FEMS – A Mechanics-oriented Finite Element Modeling Software

Modesar Shakoor

FEMS – A Mechanics-oriented Finite Element Modeling Software

Modesar Shakoor*

IMT Nord Europe, Institut Mines-Télécom, Univ. Lille, Centre for Materials and Processes, F-59000 Lille, France

December 14, 2021

Abstract

This paper is a presentation of a Finite Element Modeling Software named FEMS that integrates mesh generation and adaption features in order to alleviate significantly the difficulty of designing a Finite Element (FE) mesh for a particular problem. FEMS is targeted at engineers and scientists addressing localization problems in mechanics, although it should be suited to many other applications.
FEMS is particularly relevant for problems with internal interfaces, both in solid and fluid mechanics, as it has both explicit and implicit interface representation. The former can be generated from signed distance functions using body-fitted meshing capabilities implemented in FEMS, while the latter relies on the level-set method. The choice between the one or the other can be made by the user depending on the severity of deformations in the neighborhood of an interface.
During the simulation, FEMS adapts the FE mesh automatically to achieve the best accuracy for a prescribed number of nodes. This is possible for both linear and quadratic interpolation. Additionally, in an updated Lagrangian setting, FEMS triggers mesh adaption automatically to avoid element flipping during node motion.
The capabilities of FEMS are demonstrated in this paper for fluid and solid mechanics problems featuring turbulence, multiphase flow, large deformations and plasticity. This wide range of problems that can be handled by FEMS should prove its great interest for the computational mechanics commmunity.

Keywords: finite elements, computational mechanics, mesh adaption, level-sets

1 Introduction

The Finite Element (FE) method is a numerical method that has been developed since the middle of the 20th century to address civil and mechanical engineering problems under the assumptions of continuum mechanics. Prolific scientific research has demonstrated that this method will play a key role in addressing current and future challenges in computational mechanics:

- A wide range of mechanisms have been modeled using the FE method (*e.g.*, laminar and turbulent flow [1], small and large deformation of structures [2, 3], multiphase flow with surface tension [4], nonlinear material behavior [5], fracture [6, 3]).
- The FE method can be used to model advanced materials of increasing complexity, with material laws that can even be computed on-the-fly using computational homogenization thanks to promising developments such as the FE^2 approach [5].
- With a proper mesh generation tool, it can be used to model domains with very complex geometry and morphology, even in the presence of internal interfaces [7].

*Corresponding author

- It can be applied at the macroscale as well as at the microscale or at any scale where continuum mechanics assumptions apply. Internal interfaces may be explicitly represented using a body-fitted FE mesh or implicitly represented using for instance Level-Set (LS) functions [6, 3].
- It is compatible with parallel computing on shared- and distributed-memory architectures, as well as on Graphical Processing Units (GPUs) [8]. As the number of cores in supercomputers keeps increasing, this is of major importance for a numerical method.

Although the FE method is quite simple to implement, the design of an FE code is often elaborated carefully so that the code can be modified and improved by mechanical engineers that do not necessarily have a scientific computing background. For instance, most commercial FE codes allow the user to add material laws quite easily through so-called user material subroutines without requiring a thorough understanding of the underlying FE code.

In this paper, a Finite Element Modeling Software named FEMS (pronounced *fems*, as a single word) that has been designed with a similar ambition in mind is presented. This ambition is to alleviate significantly the difficulty of designing an FE mesh for a particular problem, which is often the most time consuming part of an FE analysis [9]. FEMS is targeted at engineers and scientists addressing localization problems (*e.g.*, turbulence and boundary layers in fluid dynamics, shear bands and fracture in solid mechanics).
The technology used to alleviate the mesh design difficulty is adaptive mesh generation and adaption. The method implemented in FEMS is described in Sec. 2, while details on the software itself are given in Sec. 3. Illustrative examples are assessed in Sec. 4.

2 Method

In computational mechanics, numerical methods are required to provide a discretization of the geometry of the domain, which may include internal interfaces, and a discretization of various mechanical variables including displacements, velocities, stresses, and pressures. Additionally, equations featuring these variables and their partial derivatives must be solved.

2.1 Geometry approximation

For a domain $\Omega \subset \mathbb{R}^d$, $d = 1$, 2, or 3 being the space dimension, the discretization in the FE method as considered in this paper consists in an FE mesh, which is a set of line segments in 1D, triangles or quadrangles in 2D, and tetrahedra or hexahedra in 3D. These different elements carry a number of nodes, which at least include their vertices. Nodes may also be placed at the middles of an element's edges (P2 and Q2 interpolation), or even inside it (higher-order interpolation).
In the following, the set of all nodes of a given FE mesh is denoted $\mathcal{N}$. A unique global number $n \in \mathcal{N}$ identifies each node, while the coordinates of this node are given by $\mathbf{A}_n \in \Omega$. The set of all elements is denoted $\mathcal{T}$, and an element K is defined by its nodes set $\mathcal{N}(K)$. A local number $n_K \in \mathcal{N}(K)$ identifies each of an element's nodes. The connectivity operator Π_K is defined to associate each local identifier to a global one, for instance $n = \Pi_K(n_K)$ is the global identifier of the node with local identifier n_K in element K.

In a body-fitted FE mesh, the geometry's boundary as well as all internal interfaces are explicitly represented by the faces of some elements. Note that the word *faces* herein means vertices in 1D, edges in 2D, and actual faces in 3D. For some large deformation problems in mechanics, the deformation is computed in different steps, nodes coordinates $(\mathbf{A}_n)_{n\in\mathcal{N}}$ being updated at each step $t \to t + \Delta t$ so that to follow the mechanical deformation of the geometry, namely

$$\forall n \in \mathcal{N}, \mathbf{A}_n(t + \Delta t) = \mathbf{A}_n(t) + \Delta\mathbf{u}_n(t), \tag{1}$$

with $\Delta\mathbf{u}_n$ the incremental displacement vector at node n. This so-called updated Lagrangian setting, or in short Lagrangian mesh, is difficult to implement when deformations become too large and too complex as is

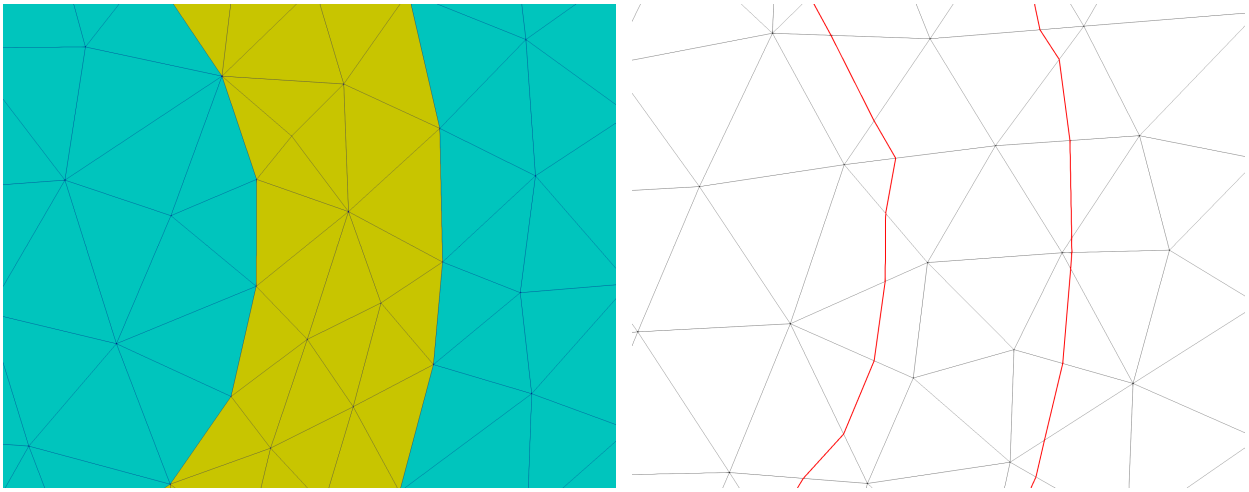
often the case in fluid mechanics.
An Eulerian mesh is usually preferred in fluid mechanics, which means that nodes coordinates remain constant throughout the simulation. If there are internal interfaces, they must be represented and convected by other means to follow the deformation of the geometry. This can be achieved by the LS method [10], where any internal interface is implicitly represented by the zero iso-level of a so-called LS function, which is a signed distance function. For instance, if at an instant t the domain $\Omega(t)$ is split into two parts $\Omega_1(t)$ and $\Omega_2(t)$ separated by an interface $\Gamma_{1,2}(t)$, LS function ϕ is defined by

$$\phi(\mathbf{x},t) = \begin{cases} +dist(\mathbf{x},\Gamma_{1,2}(t)), & \mathbf{x} \in \Omega_1(t), \\ -dist(\mathbf{x},\Gamma_{1,2}(t)), & \mathbf{x} \in \Omega_2(t), \\ 0, & \mathbf{x} \in \Gamma_{1,2}(t), \end{cases} \tag{2}$$

and is advected at each step by solving the advection equation

$$\frac{\partial \phi}{\partial t} + \mathbf{v}.\nabla\phi = 0, \tag{3}$$

with $\mathbf{v}$ the velocity vector field. This equation must be coupled to an LS reinitialization method in order to ensure that ϕ remains as close as possible to a signed distance function as per Eq. (2). This is achieved in FEMS using geometric reinitialization. Examples of explicit and implicit interface representations are shown in Fig. 1.



(a) (b)

Figure 1: Comparison of the same geometry with internal interfaces: (a) explicitly represented in a body-fitted FE mesh, (b) implicitly represented as the zero iso-level (in red) of an LS function.

2.2 FE solvers

The discretization of known and unknown variables is defined by their values at nodes of the FE mesh. A partial differential equation can be solved through numerical integration of its weak form and the solution of the resulting linear or nonlinear algebraic problem. The reader is referred to FE textbooks for more details on the FE method, its implementation, and its applications [11, 12, 13].
For instance, solid mechanics problems solved in Subsec. 4.3 using a Lagrangian mesh involve the incremental displacement vector field $\Delta\mathbf{u}$, and also the pressure field p in the incompressible case. Fluid mechanics problems solved in Subsec. 4.2 using an Eulerian mesh involve the velocity vector field $\mathbf{v}$, the pressure field

p, and also an LS function ϕ for two-phase flow cases.

FEMS integrates FE solvers for diffusion, reaction, convection equations and any combinations of those, as well as solvers for the Stokes equations, the Navier-Stokes equations for Newtonian incompressible flow, and static balance equations for linear elasticity, von Mises elasto-plasticity, and hyperelasticity. Those solvers are fully compatible with P1, P2 and higher-order elements, as well as Q1 elements (*i.e.*, quadrangles in 2D and hexahedra in 3D).
Time discretization in FEMS is always operated with the backward Euler method, which does not have a clear meaning for coupled problems and this is hence clarified in the sequel. Linear algebra solvers used to solve the associated linear problems are presented in Par. 3.2.2.

2.2.1 Incompressibility

A difficulty arises in both computational solid and fluid mechanics due to incompressibility. It is dealt with using a mixed formulation where instead of solving only for displacements (solids) or velocities (fluids), a coupled displacement-pressure or velocity-pressure problem has to be solved.
Mixed formulations are well-known to require stable discretization pairs in order to avoid pressure locking [14]. FEMS has implementations of the Taylor-Hood P2/P1 pair and the $P1^+$/P1 MINI element for small strain and finite strain solid mechanics problems with incompressible materials. The Taylor-Hood P2/P1 pair is also implemented for Navier-Stokes equations. For the latter, equal-order P1/P1 or P2/P2 pairs may be used in conjunction with Residual-Based Variational MultiScale (RBVMS) stabilization, which embeds stabilization for the incompressibility constraint.
All these mixed formulations are solved as coupled problems where the two variables are solved simultaneously. Except for the MINI element and the RBVMS stabilization, these mixed formulations lead to a saddle point problem which is solved using suitable preconditioning (*e.g.*, Schur complement).

2.2.2 Convection

Convection terms such as $\mathbf{v}.\nabla\phi$ in Eq. (3) are known to lead to instabilities if they are implemented with a standard FE approach. Both Stream Upwind Petrov Galerkin [15] (SUPG) and RBVMS [16] stabilization is implemented in FEMS. SUPG stabilization terms are explicit in time, while RBVMS stabilization terms are always implicit in time.
This is also true for the Navier-Stokes equations, which are solved using a Newton-Raphson scheme to deal with the nonlinearity of the $\mathbf{v}.\nabla\mathbf{v}$ term and of the implicit RBVMS stabilization terms, if those are enabled. For multiphase flow problems with surface tension, Eq. (3) is coupled to multiphase Navier-Stokes equations with a surface tension term. This coupled problem is solved with a fully implicit backward Euler scheme with RBVMS stabilization [17, 4].

As can be seen in this subsection, standard FEs are not enough to deal with computational mechanics problems. As a mechanics-oriented FE code, FEMS integrates advanced FE formulations for incompressible materials or flows and convection-dominated problems. This is necessary for instance to address elasto-plastic material behavior and turbulent flows. Mesh adaption is particularly relevant for such problems.

2.3 Mesh adaption

For each of the problems discussed in Subsec. 2.2, a sensor variable $\mathbf{s}$ can be defined as the vector of all relevant unknowns. For instance, for solid mechanics $\mathbf{s} = \Delta\mathbf{u}$ and for incompressible flows $\mathbf{s} = \mathbf{v}$.

For a given FE approximation $\mathbf{s}_h$ on a current mesh, two alternative objectives can be addressed using unstructured mesh adaption. The current mesh can be modified to obtain a mesh either satisfying a prescribed approximation error tolerance with a complexity as low as possible, or reducing the approximation error as much as possible for a prescribed complexity. Here the complexity is considered directly proportional

to the numbers of nodes and elements in the FE mesh.
Modifications may include removing, adding and moving nodes, as well as removing and adding elements. The criteria for determining which modifications to perform on the current mesh are defined through error estimators and then metric tensor fields, while the modifications are operated through a remeshing algorithm. Modifications are not easy to implement for all element types, and mesh adaption is thus restricted in this paper to simplexes *i.e.*, linear triangles in two dimensions (2D) and linear tetrahedra in three dimensions (3D).

2.3.1 Error estimators and metric tensor fields

Mesh adaption is a multi-objective optimization process targeting element qualities and edge lengths. In isotropic mesh adaption, a scalar mesh size field has to be defined on the FE mesh to determine the length prescribed locally for each edge. An optimal simplex has a volume as large as possible with the lengths of its edges as close as possible to this local mesh size. This contradiction between the two objectives requires to define a compromise. This will be addressed by the remeshing algorithm in the sequel.
For anisotropic mesh adaption, a metric tensor field $\mathbf{M}$ has to be defined on the FE mesh. This second order tensor defines locally d orthogonal directions and d independent scalar metrics in these directions [18]. The optimization remains identical to that of isotropic mesh adaption, but this distortion of the Euclidean metric is embedded in the definitions of element volume $\forall K \in \mathcal{T}$,

$$|K|_{\mathbf{M}} = \int_K \sqrt{\det\left(\mathbf{M}(\mathbf{x})\right)}\mathrm{d}\mathbf{x}, \tag{4}$$

and edge length $\forall m_K, n_K \in \mathcal{N}(K), m_K \neq n_K, m = \Pi_K(m_K), n = \Pi_K(n_K)$,

$$||\mathbf{A}_m\mathbf{A}_n||_{\mathbf{M}} = \int_{\mathbf{A}_m\mathbf{A}_n} \sqrt{\mathbf{M}(\mathbf{x}).\left(\mathbf{A}_m - \mathbf{A}_n\right).\left(\mathbf{A}_m - \mathbf{A}_n\right)}\mathrm{d}\mathbf{x}. \tag{5}$$

As can be seen in Eqs. (4) and (5), a valid metric tensor $\mathbf{M}(\mathbf{x}), \mathbf{x} \in \Omega$ is a symmetric positive definite matrix. It can hence be expressed in diagonal form, for instance for $d = 3$ as

$$\mathbf{M}(\mathbf{x}) = \mathbf{R}(\mathbf{x}) \begin{pmatrix} \frac{1}{h_1^2(\mathbf{x})} & 0 & 0 \\ 0 & \frac{1}{h_2^2(\mathbf{x})} & 0 \\ 0 & 0 & \frac{1}{h_3^2(\mathbf{x})} \end{pmatrix} \mathbf{R}(\mathbf{x})^T \tag{6}$$

where each column $i = 1 \ldots d$ of matrix $\mathbf{R}(\mathbf{x})$ is a direction vector along which mesh size $h_i^2(\mathbf{x})$ is prescribed. As shown in Fig. 2, the metric tensor gives direct control over element shape. Various error estimators to construct metric tensor fields are implemented in FEMS. Two of them, that are illustrated in Sec. 4, are presented in the sequel.

