 8 cases shows the result obtained at the last iteration of the adaptation process for a given complexity.

### 2.4.5 Conclusion

In this section, we have shown the implementation of the embedded boundary method in the context of a vertex-centered finite volume method applied to the Euler equations. Albeit an inevitable loss of precision, this method works fine in 2D and 3D and is able to predict a phenomenon around a fixed object without the meshing cost of a potentially complex object. The implementation of this method is fairly easy and consists primarily in a preprocessing step and in the implementation of a specific boundary condition. In particular, this does not affect the global behavior of the CFD solver in terms of CPU load. The method works as is on both steady and unsteady simulations as long as the embedded object does not move and does not require the embedded mesh to be valid in the finite element sense even if the geometry needs to be watertight in a way or another. The coupling of this method with method thus enables to retrieve the aerodynamical data of interest with a comparable accuracy than in the body-fitted case and without the requirement of a specific implementation during the mesh adaptation process.

# Conclusion

In this report, a review of the standard and mature mesh adaptation process has been performed. Then, a study of the embedded boundary method applied to the Euler equations has been made. The implementation of this method has been validated and has been shown to give the expected results but with a loss of precision in terms of aerodynamical data. To this end, the coupling with mesh adaptation has been performed and has evidenced that its use enables to get back the lost accuracy in the first place.
Some perspectives can be de drawn regarding this work. The first idea would be to create a level-set metric to improve the detection of the embedded geometry as it is performed in [47, 28]. This way, it would improve the mesh adaptation to converge quicker to the geometrical shape, in particular for the 3D case. Then, facing the issues encountered when dealing with coarse linear discretization of the embedded geometry, it would be interesting to directly consider the exact intersection of the mesh with the analytical model or with a high-order surrogate model. Finally, it would be a piece of interest to consider the generalization of this work to the Navier-Stokes equations, and in particular RANS equations.
Otherwise, it is important to point out that the interest of this work seems more limited than expected. Indeed, when the question of dealing with moving embedded geometries in 2D occurs, some issues appear:

- The implementation of the static case relies on local modifications of the Finite Volume Cell using a cut-cell approach so that the embedded boundary is well represented within the dual mesh. The underlying idea is that if a point is covered by the geometry then there is no sense in considering its nodal value and its associated Finite Volume cell has thus to be removed from the uncovered part of the mesh. However, by doing this with embedded moving geometries, it comes down to make locally move the cell interfaces from one iteration to another. To properly to take this into account, it requires the use of an ALE solver whose cost is way higher than a standard one.

- An attempted approach was thus to perform at the next iteration an exact cut-cell with the median cells of the back mesh (*e.g.* to consider the effective result of the intersection of the dual mesh with the back mesh), to perform the advance in time using a Runge-Kutta scheme and then to merge the cells so that we get back to the static case as in [54, 55].

- However by doing this, it forces the CFL number to stay at really low values as it is mandatory for an existing cell at $t^n$ and that vanishes at $t^{n+1}$ to have a non-null area at the exact cut-cell step at $t^{n+1}$. In other words, it seems impossible to have a CFL number greater than 0.5.
  In every cases, this drastically reduces the interest of considering moving objects as, based on this previous observation, it seems highly complicated to deal with both multi-step Runge-Kutta and implicit schemes. Finally, the use of unsteady mesh adaptation seems inappropriate as the smallest size of the mesh will rule the global time step using a CFL number that has to stay small.

Using these observations the development of the embedded approach on a vertex-centered Finite Volume scheme is not as advantageous as expected. One of the initial objectives of this study was indeed to use the embedded approach to simulate complex multi-bodies moving problems as it occurs in fragmentation problems.

## Acknowledgments

This work was supported by the public grants : *Investissement d'avenir* project, reference ANR-11-LABX-0056-LMH, LabEx LMH and ANR Impacts, reference ANR-18-CE46-0003.

## References

[1] R. Abgrall, H. Beaugendre, and C. Dobrzynski. An immersed boundary method using unstructured anisotropic mesh adaptation combined with level-sets and penalization techniques. *Journal of Computational Physics*, 257:83–101, 2014.