First, metric-driven mesh adaption can be used to naturally adapt the mesh to a geometry and obtain meshes refined close to internal interfaces, and in particular in regions with large maximum principal curvature. This is relevant for an internal interface $\Gamma_{1,2}$ defined through an LS function ϕ as per Eq. (2), as can be done in FEMS for simple geometric entities and even geometries segmented from 2D pictures and 3D tomography images. The sensor variable can then be defined as $\mathbf{s} = \phi$.
Although anisotropic curvature-based mesh adaption could be implemented quite easily in FEMS based on the literature [19, 20], an isotropic criterion [21] is preferred as this adaption process may be used as a first step for mesh generation as done in Subsec. 2.4. Matrix $\mathbf{R}(\mathbf{x}), \mathbf{x} \in \Omega$ is thus the identity matrix, while

$$\forall i = 1 \ldots d, h_i(\mathbf{x}) = \max\left(h_{min}, \min\left(h_{max}, \tilde{h}(\mathbf{x})\right)\right), \tag{7}$$

$$\tilde{h}(\mathbf{x}) = \frac{h_c}{\lambda_s(\mathbf{x})} + \left(h_{max} - \frac{h_c}{\lambda_s(\mathbf{x})}\right) \min\left(\frac{|\mathbf{s}(\mathbf{x})|}{h_{max}}, 1\right), \tag{8}$$

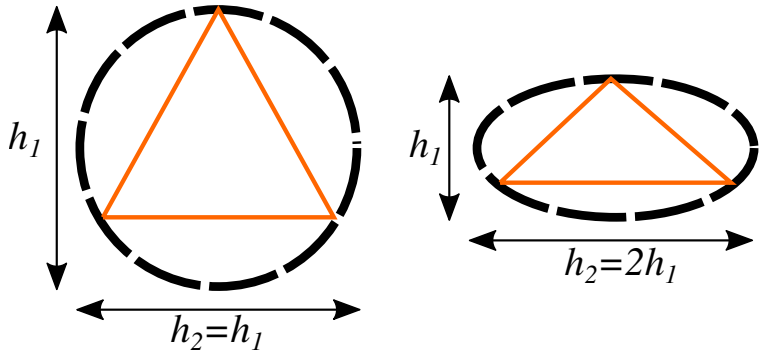


Figure 2: Influence of the metric field on the final shape of a triangle with the isotropic case on the left, and the anisotropic case on the right, assuming the first direction vector is vertical and the second one is horizontal.

where $\lambda_s(\mathbf{x})$ is the maximum eigenvalue of the Hessian matrix $\nabla\nabla\mathbf{s}(\mathbf{x})$ of $\mathbf{s} = \phi$ at point $\mathbf{x}$. As the eigenvalues of this Hessian matrix include the principal curvatures of $\Gamma_{1,2}$, it can be seen that at the interface, where $\phi(\mathbf{x}) \approx 0$, mesh size is prescribed to be inversely proportional to the curvature, h_c being a control parameter. It is hence prescribed to be very small for singularities of the interface ($\lambda_s(\mathbf{x}) \to \infty$), and very large for flat regions of the interface ($\lambda_s(\mathbf{x}) \to 0$). At a distance larger than h_{max} from $\Gamma_{1,2}$, mesh size is prescribed to be equal to h_{max}, with a linear transition from $\phi(\mathbf{x}) \approx 0$ to $\phi(\mathbf{x}) \approx h_{max}$. Overall, the prescribed mesh size is bounded between parameters h_{min} and h_{max}.
Note that the metric tensor field is to be defined at mesh nodes in order to be interpolated at quadrature points to evaluate Eqs. (4) and (5). In Eq. (7), sensor variable $\mathbf{s}$ can be replaced by its approximation $\mathbf{s}_h = \phi_h$ which is defined at mesh nodes, but not λ_s, which depends on the second derivatives of $\mathbf{s}$. The latter are not available at mesh nodes for Lagrange FEs, and are recovered in FEMS using an operation called Superconvergent Patch Recovery (SPR). SPR consists in recovering a higher-order and higher-regularity approximation of $\mathbf{s}$ around each mesh node [22]. In order to recover a regular Hessian matrix, this approximation is elevated to the third order in FEMS, as suggested in the literature [23]. This recovered approximation is fitted in a least-squares sense to the values of $\mathbf{s}$ at neighboring mesh nodes, this neighborhood being called the patch.

Second, for adaption to a sensor variable that may evolve to a very heterogeneous field during the simulation, it is preferable to use the complexity as control parameter. Indeed, this would prevent the computational cost from blowing up during the simulation, for instance due to complex topological events or very localized phenomena. The goal is hence to estimate the approximation error on $\mathbf{s}$, and prescribe a metric tensor field to distribute this error uniformly on the domain, for a given complexity. The simulation is expected to capture only a certain level of detail that can be afforded with this prescribed complexity.
The continuous mesh framework for metric-driven mesh adaption can be used to achieve this goal for simplex-type Lagrange FEs of any order [24, 25, 26, 4]. The definition of the metric tensor field is done in two steps. A geometric averaging operation is first used to define a single directional error tensor field $\mathbf{Q}$ for all components of $\mathbf{s}$. For linear (P1) FEs, this operation is defined $\forall \mathbf{x} \in \Omega$ as

$$\mathbf{Q}(\mathbf{x}) = \left(\exp\left(\frac{1}{\dim(\mathbf{s})} \sum_{i=1}^{\dim(\mathbf{s})} \log\left(\left(\nabla\nabla s_i(\mathbf{x}) \right)^{-\frac{1}{2}} \right) \right) \right)^{-2}. \tag{9}$$

Details on this geometric averaging operation can be found in the literature [18, 27]. The extension of this operation for quadratic (P2) Lagrange FEs is given by [4]

$$\mathbf{Q}(\mathbf{x}) = \left(\exp\left(\frac{1}{\dim(s)d} \sum_{i=1}^{\dim(s)} \sum_{j=1}^{d} \log\left(\left(\nabla\nabla \frac{\partial s_i}{\partial x_j}(\mathbf{x}) \right)^{-\frac{1}{2}} \right) \right) \right)^{-2}. \tag{10}$$

This first step includes a post-processing operation to control the element stretching and mesh size variations that will be induced by the error tensor field $\mathbf{Q}$. This is done by computing the median eigenvalue Q_{med} of $\mathbf{Q}$ over the whole mesh, and then bounding all its eigenvalues so that none of them are higher than $Q_{med}h_{max}$, or lower than ${}^{Q_{med}}/_{h_{max}}$, where h_{max} is the prescribed ratio.
The second step is to convert $\mathbf{Q}$ into a metric tensor field $\mathbf{M}$ minimizing the total error while uniformly distributing local errors and controlling the complexity. The solution of this constrained minimization problem can be expressed as [24, 25, 26]

$$\mathbf{M}(\mathbf{x}) = \mathcal{N}_c^{\frac{2}{d}} \left(\int_\Omega (\det(\mathbf{Q}(\mathbf{x}))^{\frac{k+1}{2(k+1)+d}} \mathrm{d}\mathbf{x} \right)^{-\frac{2}{d}} (\det(\mathbf{Q}(\mathbf{x})))^{-\frac{1}{2(k+1)+d}} \mathbf{Q}(\mathbf{x}), \tag{11}$$

where $\mathcal{N}_c$ is the prescribed number of P1 nodes (in the P2 case, this excludes nodes at edges middles), and k is the order of the FE method (1 for P1, and 2 for P2).
Note again that Eq. (9) requires to recover the second derivatives of $\mathbf{s}$ in the P1 case, and Eq. (10) in the P2 case its third derivatives. The SPR operation can be used in both cases [28, 4].

2.3.2 Remeshing algorithm

Once the metric tensor field $\mathbf{M}$ is defined and computed at mesh nodes using any of the error estimators implemented in FEMS, mesh modifications can be operated to satisfy the mesh size and orientations prescribed by this field. As mentioned previously, this requires to define a compromise between maximizing elements volumes as per Eq. (4) and bringing edge lengths as close as possible to 1 as per Eq. (5). Two strategies are available in FEMS to combine these objective.

In the first strategy [29], the edge length criterion is first applied by looping over all edges, splitting those that have a length larger than $\sqrt{2}$, and collapsing those that have a length smaller than $\sqrt{2}^{-1}$. This is done using a Delaunay kernel to ensure the FE mesh remains valid. Second, an element quality criterion is applied by looping over all elements and performing local mesh modifications such as edge flips and node re-positioning when they improve element quality. The latter is defined as

$$\mathcal{Q}_{\mathbf{M}}(K) = \alpha_d \frac{h_{\mathbf{M}}(K)}{|K|_{\mathbf{M}}}, \tag{12}$$

$$h_{\mathbf{M}}(K) = \left(\sum_{m_K, n_K \in \mathcal{N}(K), m_K < n_K} ||\mathbf{A}_{\Pi_K(m_K)} \mathbf{A}_{\Pi_K(n_K)}||^2_{\mathbf{M}} \right)^d, \tag{13}$$

where α_d is a normalization factor so that $\mathcal{Q}_{\mathbf{M}}(K) = 1$ for a regular simplex. Element quality $\mathcal{Q}_{\mathbf{M}}(K)$ must be minimized with this definition. Both edge sizing and element improving steps are performed again and again until no mesh improvement can be found by the algorithm.

In the second strategy [30, 31], a single element quality measure combining both the element quality criterion and the edge length one is defined for each element as

$$\mathcal{Q}_{\mathbf{M}}(K) = \min \left(\frac{d!}{\sqrt{d+1}} 2^{\frac{d}{2}} \frac{|K|_{\mathbf{M}}}{h_{\mathbf{M}}(K)}, h_{\mathbf{M}}(K), \frac{1}{h_{\mathbf{M}}(K)} \right), \tag{14}$$

$$h_{\mathbf{M}}(K) = \left(\frac{2}{d(d+1)} \sum_{m_K, n_K \in \mathcal{N}(K), m_K < n_K} ||\mathbf{A}_{\Pi_K(m_K)} \mathbf{A}_{\Pi_K(n_K)}||^2_{\mathbf{M}} \right)^{\frac{d}{2}}, \tag{15}$$

so that $\mathcal{Q}_{\mathbf{M}}(K) = 0$ for a degenerated simplex, and 1 for a regular simplex. Element quality $\mathcal{Q}_{\mathbf{M}}(K)$ must be maximized with this definition. This single element quality criterion is applied by looping over patches of elements neighboring all nodes and edges of the mesh, and applying local mesh modifications such as

edge flips, node re-positioning, node removal, and node addition when they improve element quality. This is performed again and again until no mesh improvement can be found by the algorithm. Examples of mesh modifications are shown in Fig. 3.

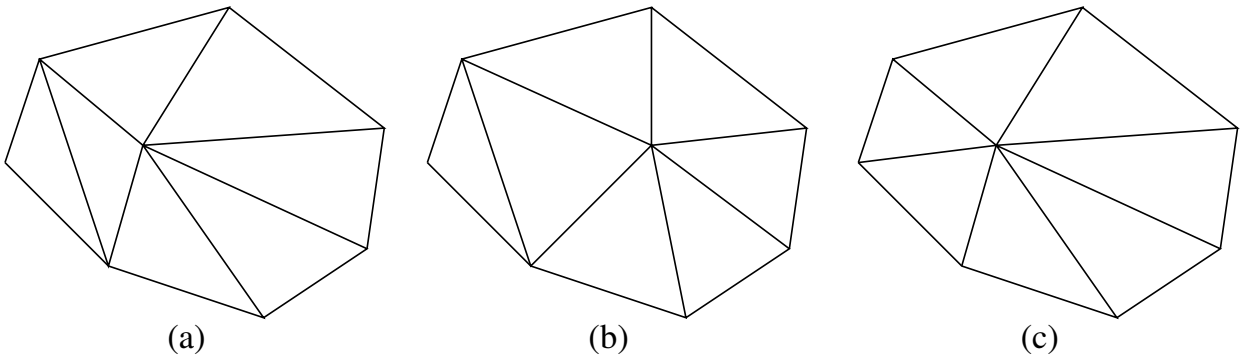


Figure 3: Example of mesh modifications in 2D: (a) initial patch of elements, (b) node re-positioning, (c) edge flip.

Due to their very general nature, the mesh modification operations used by both algorithms result in an unstructured simplex mesh, even if the initial mesh is structured. Additionally, both algorithms are restricted to linear simplex meshes both in 2D and 3D, and are fully compatible with either isotropic or anisotropic metric tensor fields.
Once the mesh adaption algorithm terminates, mechanical variables including the sensor variable must be transferred from the old mesh to the new (adapted) mesh. First, a space partitioning technique is used to locate efficiently the element of the old mesh containing each node of the new mesh. Variables values are then computed at each node of the new mesh using FE interpolation from the containing element of the old mesh.
For P2 FE meshes, since mesh adaption algorithms are restricted to linear simplexes, nodes at edge middles must be removed and then added back after the mesh has been optimized. This preserves the advantage of P2 FE interpolation during variables transfer from old to new mesh, both being P2 FE meshes, but forbids the use of isoparametric elements which could be interesting for body-fitted meshing of curved interfaces.
For variables defined at quadrature points, the space partitioning technique locates the element of the old mesh containing each quadrature point of the new mesh. Variables values are then directly copied from the closest quadrature point within that element of the old mesh to the quadrature point of the new mesh with no interpolation or smoothing.

As a conclusion, it can be seen that mesh adaption is not an easy task as it requires different mathematical theories to be understood and implemented. An error estimator is necessary to define a local mesh size criterion at each point of the domain, which in the case of FEMS can be an anisotropic metric tensor field. Then, this metric tensor field is used as input to a remeshing algorithm that will operate local modifications on the topology of the mesh and the position of its nodes to optimize a metric-based quality criterion. Finally, a variables transfer algorithm is necessary to transfer any variable defined on the old mesh to the new (adapted) mesh.
An interesting feature in FEMS is that the sensor variable can be defined as an LS function in order to adapt the mesh to an implicitly represented geometry. This representation can also automatically be made explicit, as presented in the sequel.

2.4 Mesh generation from signed distance functions

The idea of generating body-fitted FE meshes for geometries implicitly represented through signed distance functions (LS functions) has been proposed by various authors [32, 28]. This is relevant for simple geometric entities such as circles, cylinders, ellipsoids, planes, squares and combinations (unions, intersections, complements) of those for which Eq. (2) can be analytically computed. This is even more relevant for geometries segmented from 2D or 3D images which may be acquired using optical or electronic microscopy, or tomography [33, 34].
Following Ref. [34], the methodology implemented in FEMS requires an LS function as input. It may be defined analytically, for instance from a sphere's center coordinates and radius, or voxel-wise on a background image. Note that an LS function can be computed directly on a segmented 2D or 3D image in linear complexity with respect to the number of voxels [35], while such performance cannot be achieved on unstructured simplex meshes [36, 37, 33].
This input LS function is used as sensor variable for mesh adaption using the isotropic curvature-based metric tensor field defined in Eqs. (6) and (7). Depending on the initial mesh, which may be any mesh of the domain independently of the LS function, this mesh adaption process will generally have to be done in several iterations. Indeed, the LS function and hence the error estimator may not be well represented on the initial mesh, and the adaption process will improve this discretization up to convergence (usually in 5-7 iterations [34]).
At convergence, if the simulation requires a body-fitted FE mesh, a discretization of the LS function's zero iso-level can be reconstructed as a surface mesh using either a triangle and tetrahedron marching strategy [32] or a purely topological internal fitting strategy [28]. The former browses each triangle or tetrahedron of the mesh and splits it depending on how it is intersected with the zero iso-level. The latter browses each edge of the mesh and splits it if it has LS function values of opposing signs at its ends by inserting a new node at its intersection with the zero iso-level. Both strategies have a linear complexity with respect to the number of nodes, but the latter is simpler to implement as it only computes intersections at edges.
Once a body-fitted FE mesh has been constructed, the LS function is no longer required, except for a last mesh adaption step. Indeed, mesh quality as defined in Eqs. (14) or (12) is likely to be deteriorated during the reconstruction of the zero iso-level's discretization. It must be restored using mesh adaption again, which must rely on a remeshing algorithm preserving the body-fitted mesh at internal interfaces, which is the case for both algorithms presented in Par. 2.3.2 (see Refs. [32, 31]).

Using the methods presented in this section, any geometry with internal interfaces can be represented using LS functions. Through those LS functions, body-fitted FE meshes can be automatically generated for some interfaces that may be modeled using a Lagrangian mesh. For remaining interfaces, the LS method and an Eulerian mesh can be used. The flexibility of FEMS enables to combine both approaches and solve complex mechanics problems using advanced FE formulations such as mixed formulations or the RBVMS formulation.

3 Software description

FEMS integrates unique features compared to existing open source adaptive FE codes. This is explained in the sequel, as well as the parallel implementation of FE solvers and the technologies used for inputs and outputs.
FEMS is mostly implemented in *C*, as defined by the 1999 *ISO C* standard [38]. Some *C++*, as defined by the 2011 *ISO C++* standard [39], is used for code blocks that interact with *C++* libraries. Some remeshing and mesh adaption operations are also implemented in *C++*.
Apart from those exceptions, object-oriented *C++* programming has been voluntarily avoided in FEMS as it has a well-known computational overhead due to encapsulation and repetitive creation/destruction of objects. This is compensated in *C++* through templates and other techniques optimizing the code during its compilation.

As a consequence, writing a good scientific computing code in *C++* requires a high level of expertise in scientific computing, which is generally not the case of people working in the field of computational mechanics. The latter often have some scientific computing notions and a high level of expertise in some area of mechanics (*e.g.*, metallurgy, turbulence, ductile fracture). This has led developers of the *FreeFem* solver [40] and the *FEnICS* computing platform [41] to hide the underlying *C++* implementations from their users by creating simplified abstract languages.
In order to open all the FEMS code to computational mechanics researchers, *C++* has hence be avoided as much as possible. Only a fundamental knowledge of *C* is required to understand the FEMS code and extend it with new functionalities.

3.1 Mesh adaption

Among mesh adaption libraries, the *libMesh* library [42] relies on element sub-division, which consists in locally sub-dividing some elements (triangles, quadrangles, tetrahedra, hexahedra, etc.) of an initially uniform mesh to refine the mesh in some regions of the simulation domain depending on some error measure. The library embeds an FE solver and error estimators to compute FE solutions and define this error measure. Note that element sub-division cannot be used to stretch and orient elements, for instance in boundary layers.
The *MAdLib* library [43] relies on unstructured mesh adaption restricted to tetrahedra. It can handle large deformations with internal interfaces thanks to a mesh motion algorithm that automatically triggers mesh adaption while moving FE nodes to follow a given mechanical displacement in order to avoid element flipping. This library is used by *FEnICS* [41], which embeds an FE solver and an error estimator to drive the mesh adaption process [2]. Although *MAdLib* can perform anisotropic mesh adaption, there is no error estimator to do so in *FEnICS*.
The *Mmg* platform [29, 32] is another solution for unstructured mesh adaption restricted to simplexes. This platform implemented in C also includes a mesh motion algorithm and is used by *FreeFem* [40] which embeds an anisotropic error estimator so that elements may be refined in some regions and also oriented to capture all features of the FE solution.
The first remeshing algorithm described in Par. 2.3.2 is provided by the *Mmg* platform. The latter also integrates the triangle and tetrahedron marching method described in Subsec. 2.4 in order to both adapt and generate a body-fitted FE mesh for a geometry with internal boundaries, which has to be defined only through LS functions and can hence be imported from pictures or tomography scans.
The second remeshing algorithm described in Par. 2.3.2 and the purely topological mesh generation strategy described in Subsec. 2.4 are implemented as an independent *C++* module within FEMS.
These different remeshing algorithms are integrated in FEMS in order to not be restricted by the limitations of the *Mmg* platform. In particular, there is ongoing research work on a GPU-accelerated remeshing algorithm, and on mesh generation strategies that are compatible with more than one LS function.

FEMS can be seen as a state-of-the-art FE code embedding remeshing and mesh generation algorithms in a global solution. LS functions definitions, error estimators and metric tensor fields required by those algorithms are all implemented in FEMS for P1 and P2 interpolation. As described in Par. 2.3.2, FEMS integrates a wrapper function enabling unstructured mesh adaption for P2 interpolation, whose improved convergence rates and conservation properties can hence be accessed.
FEMS is flexible enough to combine an adaptive Eulerian method for some regions of the domain which may undergo large distortions and complex topological changes during the simulation with an adaptive Lagrangian method for remaining regions which may not undergo such drastic deformations.
To the extent of the author's knowledge, there is no open source code that integrates all these functionalities. There is ongoing work to improve FEMS with parallel implementations of the remeshing algorithm. Nevertheless, FEMS already relies on shared-memory parallelism for most computationally expensive tasks.

3.2 Parallel computing

The goal of keeping things simple prevented the use of distributed-memory parallelism, which, like *C++*, requires a high level of expertise in scientific computing. Indeed, new functionalities implemented in a distributed-memory parallel code must also be parallel, or at least implement some communications. This is hidden from the user using abstraction in most distributed-memory FE codes [44, 40, 41].
Shared-memory parallelism has been used in FEMS to avoid this complexity. It is implemented using *OpenMP* directives [45]. The main advantage is that a new functionality can be implemented with no parallelism and still be fully compatible with the rest of the code. In most cases, the new code can be optimized using simple parallel computing directives that can be added later once the sequential code has been tested and validated.
The drawback is that FEMS can only run on a single node and will hence be more efficient on high performance computing facilities with a high number of CPU cores and RAM space per computing node. FEMS also performs well on workstations and laptops, for which distributed-memory parallelism is not necessary. Additionally, some critical operations are GPU-accelerated.

3.2.1 Inherently parallel unitary and patch operations

In parallel computing, operations may be difficult to implement depending on what information is required as input and computed at output. In an FE code, information means variables of any dimensions which can be global scalars, vectors or tensors, and can also be scalar, vector or tensor fields stored entity-wise. An entity can be a node, an element, an element's quadrature point, a face or a face's quadrature point.
There are obvious associations, called adjacencies in FEMS, between these entities. An element has faces, nodes and quadrature points. A face has nodes and quadrature points. Additionally, an internal face appears twice in the FE mesh data structure, the two instances having two different orientations and being defined as adjacent to each other. All adjacencies go both ways, meaning for instance that as an element has nodes, a node is adjacent to all elements it belongs to. An entity is also always at least adjacent to itself.
These adjacencies are relevant for core FE operations such as quadrature, which requires interpolating a FE field from an element's nodes to its quadrature points. Quadrature plays a key role in the FE assembly and solution of partial differential equations [22, 11, 12].

A unitary operation in FEMS is an operation where all input and output variables have the same type, and which can be conducted at an entity indifferently to its type and independently from inputs and outputs at other entities. For instance, the operation in Eq. (1) is a node-wise addition of two variables defined at nodes. The addition on a given node does not depend on values from other nodes. This is also true for the metric computation operations in Eqs. (7), (9) and (10), but not for the operation in Eq. (11), which includes an integral over the whole simulation domain.
A unitary operation is called in FEMS through a function pointer and variables defined on the same mesh and for the same entity type. A parallel loop with an *OpenMP* directive will call the given function at each entity with all input and output variables as parameters. The user can thus easily implement new unitary operations and have them executed in parallel with no need to write *OpenMP* directives.

A patch operation in FEMS is an operation where all output variables have the same master entity type. Entity type node's master entity type is node, element's and element quadrature point's is element, face's and face quadrature point's is face. Input variables for a patch operation can be global variables or fields associated to any entity type.
A patch operation is called in FEMS through a function pointer and variables. Input variables can be of any type, and may even be defined on different FE meshes. However, the number of master entities for these meshes must be equal. This is true also for output variables, which must additionally have the same master type. A parallel loop with an *OpenMP* directive will call the given function at each master entity with all input and output variables as parameters, as well as the FE mesh data structure. The user can thus easily implement new patch operations and have them executed in parallel with no need to write *OpenMP*

directives.
As opposed to unitary operations, the user can program code inside a patch operation which accesses information from other entities to that for which the output is being computed. For instance, the patch operation for quadrature interpolates a node-wise field of any dimension to all quadrature points inside an element (here element is the master entity type). Among other things, this requires access to information on the FE, its nodes and basis functions.

This distinction between unitary and patch operations is inspired from distributed-memory FE codes [44], where the difficulty resides in hiding communications between processes from the user implementing the patch operation. These unitary and patch operations are very relevant for computational mechanics researchers, who may implement material laws, modify FE solvers by adding new terms, and implement all sorts of local entity-wise operations with no need to think about parallelism.
Note that the fact that an operation is easy to implement as a unitary or patch operation does not necessarily mean that it should. For instance, the node-wise summation in Eq. (1) has a too low computational cost in terms of floating point operations. It would probably be more time consuming to create and manage threads to do it in parallel instead of just implementing it in sequential.

3.2.2 Other operations

Some operations cannot be implemented efficiently as unitary or patch operations. Those are systematically implemented in sequential in FEMS, unless their computational cost is not negligible.

The node-wise metric tensor field $\mathbf{M}$ defined in Eq. (11) depends on a node-wise input tensor field $\mathbf{Q}$ and an integral over the whole simulation domain. The input tensor field can be computed in parallel using a unitary operation as per Eqs. (9) or (10), while the integral can be computed using a quadrature and integration patch operation and then sequential summation.

Geometry reinitialization of LS functions requires to reconstruct a surface mesh of the interface represented by the zero iso-level of the FE mesh. Then, each node of the FE mesh must be projected to the closest face of this surface mesh. This brute-force search of quadratic computational complexity can be done efficiently using a space partitioning technique [37], which is the same as that used to transfer FE variables after mesh adaption.
The space partitioning technique implemented in FEMS relies on a tree data structure that is optimized for finding the closest entity to a point, as well as the entity containing a point. Tree construction is implemented in sequential, but searches are done in a parallel loop using an *OpenMP* directive.
Alternatively, the brute-force search approach can be run on the GPU using an implementation in *NVIDIA's CUDA* programming language, which may be faster than the CPU version using space partitioning [4]. This is due to the fact that GPUs give access to a very large number of cores on a single node, as compared to CPUs.
CUDA code is integrated into FEMS thanks to the *CMake* tools suite, which enables users who do not have access to GPUs, or may not have a *CUDA* installation, to still compile and run FEMS on CPUs.

CMake is also used to flexibly manage the different third-party libraries that FEMS can be coupled to. These include the general purpose library *GLib* that is used for its data structures, and a quite significant number of scientific computing libraries. Entity-wise linear algebra operations such as matrix inversion or diagonalization are performed using either *Netlib*'s *CBLAS* and *LAPACKE*, or *Eigen3*. Entity-wise nonlinear systems such as those arising during numerical integration of nonlinear material laws are solved using *GSL* [46].
Additionally, the solution of partial differential equations with the FE method leads to large sparse nonlinear and/or linear systems. Nonlinear systems are solved within FEMS using a Newton-Raphson procedure, while linear systems may be solved using a FEMS implementation of the generalized minimal residual method, or third-party linear algebra libraries.

FEMS already has wrappers for sequential linear algebra libraries *PETSc* [47] and *UMFPACK* [48]. Shared-memory parallelism is accessible using *Lis* [49] or *SuperLU* [50]. GPU-accelerated solvers are also accessible using either *AmgX* [51], *PARALUTION* or *ViennaCL* [52]. Thanks to *CMake*, FEMS can still be compiled and used even if none of these third-party libraries can be found.

With *CMake* as well as simple *C* programming with *OpenMP* directives, it is possible for the user to change inputs and setup new problems to be solved with an adaptive FE method quite easily. However, it is still relevant to have the possibility to change some inputs without having to modify and recompile the code.

3.3 Inputs and outputs

All FEMS functionalities are used in *C* test files which are validated using the *CTest* tool of the *CMake* suite. A FEMS test always takes at least an FE mesh as input, for which the *MSH* format is used. Files with this format can be created with the *Gmsh* meshing software [7]. These meshes do not necessarily have to be body-fitted if there are internal interfaces, as this can be done within FEMS using the procedure explained in Subsec. 2.4.
The path to the mesh file must be indicated in an Extensible Markup Language (XML) input file which also contains all input parameters for a simulation. The format for XML input files is specified in the FEMS XML Schema Definition (XSD). The FEMS package includes *C* test files for *CTest* as well as XML input files for these tests that satisfy the FEMS XSD.
New input parameters can easily be added in the *C* code and then to the FEMS XSD. XML input files are systematically validated against the FEMS XSD and loaded using the *libxml2* library. It is hence possible for a developer to add new functionalities to FEMS and have users that have no programming experience to run simulations with varying geometrical, physical and numerical inputs.

The XML input file must also specify what inputs are desired and in which format. The only output format implemented in the current FEMS version is the VTU format, which is the unstructured grid VTK format that can be read using the ParaView visualization software [53]. The choice of FE variables to be written in the output VTU file depends on the problem being solved and is directly defined in the *C* test files.

As a summary, FEMS is accessible to different categories of users and developers.
The most popular use of FEMS is simply that of a student, engineer or researcher who wishes to run existing FEMS binaries with different geometrical, physical or numerical parameters. This includes usage of FEMS to generate body-fitted FE meshes with the method presented in Subsec. 2.4. This is possible simply by modifying XML input files.
The second and more technical use of FEMS is that of a user and developer who wishes to add new functionalities and test them with varying parameters. This requires *C* programming of unitary or patch operations to access parallel computing capabilities with no difficulty, or standard sequential *C* programming. Additionally, new parameters must be added to the FEMS XSD.
Finally, the third and last category of users will be experts in scientific computing who wish to add new functionalities that have a significant computational cost and cannot be implemented efficiently in parallel using the unitary/patch operations paradigm. Those functionalities will require some advanced *OpenMP* design and might also be GPU-accelerated using *CUDA* programming.

4 Results

The powerful capabilities of FEMS's isotropic and anisotropic unstructured mesh adaption are demonstrated in this section for various problems of computational mechanics. Input files for all simulation results shown in this section are provided in the examples directory of the FEMS package. Unless otherwise mentioned, initial meshes are structured FE meshes of the unit 1×1 mm^2 square (2D) or the $1 \times 1 \times 0.1$ mm^3 box

(3D), and internal interfaces are introduced using LS functions. Those initial structured meshes are modified automatically before and/or during the simulation.
All simulations were run on a workstation with a 28-core *Intel Xeon W-2175* 2.50GHz CPU and a 1024-core *NVIDIA Quadro P2000* GPU. Unless otherwise mentioned, sparse linear problems resulting from FE discretizations are solved using the direct solver *UMFPACK*, which operates an LU decomposition, and local entity-wise algebra operations are done using *Netlib*'s CBLAS/LAPACKE. For nonlinear problems, the tolerance for the residual of nonlinear solves using the Newton-Raphson scheme is always 10^{-6}.

4.1 Mesh generation from signed distance functions

This first set of simulations aims at showing the capabilities of the FEMS regarding the generation of body-fitted FE meshes for geometries with internal interfaces initially represented through LS functions, as presented in Subsec. 2.4. The geometry for these simulations is based on the 2D FEMS image shown in Fig. 4(a). The LS function to the surface of the letters is computed using the *Fiji* software [54] and is shown in Fig. 4(b,c).
As described in Subsec. 4.1, the metric tensor field used for mesh generation is that of Eqs. (6) and (7), with three metric parameters h_c, h_{min} and h_{max} to prescribe. In the sequel, body-fitted FE meshes are generated both in 2D and 3D by projecting the image-based LS function shown in Fig. 4(b,c) to the initial FE mesh of the 2D or 3D domain. Parameters h_{min} and h_{max} are set respectively to 4 µm and 32 µm both in 2D and 3D, while control parameter h_c is varied in order to show its influence.
The remeshing algorithm of the first strategy described in Par. 2.3.2 is used for these mesh generation simulations. To generate the body-fitted FE mesh, the strategy based on triangle and tetrahedron marching [32] is used.

4.1.1 2D

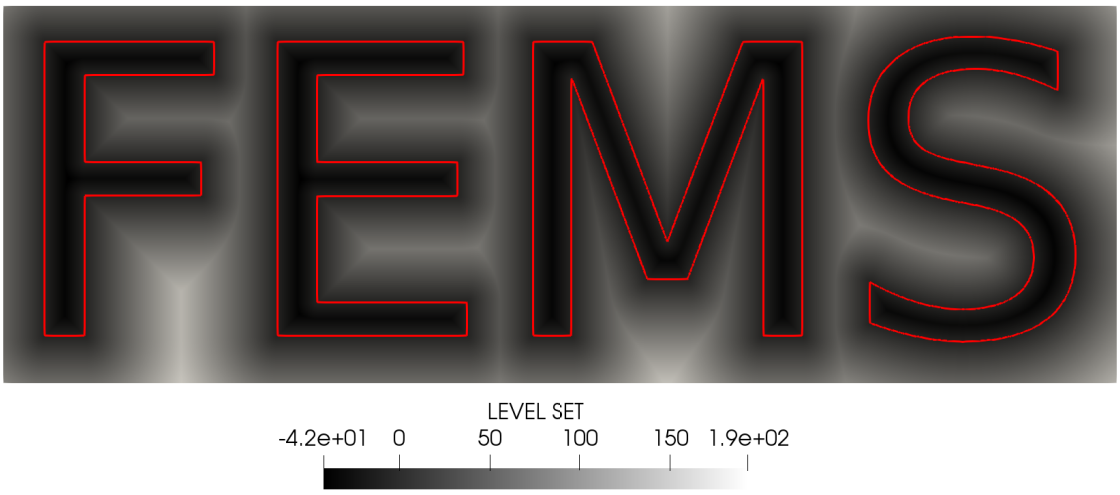
As described in Subsec. 4.1, the image-based LS function shown in Fig. 4(b,c) is first interpolated to a structured FE mesh of the domain. The result is shown in Fig. 5(a,b). Once the LS function is available at nodes of this FE mesh, the metric tensor field can be computed and mesh adaption can be performed. The result is shown in Fig. 5(c,d) for $h_c = 0.128$.
The LS function is then re-interpolated as it should be better captured and represented using the new adapted mesh, and the metric tensor is re-computed in order to re-adapt the mesh. The result after 8 cycles is shown in Fig. 6(a,b). These figures show how the mesh is automatically refined in regions with large local maximum principal curvature, which are mainly the regions of sharp angles in the letters.
Finally, this adapted mesh is modified through triangle marching and re-adapted in order to produce the mesh shown in Fig. 6(c,d). This final body-fitted mesh accurately captures all features of the geometry, especially the M letter which has three regions with very sharp angles.

The accuracy is obviously guided by the choice of parameters h_{min} and h_{max}, which determine bounds on the prescribed mesh size. However, metric parameter h_c has a major influence on how the local maximum principal curvature influences mesh size. This is shown in Fig. 7(a,b) where $h_c = 0.256$ has been used, and Fig. 7(c,d) where $h_c = 0.512$ has been used.
On the one hand, if h_c is too large, the local principal curvatures do not have any influence and a uniform mesh size of h_{max} is prescribed everywhere. On the other hand, if h_c is too low, h_{min} is prescribed everywhere. It is thus necessary to choose an intermediary value so that fine features with large local maximum principal curvature are well described but a coarser mesh size is prescribed in regions with low local maximum principal curvature.