[2] F. Alauzet. *Adaptation de maillage anisotrope en trois dimensions. Application aux simulations instationnaires en Mécanique des Fluides*. Ph.D. Thesis, Université de Montpellier, October 2003.

[3] F. Alauzet. A parallel matrix-free conservative solution interpolation on unstructured tetrahedral meshes. *Computer Methods in Applied Mechanics and Engineering*, 299:116–142, 2016.

[4] F. Alauzet. `Wolf` documentation. a navier-stokes flow solver based on the mixed element-volume numerical scheme. Internal report, INRIA, 2019.

[5] F. Alauzet and M. Mehrenberger. P1-conservative solution interpolation on unstructured triangular meshes. *International Journal for Numerical Methods in Engineering*, 84(13), 2010.

[6] T. Amari, A. Canou, J.-J. Aly, F. Delyon, and F. Alauzet. Magnetic cage and rope as the key for solar eruptions. *Nature*, 554:211–215, 2018.

[7] P. Angot, G.-H. Bruneau, and P. Fabrie. A penalization method to take into account obstacles in incompressible flows. *Numerische Mathematik*, 81:497–520, 1999.

[8] I. Babuska, B. A. Szabo, and I. N. Katz. The p-version of the finite element method. *SIAM journal on numerical analysis*, 18(3):515–545, 1981.

[9] R. E. Bank and R. K. Smith. A posteriori error estimates based on hierarchical bases. *SIAM Journal on Numerical Analysis*, 30(4):921–935, 1993.

[10] N. Barral. *Time-accurate anisotropic mesh adaptation for three-dimensional moving mesh problems*. Ph.D. Thesis, Université Pierre et Marie Curie - Paris VI, November 2015.

[11] P. Batten, N. Clarke, C. Lambert, and D. M. Causon. On the Choice of Wavespeeds for the HLLC Riemann Solver. *SIAM Journal on Scientific Computing*, 18(6):1553–1570, November 1997.

[12] J.A. Benek, P.G. Buning, and J.L. Steger. A 3D chimera grid embedding technique. In *7th AIAA Computational Fluid Dynamics Conference*, AIAA Paper 1985-1523, Cincinnati, OH, USA, Jul 1985.

[13] M. J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of computational Physics*, 53(3):484–512, 1984.

[14] L. Billon. *Boundary-volume mesh generation and adaptation for turbulent flows around complex geometries.* Ph.D. Thesis, PSL Research University, December 2016.

[15] H. Borouchaki, T. Grosges, and D. Barchiesi. Improved 3d adaptive remeshing scheme applied in high electromagnetic field gradient computation. *Finite Elements in Analysis and Design*, 46(1):84 – 95, 2010.

[16] H. Borouchaki, P. Laug, A. Cherouat, and K. Saanouni. Adaptive remeshing in large plastic strain with damage. *International Journal for Numerical Methods in Engineering*, 63(1):1–36, 2005.

[17] S. Chaillat, S. Groth, and A. Loseille. Metric-based anisotropic mesh adaptation for 3D acoustic boundary element methods . *Journal of Computational Physics*, 372:473–499, 11 2018.

[18] P. Clément. Approximation by finite element functions using local regularization. *ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique*, 9(R2):77–84, 1975.

[19] O. Coulaud and A. Loseille. Very high order anisotropic metric-based mesh adaptation in 3d. *Procedia Engineering*, 163:353–364, 2016.

[20] C. Debiez and A. Dervieux. Mixed-element-volume muscl methods with weak viscosity for steady and unsteady flow calculations. *Computers & Fluids*, 29(1):89 – 118, 2000.

[21] C. Dobrzynski. *3D anisotropic mesh adaptation and application for air-conditionning.* Ph.D. Thesis, Université Pierre et Marie Curie - Paris VI, November 2005.

[22] K.J. Fidkowski and D.L. Darmofal. An adaptive simplex cut-cell method for discontinuous Galerkin discretizations of the Navier-Stokes equations. In *18th AIAA Computational Fluid Dynamics Conference*, AIAA-2007-3941, Miami, FL, USA, Jun 2007.