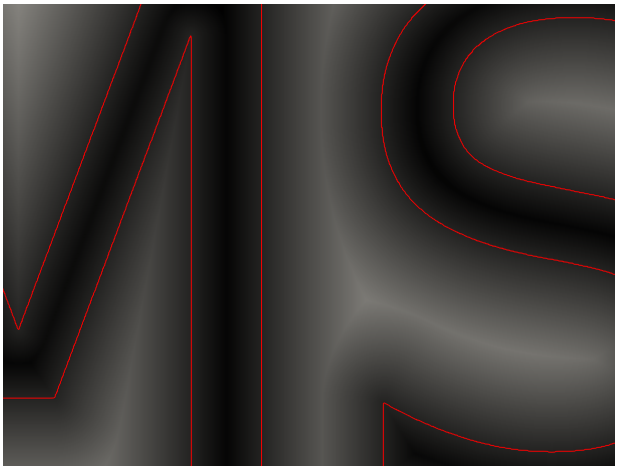
4.1.2 3D

The initial structured FE mesh has $21 \times 21 \times 3$ nodes, and is then adapted iteratively until the body-fitted FE mesh is generated, as done in the 2D case.

(a)



(b)



(c)

Figure 4: FEMS image used for mesh generation simulations: (a) image of 1999×679 pixels, (b) LS function computed on the image with signed distances in pixels, (c) zoom on the LS function.

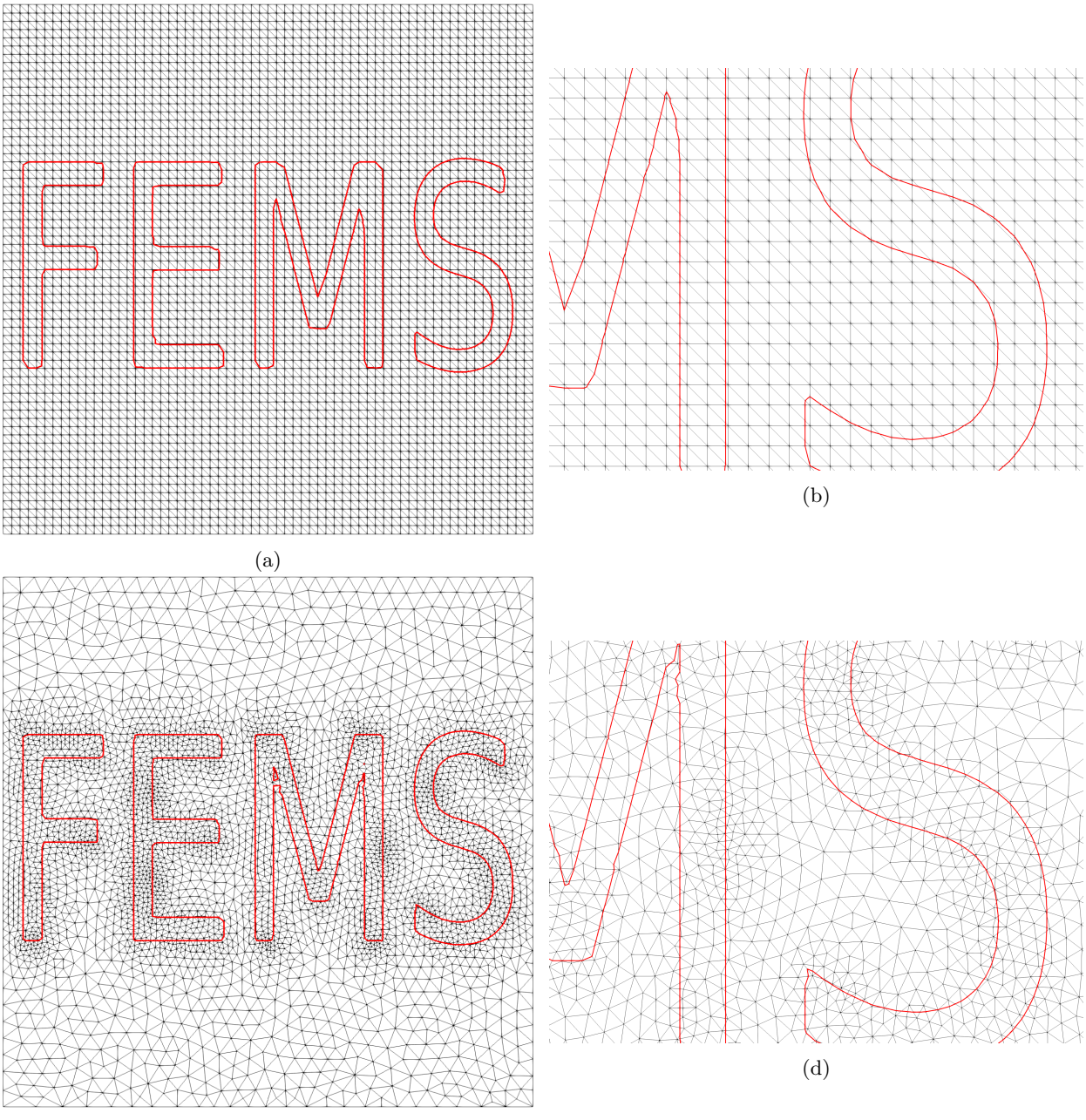


Figure 5: First steps of 2D mesh generation for the FEMS image using $h_c = 0.128$: (a,b) interpolation of the LS function from the image to the structured mesh, (c,d) mesh adapted once using the isotropic curvature-based metric tensor field.

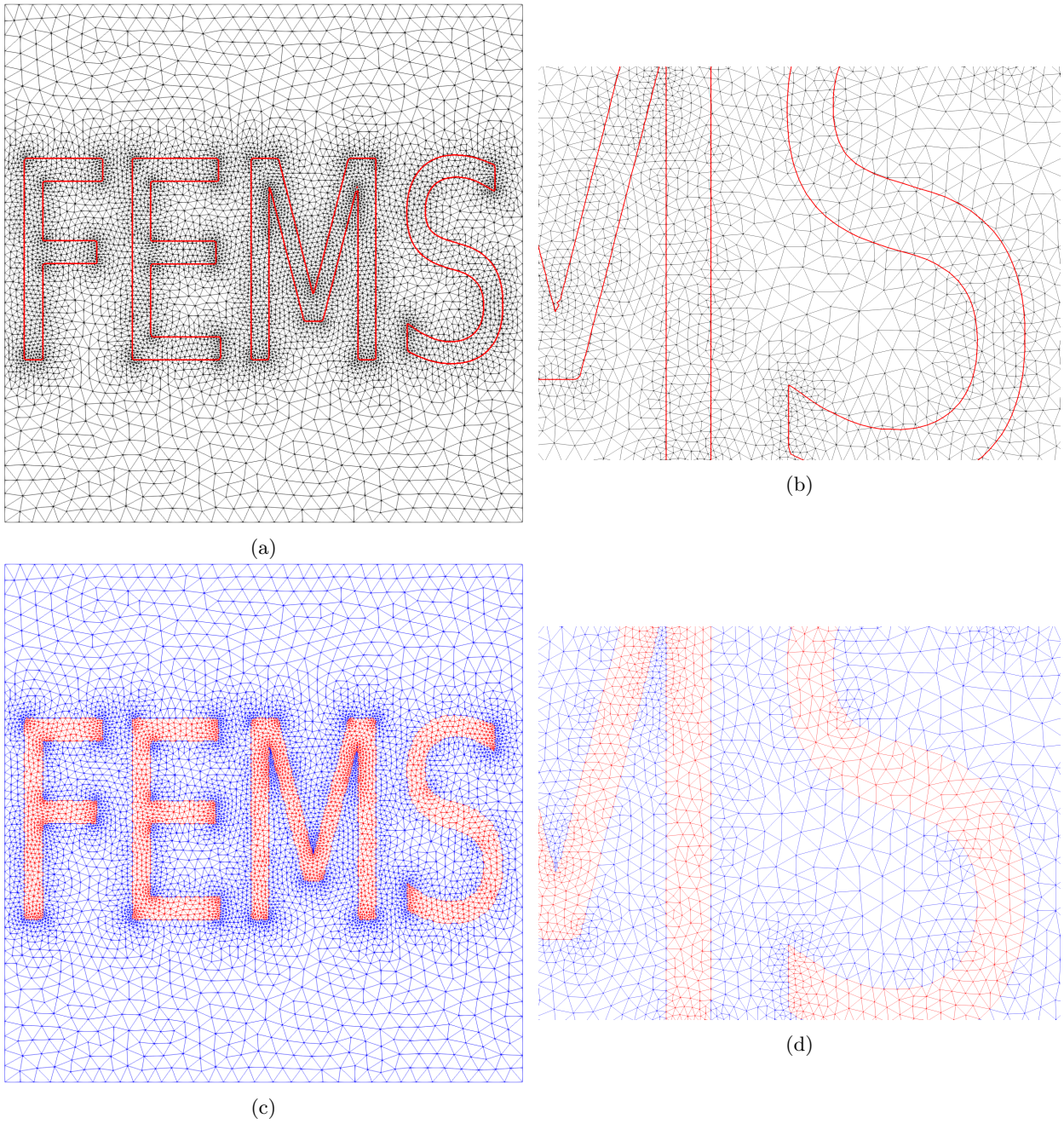


Figure 6: Last steps of 2D mesh generation for the FEMS image using $h_c = 0.128$: (a,b) mesh adapted 8 times using the isotropic curvature-based metric tensor field, (c,d) mesh after internal fitting and body-fitted mesh adaption.

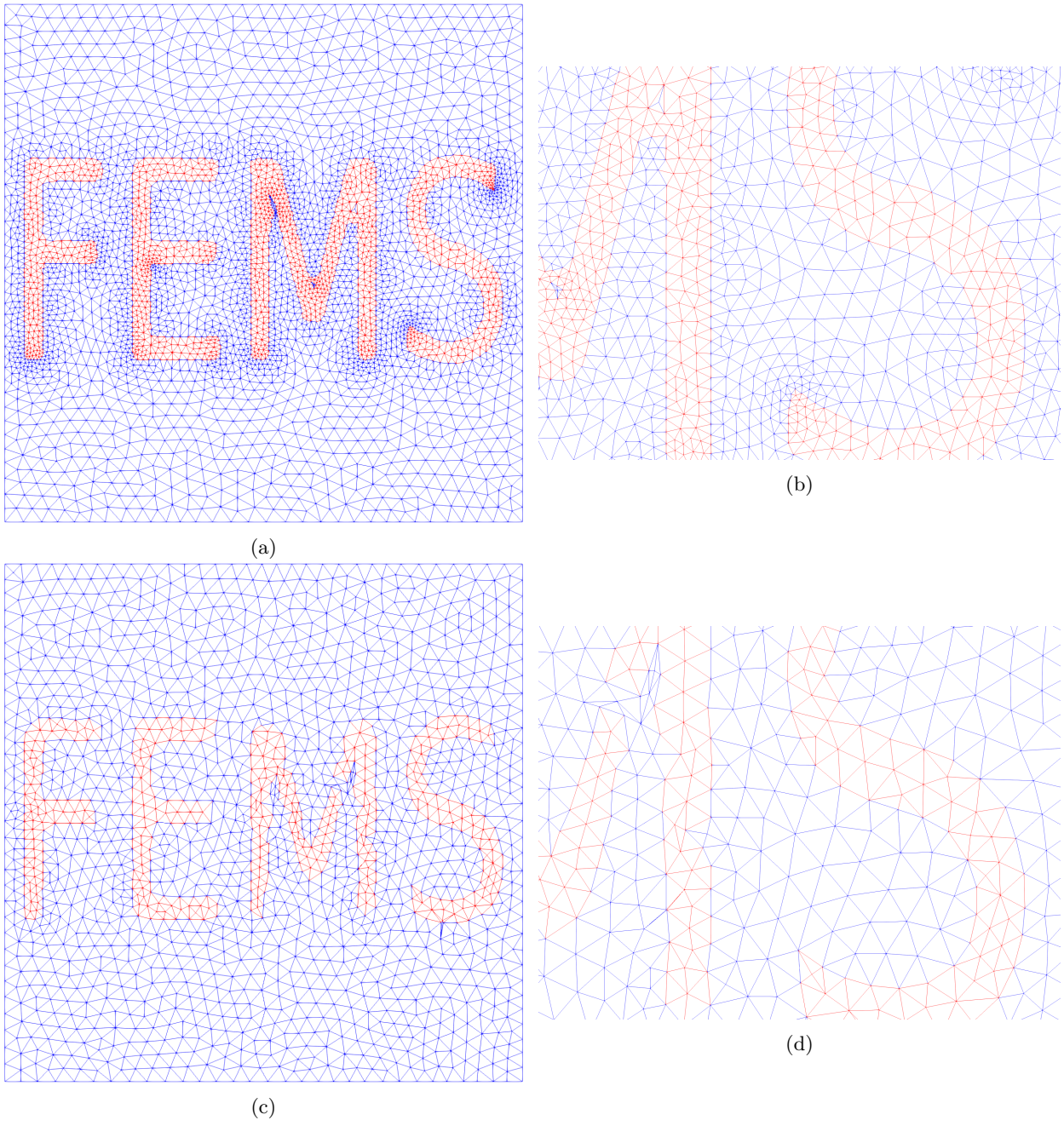


Figure 7: Final 2D meshes for the FEMS image using: (a,b) $h_c = 0.256$, (c,d) $h_c = 0.512$.

Resulting meshes are shown in Fig. 8 using different values of control parameter h_c. Clearly, this parameter has no effect away from the letters, where the mesh size is set to h_{max}. Close to the letters, of which only the external surface is shown in the figure, the mesh is refined with lower h_c, as was observed in the 2D case. To inspect more closely whether fine features are well represented, a zoomed view is shown in Fig. 9. With $h_c = 0.512$, some corners of the E letter are lost. These might not be relevant for linear mechanics, for instance to estimate linear elastic properties. However, for other applications such as fatigue life prediction, stress concentration is very important and thus h_c should be chosen carefully.
For these 3D simulations, mesh generation entails a significant computation time and thus has been measured. The initial structured FE mesh has about 5,000 elements. The mesh with $h_c = 0.512$, which has about 240,000 elements, is generated in about 100 s. The mesh with $h_c = 0.128$ has about 690,000 elements and is generated in about 500 s. This computation time is spent on mesh modification operations, all other operations having a negligible cost (LS function interpolation, metric computation). This justifies ongoing research work on parallel remeshing strategies.

As a conclusion, both in 2D and 3D, internal interfaces can be introduced in an FE mesh in FEMS using an LS function. The body-fitted FE mesh that is produced by FEMS can be controlled in terms of accuracy and how well fine and sharp features are captured. This control is achieved through the metric parameters, and especially the control parameter h_c, for which the value must be chosen carefully.

4.2 Localization and interface tracking problems in fluid mechanics

This second set of simulations shows the capabilities of FEMS for computational fluid dynamics. Simulations are conducted in the transient regime and the mesh is adapted several times during simulations to track and capture new flow features that appear and evolve.

4.2.1 Transient incompressible flow

This first problem is that of the well-known lid-driven cavity in 2D [1], which requires the solution of Navier-Stokes equations for Newtonian incompressible flow. This solution is performed using an Eulerian mesh and the mixed formulation with the Taylor-Hood P2/P1 pair and SUPG stabilization presented in Subsec. 2.2. The fluid mass density is fixed to $1\,\mathrm{g\,mm^3}$, gravity is neglected and the fluid dynamic viscosity is fixed to $0.2\,\mathrm{kPa\,ms}$ so that the Reynolds number is equal to $5,000$. Boundary conditions are given in Fig. 10.
The time step for this transient simulation is 1 ms, and the simulation is stopped when, over a time increment, the relative change in $L^2(\Omega)$ norm of the velocity field is below 10^{-6}. The whole FE mesh is automatically adapted at the beginning of a time increment when the quality of at least one element as defined by Eq. (14) is found to be below $^1/_3$. The metric tensor field is that of Eqs. (10) and (11), with $\mathbf{s} = \mathbf{v}$ and two metric parameters h_{max} and $\mathcal{N}_c$ to prescribe. The remeshing algorithm is that of the first strategy described in Par. 2.3.2.
Note that as a result of this scheme there is no significant change of the solution in the last time increments. Consequently, the metric tensor field does not vary significantly either, and no mesh adaption is triggered in those increments. Convergence is thus achieved both regarding the solution and the mesh.