[23] K.J. Fidkowski and D.L. Darmofal. Triangular cut-cell adaptive method for high-order discretizations of the compressible Navier-Stokes equations. *Journal of Computational Physics*, 225:1653–1672, 2007.

[24] L. Frazza. *3D anisotropic mesh adaptation for Reynolds Averaged Navier-Stokes simulations.* Ph.D. Thesis, Sorbonne Université , UPMC, December 2018.

[25] P. J. Frey and F. Alauzet. Anisotropic mesh adaptation for cfd computations. *Computer methods in applied mechanics and engineering*, 194(48-49):5068–5082, 2005.

[26] E. Gauci. *Goal-oriented metric-based mesh adaptation for unsteady CFD simulations involving moving geometries.* Ph.D. Thesis, Université Côte d'Azur, December 2018.

[27] P. L. George, H. Borouchaki, F. Alauzet, P. Laug, A. Loseille, and L. Maréchal. *Meshing, Geometric Modeling and Numerical Simulation 2: Metrics, Meshes and Mesh Adaptation.* John Wiley & Sons, 2019.

[28] E. Hachem, S. Feghali, R. Codina, and T. Coupez. Immersed stress method for fluid-structure interaction using anisotropic mesh adaptation. *International Journal for Numerical Methods in Engineering*, 94(9):805–825, 2013.

[29] F.S. Hill. Ii.5. - the pleasures of "perp dot" products. In Paul S. Heckbert, editor, *Graphics Gems*, pages 138 – 148. Academic Press, 1994.

[30] W. Huang, L. Kamenski, and J. Lang. A new anisotropic mesh adaptation method based upon hierarchical a posteriori error estimates. *Journal of Computational Physics*, 229:2179–2198, 2010.

[31] B. Koren. A robust upwind discretization method for advection, diffusion and source terms. In *Numerical methods for advection-diffusion problems*, pages 117–138. Vieweg, 1993.

[32] R. Löhner. *Applied computational fluid dynamics techniques: an introduction based on finite element methods*. John Wiley & Sons, 2008.

[33] R. Löhner, S. Appanaboyina, and J.R. Cebral. Comparison of body-fitted, embedded and immersed solutions of low Reynolds-number 3-d incompressible flows. In *45th AIAA Aerospace Sciences Meeting*, AIAA-2007-1296, Reno, NV, USA, Jan 2007.

[34] R. Löhner and J. D. Baum. Adaptive h-refinement on 3d unstructured grids for transient problems. *International Journal for Numerical Methods in Fluids*, 14(12):1407–1419, 1992.

[35] R. Löhner, J.D. Baum, E. Mestreau, D. Sharov, C. Charman, and D. Pelessone. Adaptive embedded unstructured grid methods. *International Journal for Numerical Methods Engineering*, 60:641–660, 2004.

[36] R. Löhner, J.D. Baum, E.L. Mestreau, and D. Rice. Comparison of body-fitted, embedded and immersed 3-d Euler predictions for blast loads on columns. In *45th AIAA Aerospace Sciences Meeting*, AIAA-2007-1113, Reno, NV, USA, Jan 2007.

[37] A. Loseille. *Adaptation de maillage anisotrope 3D multi-échelles et ciblée à une fonctionnelle pour la mécanique des fluides : Application à la prédiction haute-fidélité du bang sonique.* Ph.D. Thesis, Université Pierre et Marie Curie - Paris VI, December 2008.

[38] A. Loseille and F. Alauzet. Continuous mesh framework part I: well-posed continuous interpolation error. *SIAM J. Numer. Anal.*, 49(1):38–60, 2011.

[39] A. Loseille and F. Alauzet. Continuous mesh framework part II: validations and applications. *SIAM J. Numer. Anal.*, 49(1):61–86, 2011.

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

[41] V. Menier. *Numerical methods and mesh adaptation for reliable RANS simulations*. Ph.D. Thesis, Université Pierre et Marie Curie - Paris VI, November 2015.

[42] G. Olivier. *Anisotropic metric-based mesh adaptation for unsteady CFD simulations involving moving geometries*. Ph.D. Thesis, Université Pierre et Marie Curie - Paris VI, April 2011.