Two potential uses of unstructured anisotropic mesh adaption are investigated. The first one consists in trying to obtain a more accurate solution with a similar number of degrees of freedom. The second one consists in trying to obtain a higher convergence rate when increasing the number of degrees of freedom, as compared to uniform mesh refinement.
Velocity fields using a fixed complexity are shown in Figs. 13(a,b) and 14(a,b) and compared with a reference result computed using a uniform grid mesh of 601×601 in Ref. [55]. Obviously, local mesh refinement is of great interest for this case and it can clearly be seen when comparing the curves for $h_{max} = 1$ and $h_{max} = 4$. A higher ratio of element size and stretching is also interesting, especially in regions where the flow is nearly unidirectional, as shown in Fig. 11. Elements get progressively stretched along the horizontal direction at the top of the cavity with increasing h_{max}. Refined but isotropic elements are automatically placed in the

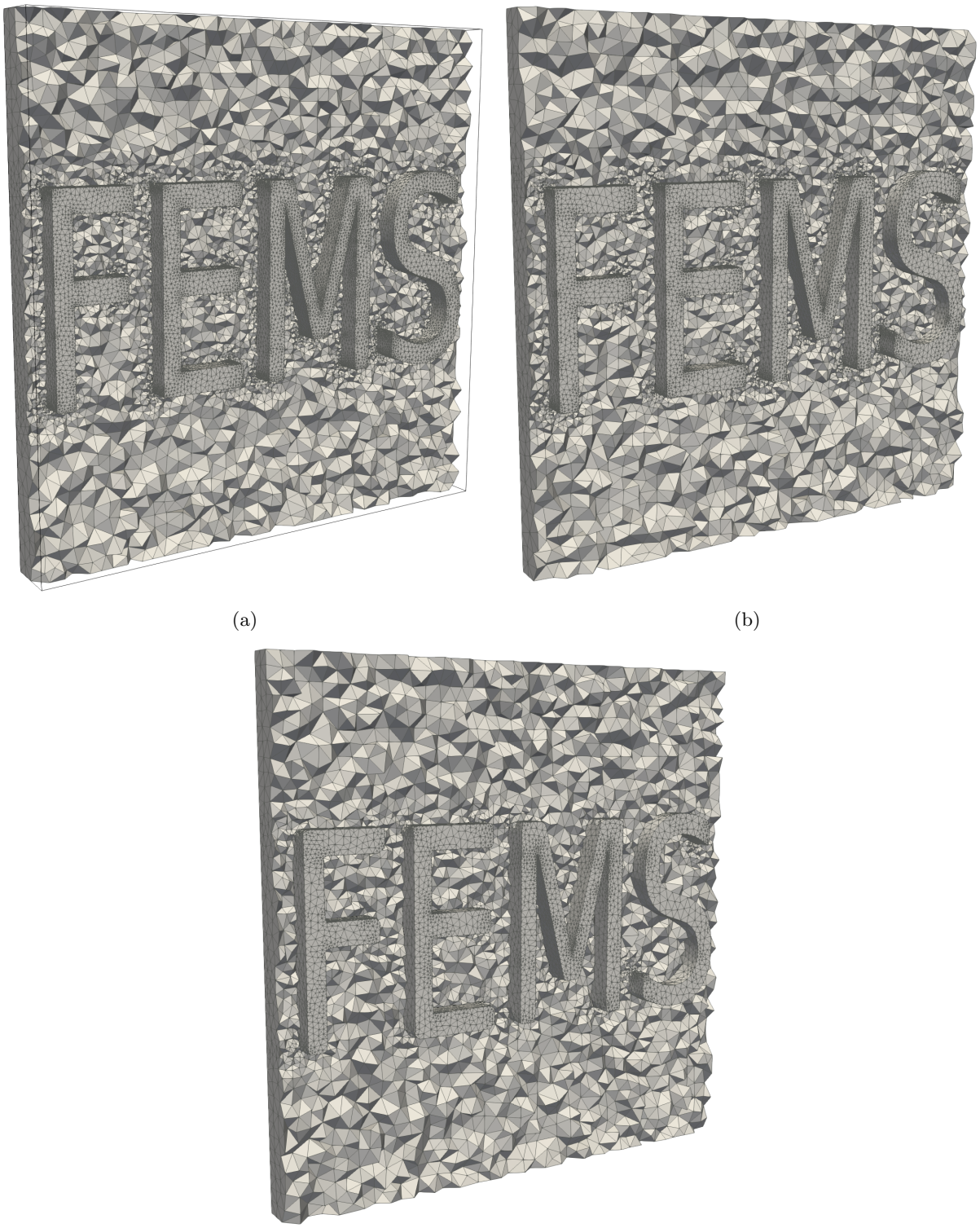
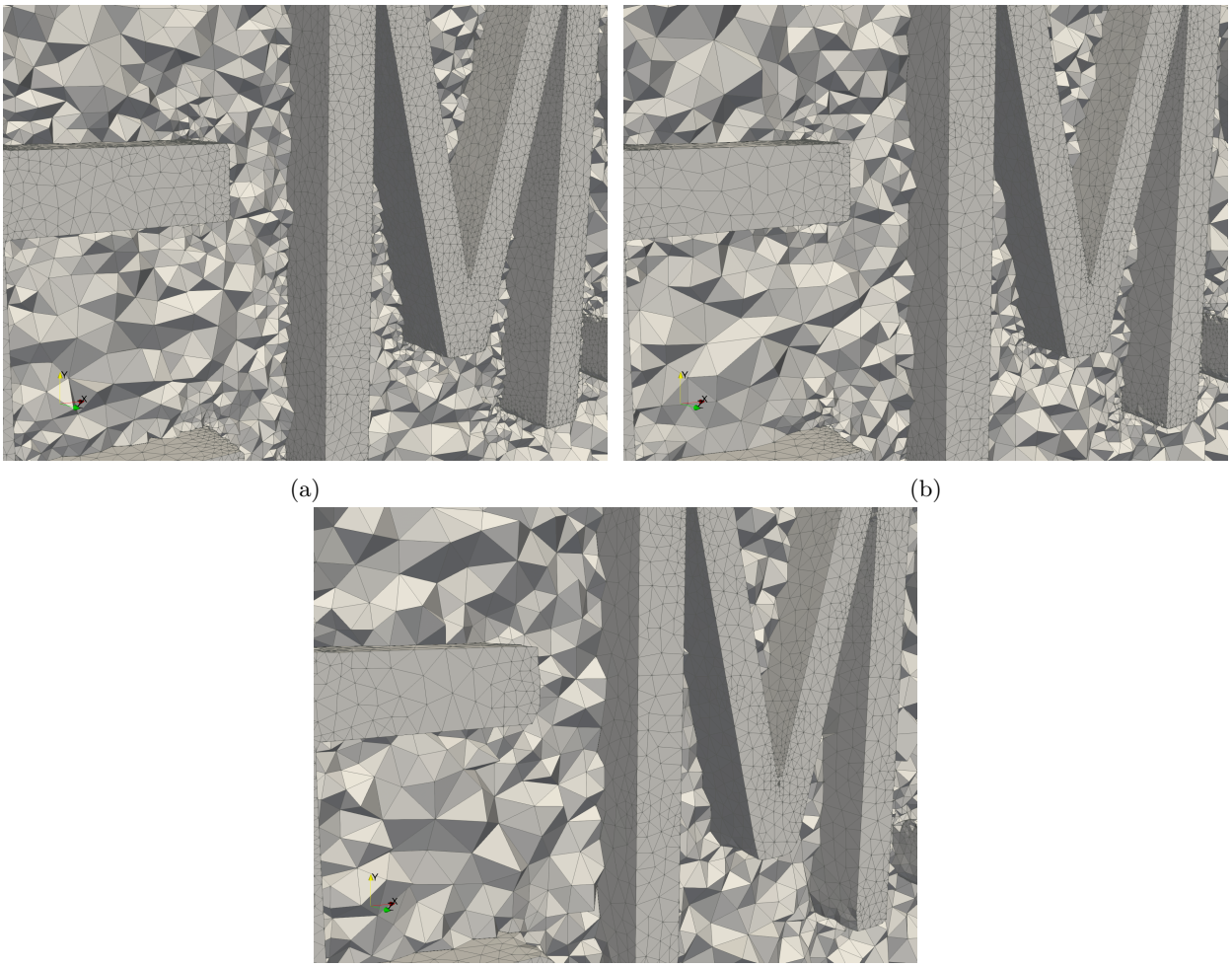


Figure 8: Inside view of final 3D meshes for the FEMS image using: (a) $h_c = 0.128$, (b) $h_c = 0.256$, (c) $h_c = 0.512$.



(a) (b)

(c)

Figure 9: Inside zoomed view of final 3D meshes for the FEMS image using: (a) $h_c = 0.128$, (b) $h_c = 0.256$, (c) $h_c = 0.512$.

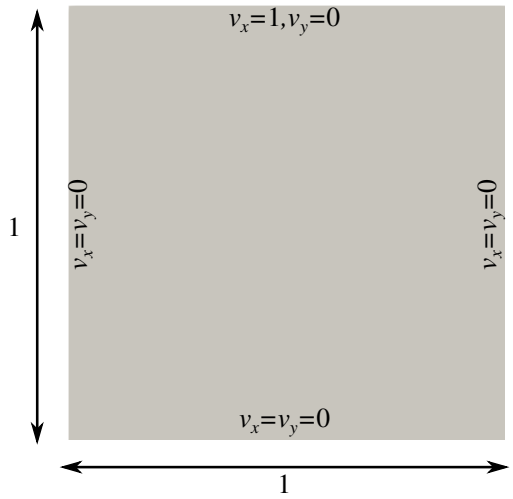


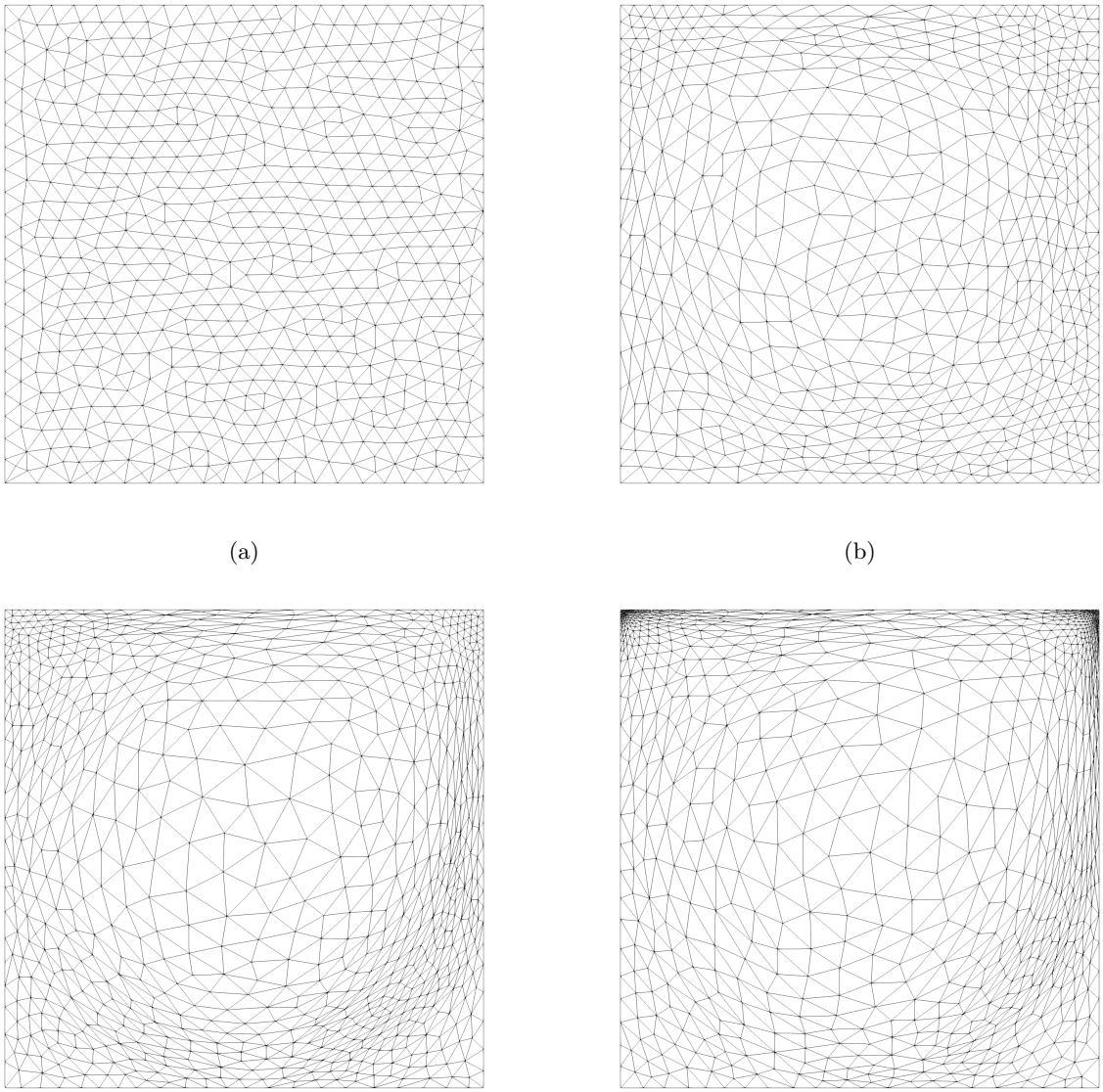
Figure 10: Boundary conditions for the 2D lid-driven cavity problem. Lengths are in millimeters and velocities in millimeters per millisecond.

two upper corners where the velocity field is singular. Because P2 interpolation is used for the velocity, excessive mesh refinement is not needed in the lower corners of the cavity in order to capture the secondary vortexes.
Velocity fields using uniform isotropic mesh refinement with $h_{max} = 1$ are shown in Figs. 13(c,d) and 14(c,d) and can be compared with results using local anisotropic mesh refinement with $h_{max} = 1000$ shown in Figs. 13(e,f) and 14(e,f). Asymptotic convergence does not seem to be reached using uniform isotropic mesh refinement for $\mathbf{v_y}$ in Fig. 14(d), while it seems clearly obtained for both components of the velocity field using local anisotropic mesh refinement. As shown in Fig. 12, this is due to the improvement of the discretization in the upper corners of the cavity using local anisotropic mesh refinement, while edge length in some regions like the center of the cavity remains constant.

Computation times vary mainly depending on the number of degrees of freedom and the number of time increments needed to reach convergence. Additionally, there is an approximation error on the complexity constraint in Eq. (11), which means the final number of P1 nodes generally does not match exactly $\mathcal{N}_c$. This error on the complexity constraint is known to be more significant with higher anisotropy but to decrease with mesh refinement [25].
Computation times are given in Tab. 1. For mesh adaption, they include metric computation, mesh modification operations and variables transfer.

$\mathcal{N}_c$	h_{max}	# P1 nodes	# Time increments	# Mesh adaption	Mesh adaption	Solution
512	1	663	799	1	8 s	244 s
512	4	668	681	2	8 s	172 s
512	16	817	651	11	11 s	179 s
512	1000	928	658	13	11 s	194 s
1024	1	1275	710	2	13 s	334 s
2048	1	2438	697	2	25 s	743 s
1024	1000	1573	664	19	20 s	353 s
2048	1000	2788	678	36	51 s	775 s

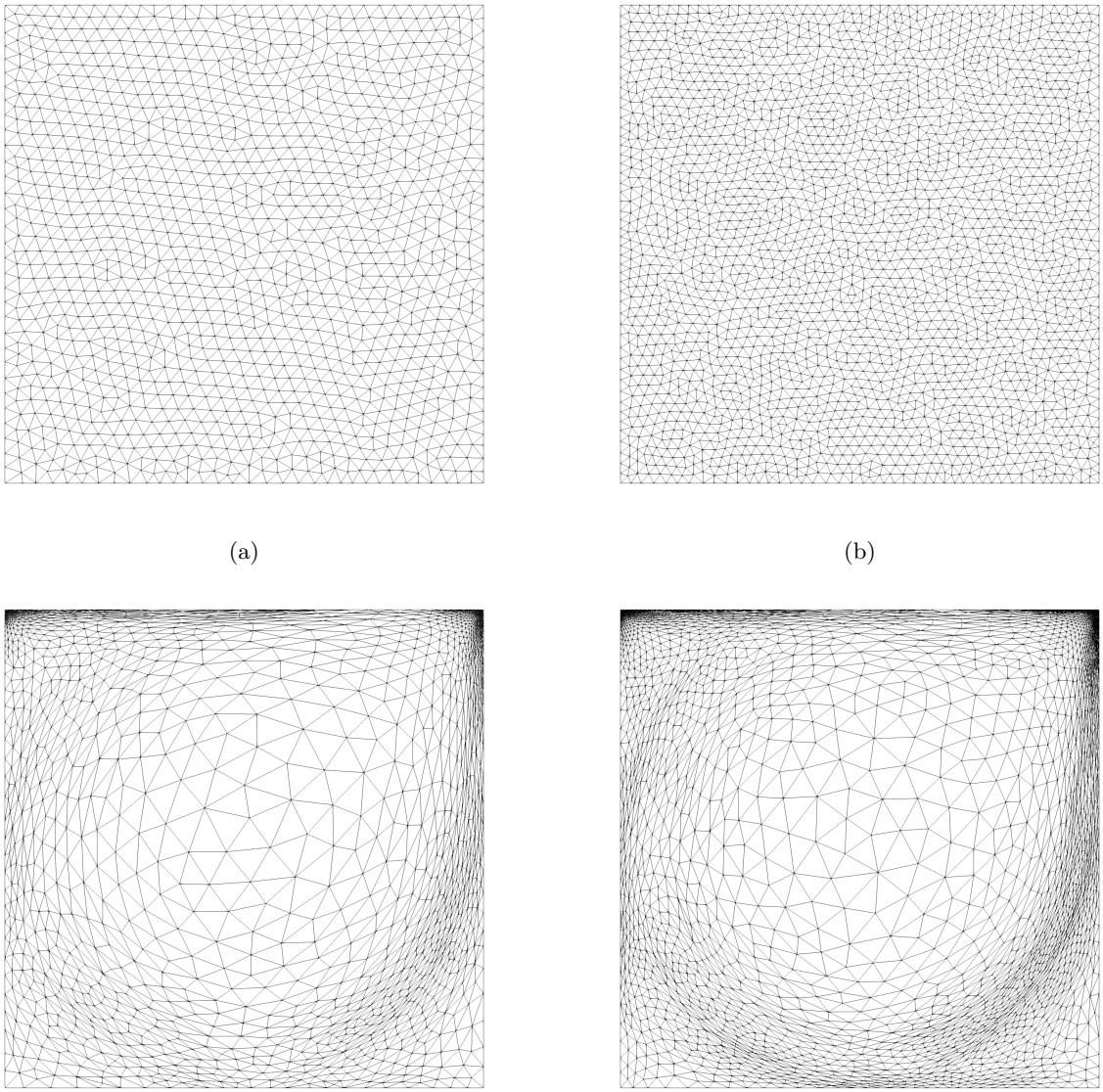
Table 1: Final number of P1 nodes, total number of time increments, total number of mesh adaptions, total mesh adaption time and total Navier-Stokes equations FE solution time for lid-driven simulations.



(a) (b)

(c) (d)

Figure 11: Final adapted meshes for the lid-driven cavity computed using $\mathcal{N}_c = 512$ and: (a) $h_{max} = 1$, (b) $h_{max} = 4$, (c) $h_{max} = 16$, (d) $h_{max} = 1000$.



(a) (b)

(c) (d)

Figure 12: Final adapted meshes for the lid-driven cavity computed using: (a) $\mathcal{N}_c = 1024$ and $h_{max} = 1$, (b) $\mathcal{N}_c = 2048$ and $h_{max} = 1$, (c) $\mathcal{N}_c = 1024$ and $h_{max} = 1000$, (d) $\mathcal{N}_c = 2048$ and $h_{max} = 1000$.

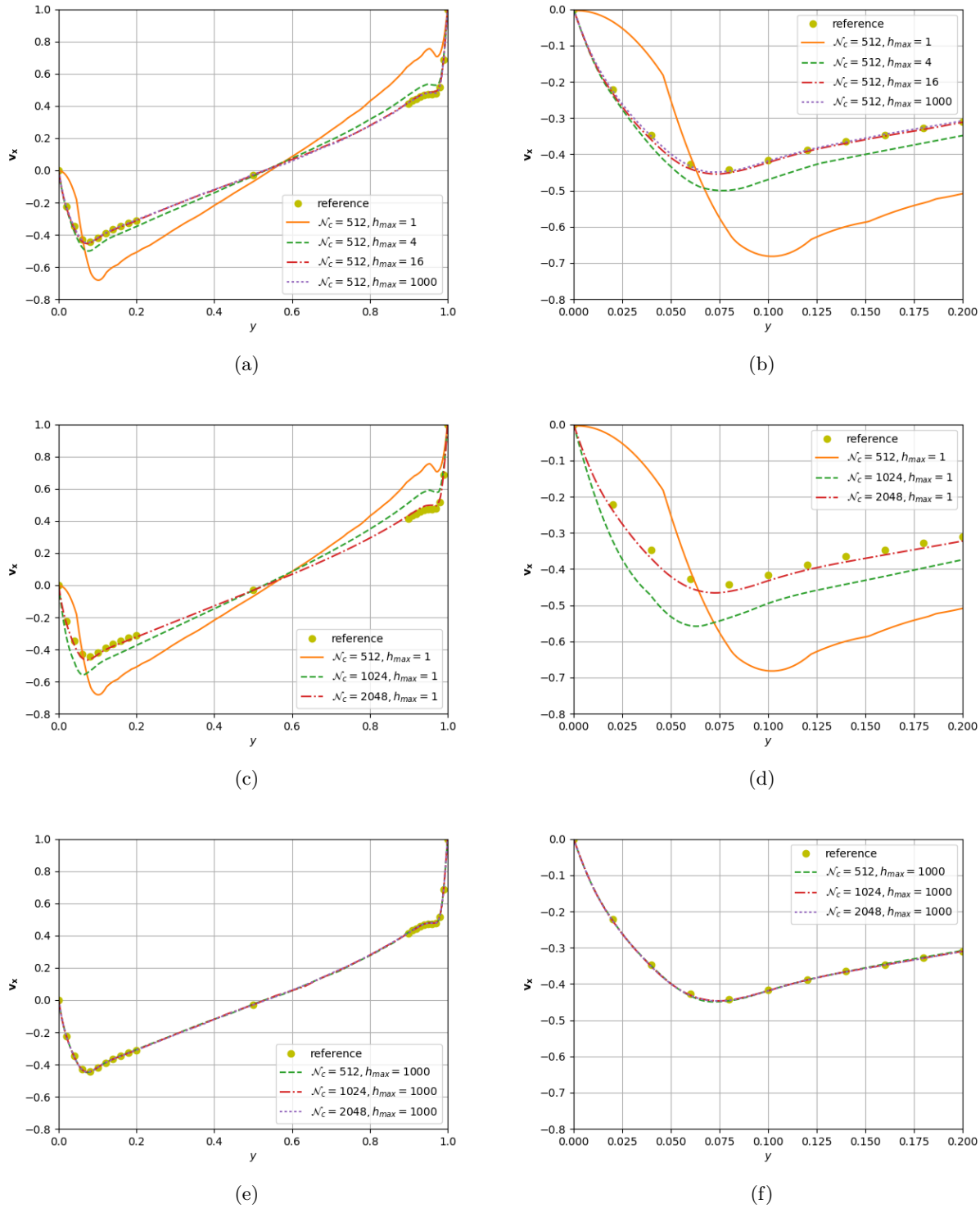


Figure 13: Final velocity component $\mathbf{v}_x$ along a vertical line passing through the geometric center of the lid-driven cavity computed using: (a,b) $\mathcal{N}_c = 512$ and different values of h_{max}, (c,d) $h_{max} = 1$ and different values of $\mathcal{N}_c$, (e,f) $h_{max} = 1000$ and different values of $\mathcal{N}_c$. Reference results are from Ref. [55].

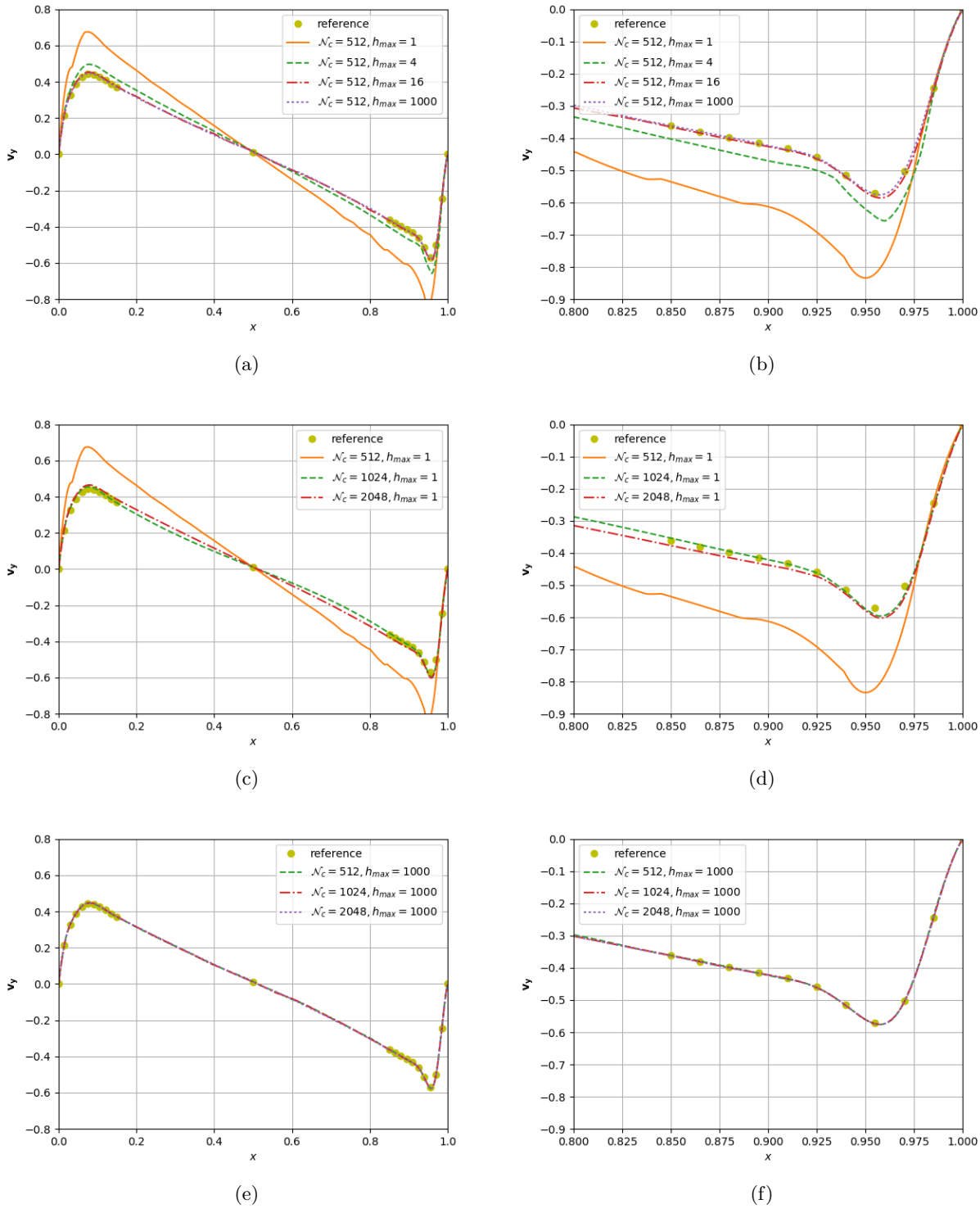


Figure 14: Final velocity component $\mathbf{v}_y$ along a horizontal line passing through the geometric center of the lid-driven cavity computed using: (a,b) $\mathcal{N}_c = 512$ and different values of h_{max}, (c,d) $h_{max} = 1$ and different values of $\mathcal{N}_c$, (e,f) $h_{max} = 1000$ and different values of $\mathcal{N}_c$. Reference results are from Ref. [55].

There is clearly an increase of the error between the prescribed and the obtained number of P1 nodes with higher anisotropy. This error is reduced when increasing $\mathcal{N}_c$, as it is of 81% using $\mathcal{N}_c = 512$, 54% using $\mathcal{N}_c = 1024$, and 36% using $\mathcal{N}_c = 2048$.
The total computation time spent on mesh adaption varies mainly due to the variation of the number of P1 nodes and the fact that there are more mesh adaptions for some simulations. Clearly from Tab. 1, more mesh adaptions are needed to converge when using local anisotropic mesh refinement. This clearly leads to a computation time overhead when using anisotropic elements. For instance for $\mathcal{N}_c = 2048$ it is doubled.
However, the ratio between the computation time spent on mesh adaption and that spent on FE solution of Navier-Stokes equations is below 7%. The computational overhead of mesh adaption and in particular anisotropic mesh adaption is hence clearly affordable given the gain in accuracy that it provides.

4.2.2 Transient incompressible two-phase flow with surface tension and obstacles

The numerical framework for modeling two-phase flow problems with surface tension using FEMS has already been presented in a previous work [4]. To go beyond the latter, a 2D problem with a third phase is addressed in the following. A circular droplet is initially placed in the higher half of the domain. This liquid is of mass density $1\,\mathrm{kg\,mm^3}$ and dynamic viscosity $10\,\mathrm{MPa\,ms}$. The surface tension coefficient is $1.96\,\mathrm{g\,ms^{-2}}$. Rigid solid obstacles composed of the FEMS letters as shown in Fig. 16(a) are placed in the lower part of the domain. The rest of the domain is occupied by a gas of mass density $1\,\mathrm{g\,mm^3}$ and dynamic viscosity $0.1\,\mathrm{MPa\,ms}$.
No-slip boundary conditions are applied at all domain boundaries, including that of the solid phase. Due to gravity, which is equal to $0.98\,\mathrm{mm\,ms^{-2}}$, the liquid is expected to fall and flow along the solid, while the gas is expected to elevate, unless it is trapped due to the solid phase.

As described in Subsec. 2.2, the Navier-Stokes equations for Newtonian incompressible two-phase flow with the surface tension term are solved using a P2/P2/P2 RBVMS formulation where the velocity $\mathbf{v}$, the pressure p and the liquid phase LS function ϕ are fully coupled. The transition of fluid properties between liquid and gas is smoothed over a thickness $\epsilon = 8\,\mu\mathrm{m}$ using the regularized Heaviside function H_ϵ defined by

$$H_\epsilon(\phi) = \begin{cases} 1, & \phi > \epsilon, \\ \frac{1}{2}\left(1 + \frac{\phi}{\epsilon} + \frac{1}{\pi}\sin\left(\frac{\pi\phi}{\epsilon}\right)\right), & |\phi| \leq \epsilon \\ 0, & \phi < -\epsilon. \end{cases} \tag{16}$$

The time step for this transient simulation is initially $0.1\,\mathrm{ms}$ but it is automatically decreased when the Newton-Raphson solution of the RBVMS formulation fails to converge. GPU-accelerated LS reinitialization is performed at the beginning of each time increment to maintain the distance property of the LS function.

The remeshing algorithm of the second strategy described in Par. 2.3.2 is used to perform different kinds of mesh adaption in this multiphase flow simulation.
There is a first pre-processing step in order to generate a body-fitted FE mesh of the obstacles as done in Subsec. 4.1. Parameters h_{min} and h_{max} are set respectively to $8\,\mu\mathrm{m}$ and $32\,\mu\mathrm{m}$, while control parameter h_c is set to 0.128. The number of mesh adaption cycles is 8. The mesh generation strategy based on purely topological internal fitting is used [28]. The generated mesh is shown in Fig. 16(b). The triangles within the obstacles are then fixed and cannot change in later remeshing operations.
Then, in a second pre-processing step, the mesh is adapted in order to be locally refined close to the liquid interface. This is done using the metric tensor field of Eqs. (10) and (11), with $\mathbf{s} = H_\epsilon(\phi^0)$, where ϕ^0 is the discrete LS function at the start of the simulation. Anisotropy control parameter h_{max} is set to 1000, and the complexity $\mathcal{N}_c$ is set to 4096. Similarly to the first pre-processing step, multiple mesh adaption cycles, here 5, are needed. The resulting mesh is shown in Fig. 16(c). A close-up in 16(d) shows how the mesh is body-fitted for solid obstacles but not for the liquid phase.
The metric tensor field of Eqs. (10) and (11) is used during the simulation without changing the parameters.

However, the sensor variable is modified to $\mathbf{s} = (H_\epsilon(\phi^n), H_\epsilon(2\phi^n - \phi^{n-1}))$, where ϕ^n is the discrete LS function at the start of a time increment (before Newton-Raphson solution), and $2\phi^n - \phi^{n-1}$ is an extrapolation of ϕ^{n+1}. The whole FE mesh is automatically adapted at the beginning of a time increment when the quality of at least one element as defined by Eq. (14) is found to be below $^1/_3$.

The simulation reaches the final time $T = 10\,\text{ms}$ in 750 time increments, among which 454 included mesh adaption due to the quality drop criterion. The liquid interface as well as adapted FE meshes are shown in Fig. 15 at different time increments. The droplet enters in contact with the obstacles in the center in Fig. 15(a). It then spreads widely due to inertia in Fig. 15(b), and starts pouring down until the final state shown in Fig. 15(c).
Because the solid obstacles are explicitly meshed and are not modified during the simulation, there is exact conservation of the solid volume. This is interesting compared to methods where solid obstacles are represented implicitly [20]. An interesting error measure is gas volume, which should not change due to incompressibility and boundary conditions. The relative error on gas volume can be computed using

$$V_{gas}(t) = \int_\Omega H_\epsilon(-\phi(\mathbf{x}, t))\mathrm{d}\mathbf{x},$$

$$\text{Error}(V_{gas}) = \sqrt{\frac{\int_0^T (V_{gas}(t) - V_{gas}(0))^2 \mathrm{d}t}{\int_0^T (V_{gas}(0))^2 \mathrm{d}t}}.$$

An error of 2.23% is obtained, which clearly shows the relevance of anisotropic mesh adaption using a P2 interpolation for the LS function. This was already evidenced in a previous work [4], and is here confirmed for a problem with obstacles.
The close-up in Fig. 15(d) must be compared with that in Fig. 16(d). The challenge is that liquid volume does not change, but there is much more liquid interface to cover with a fixed complexity. This is achieved automatically by stretching the elements, as can be seen in Fig. 16(d).

Overall computation time is nearly 7 h, with 81% of that time spent on Newton-Raphson solution, and 14% on mesh adaption. The latter is more significant compared to the lid-driven cavity simulation. Every time the interface moves, the mesh must be adapted in order to keep fine elements in the region where fluid properties transition, which is also the region where the surface tension term is active. Thus, there are more mesh adaptions and the associated computation time increases. Future work on parallel computing applied to mesh adaption should be very relevant in that regard.
Looking at the very small size of the elements in the direction orthogonal to the interface in Fig. 16(d), it is easy to imagine the huge number of elements that would be necessary to get the same accuracy with a fixed mesh. Indeed, the mesh would then have to be refined in every part of the domain reached by the liquid phase. The 14% of computation time spent on mesh adaption can hence be considered as quite affordable.