[43] M.A. Park and D.L. Darmofal. Validation of an output-adaptive tetrahedral cut-cell method for sonic boom prediction. *AIAA Journal*, 48(9):1928–1945, 2010.

[44] C. S. Peskin. Flow patterns around heart valves: A numerical method. *Journal of Computational Physics*, 10(2):252 – 271, 1972.

[45] S. Piperno and S. Depeyre. Criteria for the design of limiters yielding efficient high resolution tvd schemes. *Computers & Fluids*, 27(2):183 – 197, 1998.

[46] M. A. Puscas, L. Monasse, A. Ern, C. Tenaud, and C. Mariotti. A conservative Embedded Boundary method for an inviscid compressible flow coupled with a fragmenting structure. *International Journal for Numerical Methods in Engineering*, 103(13):970–995, 2015.

[47] D. Quan, T. Toulorge, G. Bricteux, J.-F. Remacle, and E. Marchandise. Anisotropic adaptive nearly body-fitted meshes for cfd. *Engineering with Computers*, 30(4):517–533, Oct 2014.

[48] D.-L. Quan, T. Toulorge, E. Marchandise, J.-F. Remacle, and G. Bricteux. Anisotropic mesh adaptation with optimal convergence for finite elements using embedded geometries. *Computer Methods in Applied Mechanics and Engineering*, 268:65 – 81, 2014.

[49] M.-G. Vallet. *Génération de maillages éléments finis anisotropes et adaptatifs*. Ph.D. Thesis, Université Pierre et Marie Curie - Paris VI, 1992.

[50] B. Van Leer. Towards the ultimate conservative difference scheme. A second-order sequel to Godunov's method. *Journal of Computational Physics*, pages 101–136, 1979.

[51] M. Vanella, P. Rabenold, and E. Balaras. A direct-forcing embedded-boundary method with adaptive mesh refinement for fluid-structure interaction problems. *Journal of Computational Physics*, 229(18):6427 – 6449, 2010.

[52] J. Vanharen, R. Feuillet, and F. Alauzet. Mesh adaptation for fluid-structure interaction problems. In *24th AIAA Fluid Dynamics Conference*, AIAA Paper 2018-3244, Atlanta, GA, USA, Jun 2018.

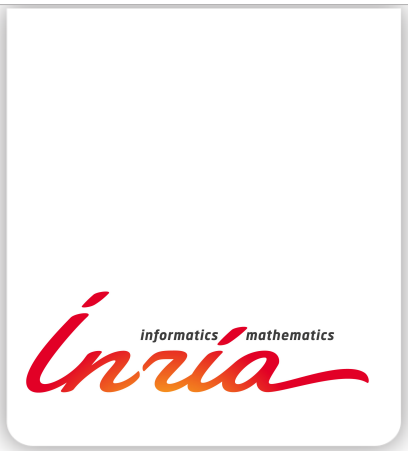
[53] K. Wang, A. Rallu, J.-F. Gerbeau, and C. Farhat. Algorithms for interface treatment and load computation in embedded boundary methods for fluid and fluid-structure interaction problems. *International Journal for Numerical Methods in Fluids*, 67:1175–1206, 2011.

[54] G Yang, DM Causon, DM Ingram, R Saunders, and P Batten. A Cartesian cut cell method for compressible flows part A: Static body problems. *Aeronautical Journal*, 101(1002):47–56, 1997.

[55] G Yang, DM Causon, DM Ingram, R Saunders, and P Batten. A cartesian cut cell method for compressible flows Part B: moving body problems. *Aeronautical Journal*, 101(1002):57–65, 1997.

[56] O. C. Zienkiewicz and J. Z. Zhu. The superconvergent patch recovery and a posteriori error estimates. part 1: The recovery technique. *International Journal for Numerical Methods in Engineering*, 33(7):1331–1364, 1992.

[57] O. C. Zienkiewicz and J. Z. Zhu. The superconvergent patch recovery and a posteriori error estimates. part 2: Error estimates and adaptivity. *International Journal for Numerical Methods in Engineering*, 33(7):1365–1382, 1992.




**RESEARCH CENTRE**
**SACLAY – ÎLE-DE-FRANCE**

1 rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing
Campus de l'École Polytechnique
91120 Palaiseau

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399