4.3 Large deformation and localization problems in solid mechanics

This third and last set of simulations addresses computational solid mechanics and in particular loading conditions leading to large and/or localized strains. Although static conditions are assumed, the load is applied progressively and the Lagrangian mesh is automatically adapted several times during the simulations in order to maintain element quality and avoid element flipping.
A load increment consists in solving balance equations in order to obtain the incremental displacement vector field $\boldsymbol{\Delta}\mathbf{u}$, the Cauchy stress tensor field σ and state variables $\mathbf{q}$ for that increment. Then, $\boldsymbol{\Delta}\mathbf{u}$ is used to update mesh nodes coordinates, as per Eq. (1) which is standard for an updated Lagrangian formulation. The use of Eq. (1) might generate elements of negative volume, which cannot be accepted neither by the FE method nor the remeshing algorithms available in FEMS. Thus, if such elements are generated, the mesh motion algorithm rolls back and tries an update with $0.5\boldsymbol{\Delta}\mathbf{u}$ instead of the full displacement increment. If this fails again, $0.25\boldsymbol{\Delta}\mathbf{u}$ is used, etc. Once an acceptable displacement increment is found, mesh adaption is

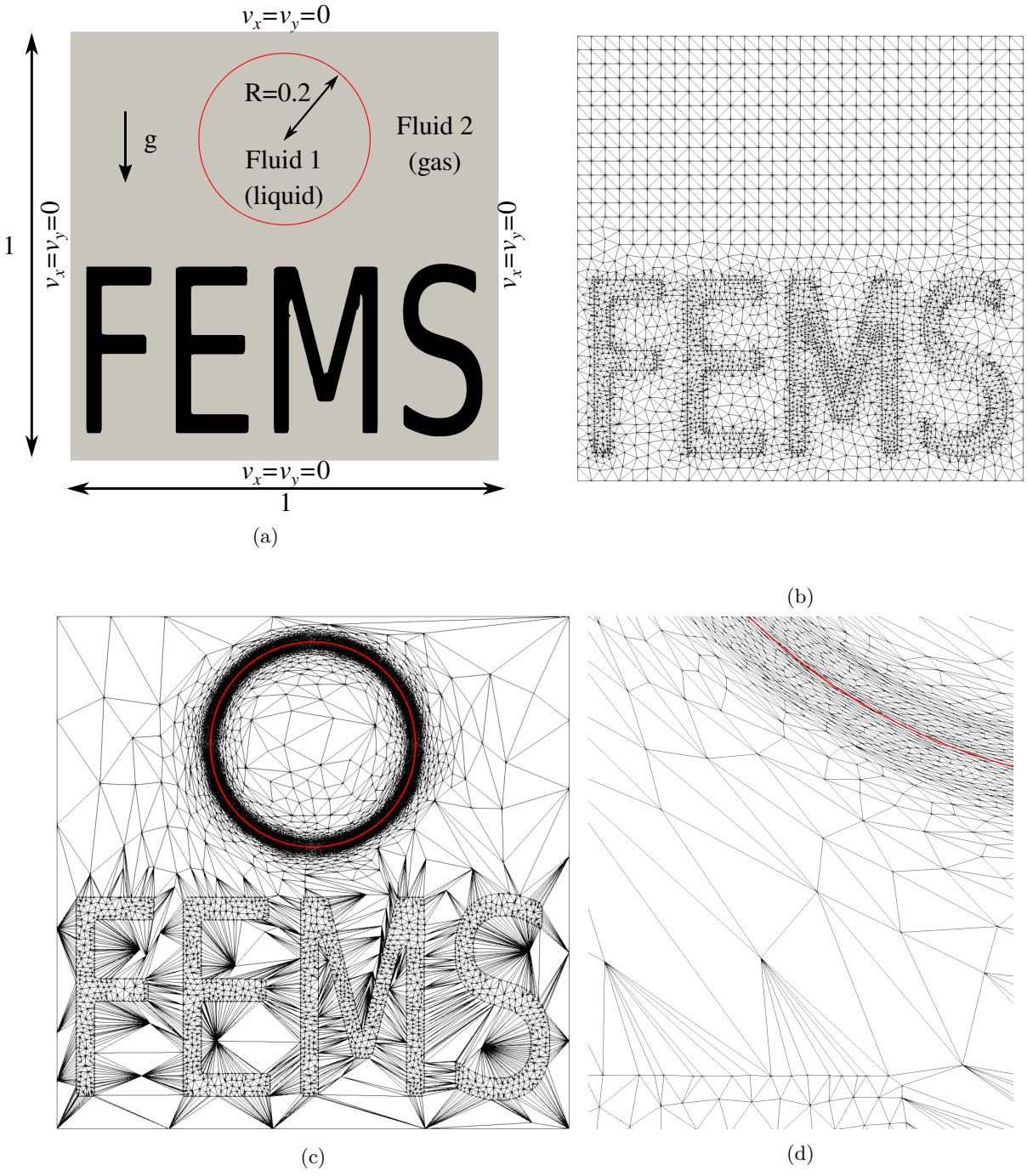


Figure 15: (a) Boundary conditions for the two-phase flow problem with obstacles. Lengths are in millimeters and velocities in millimeters per millisecond. (b) Generated body-fitted FE mesh of the obstacles. (c) Adapted FE mesh combining body-fitted meshing of obstacles and local anisotropic mesh adaption to the liquid phase LS function. (d) Zoom on the adapted mesh.

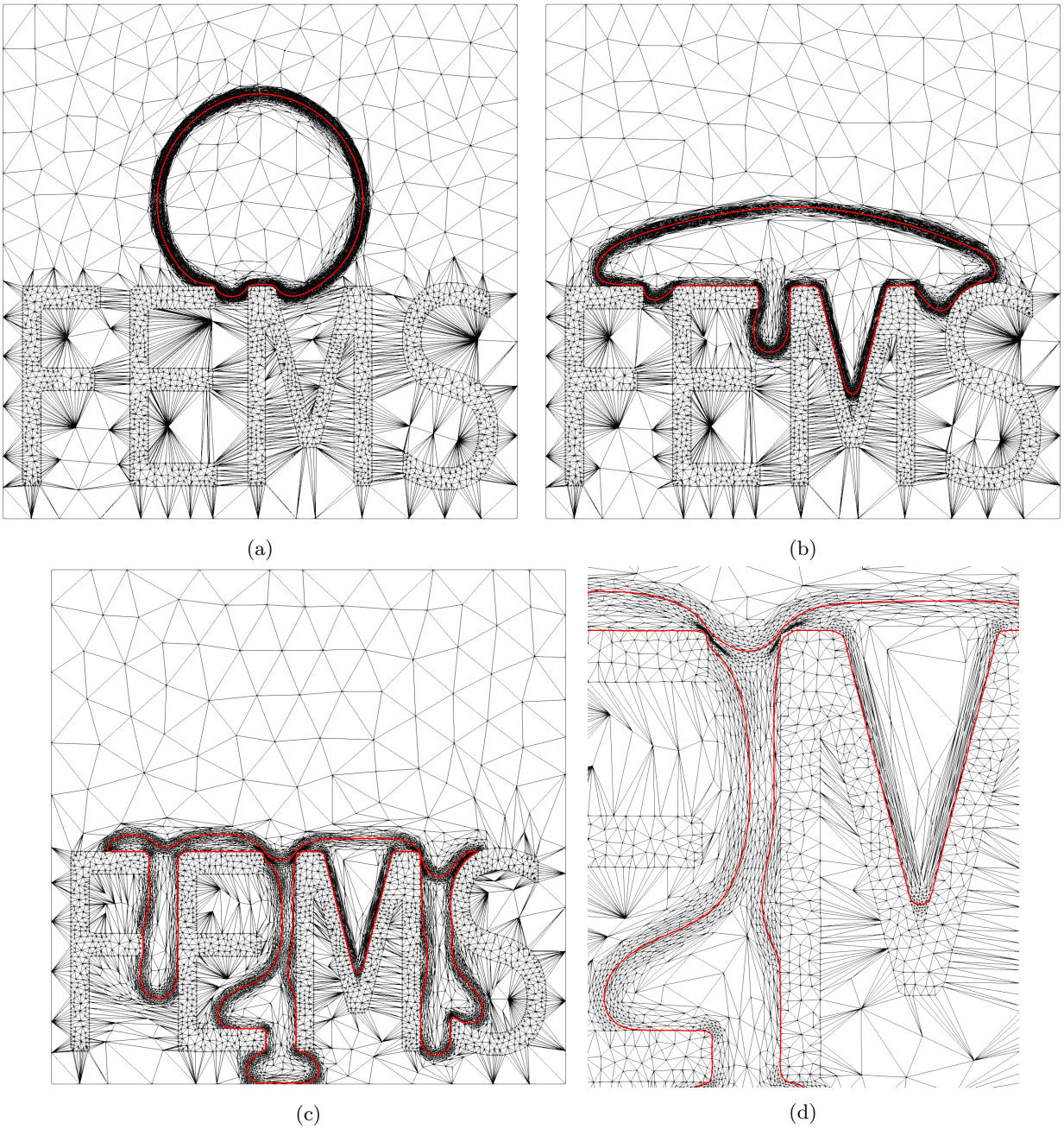


Figure 16: Results of the two-phase flow simulation with obstacles at: (a) $t = 0.5\,\text{ms}$, (b) $t = 1\,\text{ms}$, (c) $10\,\text{ms}$. (d) Zoom on the adapted mesh at $t = 10\,\text{ms}$.

triggered to improve element quality, and an attempt is made to apply the remaining displacement increment ($0.5\Delta\mathbf{u}$, $0.75\Delta\mathbf{u}$ or more depending on the current state). This mesh motion algorithm was used in Ref. [31] and systematically succeeds in applying the full displacement increment without any element flipping. After mesh motion, mesh adaption can be triggered if element quality decreased significantly, then the solution is output on the deformed mesh and the simulation continues with the next load increment.
For all solid mechanics simulations, the domain is the 3D box decomposed as a heterogeneous material with a matrix and two reinforcements as shown in Fig. 17(a). A body-fitted FE mesh is generated using the remeshing and mesh generation algorithms of the first strategy in 8 mesh adaption cycles with parameters h_{min} and h_{max} set respectively to 16 µm and 64 µm, and control parameter h_c set to 0.256. This initial mesh is shown in Fig. 17(b) and loaded horizontally as shown in Fig. 17(a), with an applied displacement U that is defined in the sequel.
The remeshing algorithm of the first strategy is also used during the simulations. It is coupled to the metric tensor field of Eqs. (9) and (11), with $\mathbf{s} = \Delta\mathbf{u}$. Anisotropy control parameter is set to 10 as excessively stretched elements are not well-suited to a Lagrangian mesh, and the complexity $\mathcal{N}_c$ is varied using values 512, 1024 and 2048. Because this metric tensor field is not identical to the one used when generating the initial mesh, mesh adaption is always triggered at the end of the first load increment. It is then triggered again when there is element flipping during mesh motion and when after mesh motion the quality of at least one element has dropped more than twice since the last time the mesh was adapted. Simulation results for different material behaviors are presented in the sequel.

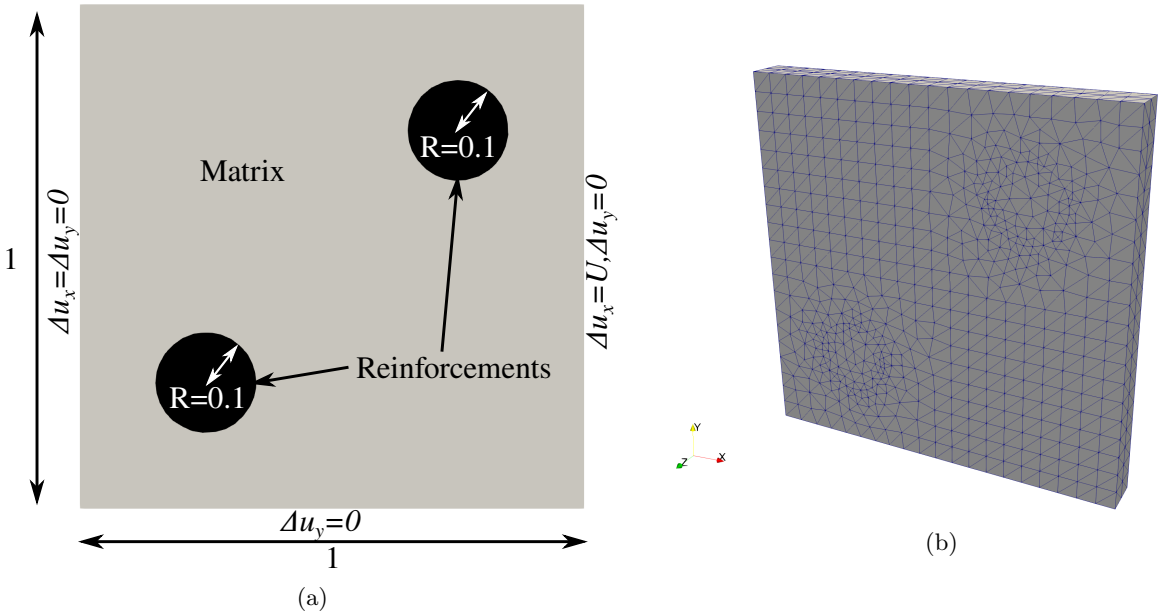


Figure 17: (a) Boundary conditions for solid mechanics problems. Lengths are in millimeters and the thickness in the third direction (not shown in this figure) is 0.1 mm. (b) Initial body-fitted FE mesh of the heterogeneous material for all solid mechanics simulations.

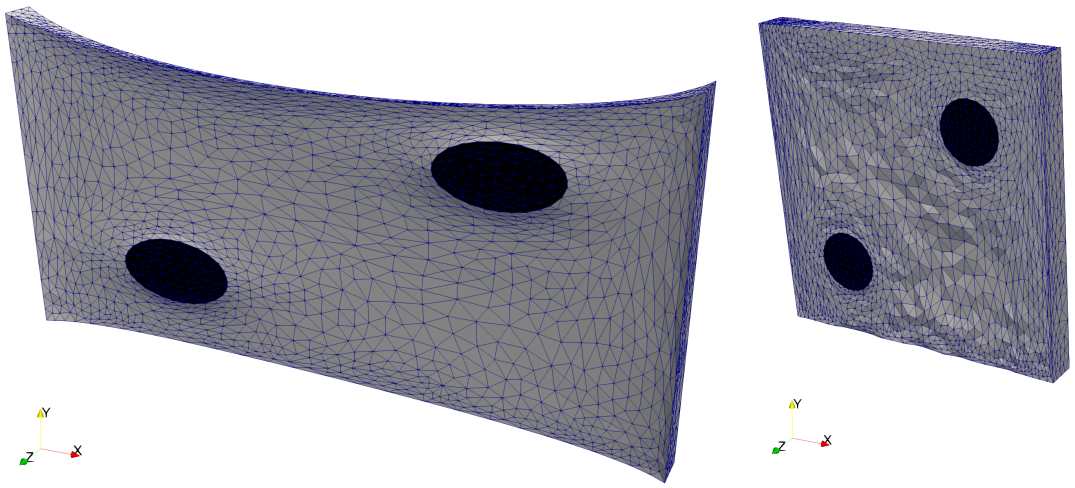
4.3.1 Hyperelasticity

As a first test, material behavior in both the matrix and the reinforcements is defined as hyperelastic, with a Saint Venant-Kirchhoff model, which means that the Cauchy stress tensor σ at any point is given by the relations

$$\sigma = \frac{1}{J}\mathbf{F}\mathbf{S}\mathbf{F}^T, J = \det(\mathbf{F}),$$

$$\mathbf{S} = 2\mu^{el}\mathbf{E} + \lambda^{el}\mathrm{tr}(\mathbf{E})\mathbf{I}_2, \mathbf{E} = \frac{1}{2}(\mathbf{C} - \mathbf{I}_2), \mathbf{C} = \mathbf{F}^T\mathbf{F},$$

where $\mathbf{F} = \mathbf{F}(\Delta\mathbf{u})$ is the deformation gradient tensor, $\mathbf{C}$ the right Cauchy-Green deformation tensor, $\mathbf{E}$ the Green-Lagrange strain tensor, $\mathbf{S}$ the second Piola-Kirchhoff stress tensor and $\mathbf{I}_2$ the identity tensor. Lamé parameters μ^{el} and λ^{el} depend on the Young's modulus, which is set to 200 MPa for the matrix and 400 MPa for the reinforcements, and the Poisson's coefficient which is set to 0.3 for the matrix and 0.2 for the reinforcements.
The finite strain updated Lagrangian weak form is discretized using standard P1 FEs and a Newton-Raphson algorithm as the relation between the Cauchy stress tensor and the displacement increment is nonlinear (see *e.g.*, the finite strain chapter in Ref. [12]). The sparse linear problem at each Newton-Raphson increment is solved using the GMRES linear solver with ILUT preconditioning of *Lis* [49]. The loading consists in 25 increments with U set to 0.04 mm, and then 25 increments with U set to -0.04 mm.



(a) (b)

Figure 18: Results of the solid mechanics simulation with hyperelastic material behavior and $\mathcal{N}_c = 2048$ after: (a) 25 load increments, (b) 50 load increments.

Results are shown in Fig. 18. Even though $\mathcal{N}_c$ is varied with values 512, 1024 and 2048, mesh adaption is always triggered twice in the three simulations: the first time at the end of the 1st load increment, and the second time around the 23rd load increment. The mesh motion algorithm always succeeds in applying the displacement increment in a single trial in all these simulations. Indeed, except from the regions close to the reinforcements, the purely elastic strain field is not significantly localized. Therefore, the large deformations are diffused and shared by all elements. The only reason mesh adaption is triggered twice is the excessive stretching which leads to a deteriorated element quality.
Mesh adaption results to some diffusion of the strain energy, which has the consequence that the final result in Fig. 18(b) is not identical to the 3D box shown in Fig. 17(b). If strain energy conservation is of interest, the user might choose to avoid mesh adaption as much as possible in these simulations. FEMS would still be of interest for such computational approach as it could be used to prepare the initial mesh.

4.3.2 Elasto-plasticity

In this second application, material behavior in the reinforcements is linear elastic but the matrix is defined as elasto-plastic with a von Mises yield criterion and linear isotropic hardening. The Cauchy stress tensor

σ at any point of the reinforcements is given by Hooke's law for isotropic linear elasticity with a Young's modulus of 400 MPa and a Poisson's coefficient of 0.2. In the matrix, the following nonlinear equations must be solved at each integration point:

$$\sigma = \sigma^{dev} - p\mathbf{I}_2, p = -\frac{1}{3}\mathrm{tr}(\sigma),$$

$$\dot{\sigma}^{dev} = 2\mu^{el}\left(\dot{\varepsilon}^{el} - \frac{1}{3}\mathrm{tr}(\dot{\varepsilon}^{el})\mathbf{I}_2\right), -\frac{1}{\chi^{el}}\dot{p} - \mathrm{tr}(\dot{\varepsilon}^{el}) = 0, \dot{\varepsilon}^{el} = \dot{\varepsilon} - \dot{\varepsilon}^{pl}, \mathrm{tr}(\dot{\varepsilon}^{pl}) = 0,$$

$$\overline{f} = \sqrt{\frac{3}{2}\sigma^{dev} : \sigma^{dev}}, \overline{\varepsilon}^{pl} = \int_0^t \dot{\overline{\varepsilon}}^{pl}(\tau)\mathrm{d}\tau, \dot{\overline{\varepsilon}}^{pl} = \sqrt{\frac{2}{3}\dot{\varepsilon}^{pl} : \dot{\varepsilon}^{pl}}, \overline{\sigma} = 500 + 1000\overline{\varepsilon}^{pl},$$

$$\left| \begin{array}{l} \dot{\overline{\varepsilon}}^{pl} = 0, \dot{\varepsilon}^{pl} = \mathbf{0}, \overline{f} < \overline{\sigma}, \\ \dot{\varepsilon}^{pl} = \frac{3}{2}\dot{\overline{\varepsilon}}^{pl}\frac{\sigma^{dev}}{\overline{f}}, \dot{\overline{\varepsilon}}^{pl} > 0, \overline{f} = \overline{\sigma}. \end{array} \right.$$

These equations involve the deviatoric part σ^{dev} of the Cauchy stress tensor and the hydrostatic pressure p. There is an additive decomposition of the strain rate tensor $\dot{\varepsilon}$ into an elastic part $\dot{\varepsilon}^{el}$ and a plastic part $\dot{\varepsilon}^{pl}$, the plastic part being incompressible. The equivalent stress $\overline{f}$ is defined using the von Mises yield criterion, and the yield stress $\overline{\sigma}$ using a linear isotropic hardening law which is a function of the only state variable, namely the equivalent plastic strain $\overline{\varepsilon}^{pl}$.
The last system of equations defines the conditions for plastic flow. This system is solved using a standard predictor-corrector scheme with return mapping [3, 28], where the only unknown is the equivalent plastic strain rate $\dot{\overline{\varepsilon}}^{pl}$. A Young's modulus of 200 MPa and a Poisson's coefficient of 0.3 are defined for the matrix in order to compute the Lamé parameter μ^{el} and the bulk modulus χ^{el}.
The small strain updated Lagrangian weak form is discretized using a $\mathrm{P1}^+$/P1 MINI element to deal with the incompressibility of plastic strains. This weak form is solved using a Newton-Raphson algorithm in addition to the local Newton-Raphson procedure that has already been mentioned for the return mapping at each integration point. The loading consists in 25 increments with U set to 0.04 mm.

The result for the simulation with the finest mesh is shown in Fig. 19. For these simulations with localized plastic strains, there is one mesh adaption for the simulation with $\mathcal{N}_c = 512$, two adaptions for $\mathcal{N}_c = 1024$, and three adaptions for $\mathcal{N}_c = 2048$. Although none of them has been triggered by the mesh motion procedure, element flipping is very likely in these simulations and maintaining a good element quality is very important. In addition, it is clear in Fig. 19 that the mesh has been automatically refined in the region with large plastic strain.

Finally, for all solid mechanics simulations, the computation time spent solving balance equations is always around 20 times the time spent on mesh adaption. Given that some of these simulations, especially those with plasticity, could not be conducted without mesh adaption, it is relevant to apply FEMS to solid mechanics as well as fluid mechanics.

5 Conclusions and perspectives

A state-of-the-art Finite Element Modeling Software (FEMS) has been presented in this paper. FEMS is targeted at engineers and scientists addressing localization problems. Those include a wide range of computational fluid dynamics problems involving turbulence and boundary layers, or multiphase flows, but also computational solid mechanics problems such as plastic localization bands. Examples of such problems are addressed in this paper to show the capabilities of FEMS.
A transient incompressible flow problem with a high Reynolds number has been solved using the higher-order mesh adaption capabilities of FEMS. A mesh composed of triangles has been automatically adapted

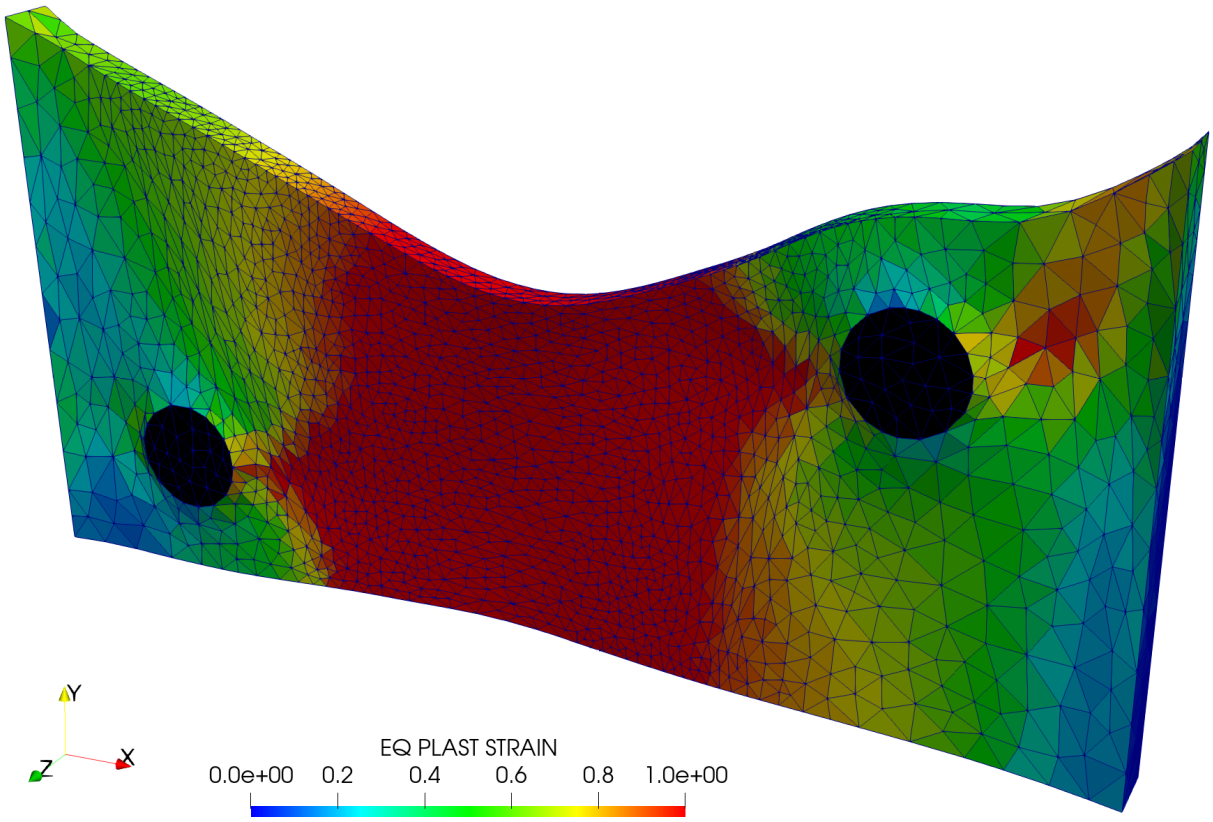


Figure 19: Results of the solid mechanics simulation with elasto-plastic material behavior for the matrix, linear elastic behavior for the reinforcements, and $\mathcal{N}_c = 2048$ after 25 load increments.

to the features of the solution, with triangles refined, stretched and oriented according to the fluid velocity field. Control parameters allowing to keep a fixed number of nodes during the simulation and to fix element stretching have been varied, showing that anisotropic mesh adaption provides a gain in accuracy with no significant computation time overhead.
Simulations in a heterogeneous domain have also been solved using FEMS. Two different modeling approaches have been proposed: the one relying on a body-fitted FE mesh with explicit meshing of internal interfaces, and the other relying on the Level-Set (LS) method to represent interfaces implicitly in the FE mesh. The first approach has been demonstrated to be robust for solid mechanics problems in an updated Lagrangian setting as FEMS can adapt the mesh automatically during the simulation to avoid element flipping. The second approach has been shown to be efficient for multiphase flow problems. For the latter, FEMS' higher-order interpolation of the LS function improves further the accuracy of the solution.
In addition, body-fitted FE meshes can be generated within FEMS, simply from signed distance functions. Those can be easily computed for most geometrical entities, and even from gray scale two-dimensional and three-dimensional images. This feature has been demonstrated in the paper with varying control parameters allowing to automatically vary the mesh size depending on the local maximum principal curvature of internal interfaces.
All in all, FEMS proves to be a powerful tool for modeling nonlinear phenomena in computational mechanics, and give access to cutting-edge adaptive finite elements for the first time in an open source software.

References

[1] R. Codina, "Stabilized finite element approximation of transient incompressible flows using orthogonal subscales," Computer Methods in Applied Mechanics and Engineering, vol. 191, pp. 4295–4321, aug 2002.

[2] G. Compère, J.-F. Remacle, J. Jansson, and J. Hoffman, "A mesh adaptation framework for dealing with large deforming meshes," International Journal for Numerical Methods in Engineering, vol. 82, pp. 843–867, 2010.

[3] E. Roux, M. Shakoor, M. Bernacki, and P.-O. Bouchard, "A new finite element approach for modelling ductile damage void nucleation and growth—analysis of loading path effect on damage mechanisms," Modelling and Simulation in Materials Science and Engineering, vol. 22, p. 075001, oct 2014.

[4] M. Shakoor and C. H. Park, "A higher-order finite element method with unstructured anisotropic mesh adaption for two phase flows with surface tension," Computer Methods in Applied Mechanics and Engineering, vol. (Submitted), 2019.

[5] F. Feyel, "Multiscale FE2 elastoviscoplastic analysis of composite structures," Computational Materials Science, vol. 16, pp. 344–354, dec 1999.

[6] N. Sukumar, D. Chopp, N. Moës, and T. Belytschko, "Modeling holes and inclusions by level sets in the extended finite-element method," Computer Methods in Applied Mechanics and Engineering, vol. 190, pp. 6183–6200, sep 2001.

[7] C. Geuzaine and J.-F. Remacle, "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities," International Journal for Numerical Methods in Engineering, vol. 79, pp. 1309–1331, sep 2009.

[8] C. Cecka, A. J. Lew, and E. Darve, "Assembly of finite element methods on graphics processors," International Journal for Numerical Methods in Engineering, vol. 85, pp. 640–669, feb 2011.

[9] T. Hughes, J. Cottrell, and Y. Bazilevs, "Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement," Computer Methods in Applied Mechanics and Engineering, vol. 194, pp. 4135–4195, oct 2005.

[10] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations," Journal of Computational Physics, vol. 79, pp. 12–49, nov 1988.

[11] A. Ern and J.-L. Guermond, Theory and Practice of Finite Elements, vol. 159 of Applied Mathematical Sciences. Springer New York, 2004.

[12] A. Fortin and A. Garon, "Les éléments finis : de la théorie à la pratique," 2011.

[13] O. Zienkiewicz, R. Taylor, and J. Zhu, The Finite Element Method: Its Basis and Fundamentals. Elsevier, 2013.

[14] D. Boffi, F. Brezzi, L. F. Demkowicz, R. G. Durán, R. S. Falk, and M. Fortin, Mixed Finite Elements, Compatibility Conditions, and Applications, vol. 1939 of Lecture Notes in Mathematics. Berlin, Heidelberg: Springer, 2008.

[15] A. N. Brooks and T. J. Hughes, "Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations," Computer Methods in Applied Mechanics and Engineering, vol. 32, pp. 199–259, sep 1982.

[16] Y. Bazilevs, V. Calo, J. Cottrell, T. Hughes, A. Reali, and G. Scovazzi, "Variational multiscale residual-based turbulence modeling for large eddy simulation of incompressible flows," Computer Methods in Applied Mechanics and Engineering, vol. 197, pp. 173–201, dec 2007.

[17] J. Yan, S. Lin, Y. Bazilevs, and G. Wagner, "Isogeometric analysis of multi-phase flows with surface tension and with application to dynamics of rising bubbles," Computers & Fluids, vol. 179, pp. 777–789, jan 2019.

[18] V. Arsigny, P. Fillard, X. Pennec, and N. Ayache, "Log-Euclidean metrics for fast and simple calculus on diffusion tensors," Magnetic Resonance in Medicine, vol. 56, pp. 411–421, aug 2006.

[19] 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, vol. 257, pp. 83–101, jan 2014.

[20] 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, vol. 268, pp. 65–81, jan 2014.

[21] M. Shakoor, M. Bernacki, and P.-O. Bouchard, "Ductile fracture of a metal matrix composite studied using 3D numerical modeling of void nucleation and coalescence," Engineering Fracture Mechanics, vol. 189, pp. 110–132, feb 2018.

[22] O. C. Zienkiewicz and J. Z. Zhu, "A simple error estimator and adaptive procedure for practical engineerng analysis," International Journal for Numerical Methods in Engineering, vol. 24, pp. 337–357, feb 1987.

[23] Z. Zhang and A. Naga, "A New Finite Element Gradient Recovery Method: Superconvergence Property," SIAM Journal on Scientific Computing, vol. 26, pp. 1192–1213, jan 2005.

[24] A. Loseille and F. Alauzet, "Continuous Mesh Framework Part I: Well-Posed Continuous Interpolation Error," SIAM J. Numer. Anal., vol. 49, no. 1, pp. 38–60, 2011.

[25] A. Loseille and F. Alauzet, "Continuous Mesh Framework Part II: Validations and Applications," SIAM Journal on Numerical Analysis, vol. 49, pp. 61–86, jan 2011.

[26] O. Coulaud and A. Loseille, "Very High Order Anisotropic Metric-Based Mesh Adaptation in 3D," Procedia Engineering, vol. 163, pp. 353–365, 2016.

[27] P. Laug and H. Borouchaki, "Construction d'un champ continu de métriques," Comptes Rendus Mathematique, vol. 351, pp. 639–644, aug 2013.

[28] M. Shakoor, M. Bernacki, and P.-O. Bouchard, "A new body-fitted immersed volume method for the modeling of ductile fracture at the microscale: Analysis of void clusters and stress state effects on coalescence," Engineering Fracture Mechanics, vol. 147, pp. 398–417, oct 2015.

[29] C. Dobrzynski and P. Frey, "Anisotropic Delaunay Mesh Adaptation for Unsteady Simulations," in Proceedings of the 17th International Meshing Roundtable (R. V. Garimella, ed.), pp. 177–194, Berlin, Heidelberg: Springer, 2008.

[30] C. Gruau and T. Coupez, "3D tetrahedral, unstructured and anisotropic mesh generation with adaptation to natural and multidomain metric," Computer Methods in Applied Mechanics and Engineering, vol. 194, pp. 4951–4976, nov 2005.

[31] M. Shakoor, P.-O. Bouchard, and M. Bernacki, "An adaptive level-set method with enhanced volume conservation for simulations in multiphase domains," International Journal for Numerical Methods in Engineering, vol. 109, pp. 555–576, jan 2017.

[32] C. Dapogny, C. Dobrzynski, and P. Frey, "Three-dimensional adaptive domain remeshing, implicit domain meshing, and applications to free and moving boundary problems," Journal of Computational Physics, vol. 262, pp. 358–378, apr 2014.

[33] J. X. Zhao, T. Coupez, E. Decencière, D. Jeulin, D. Cárdenas-Peña, and L. Silva, "Direct multiphase mesh generation from 3D images using anisotropic mesh adaptation and a redistancing equation," Computer Methods in Applied Mechanics and Engineering, vol. 309, pp. 288–306, 2016.

[34] M. Shakoor, A. Buljac, J. Neggers, F. Hild, T. F. Morgeneyer, L. Helfen, M. Bernacki, and P.-O. Bouchard, "On the choice of boundary conditions for micromechanical simulations based on 3D imaging," International Journal of Solids and Structures, vol. 112, pp. 83–96, may 2017.

[35] C. Maurer, Rensheng Qi, and V. Raghavan, "A linear time algorithm for computing exact Euclidean distance transforms of binary images in arbitrary dimensions," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, pp. 265–270, feb 2003.

[36] J. A. Sethian and A. Vladimirsky, "Fast methods for the Eikonal and related Hamilton- Jacobi equations on unstructured meshes.," Proceedings of the National Academy of Sciences of the United States of America, vol. 97, pp. 5699–703, may 2000.

[37] M. Shakoor, B. Scholtes, P.-O. Bouchard, and M. Bernacki, "An efficient and parallel level set reinitialization method – Application to micromechanics and microstructural evolutions," Applied Mathematical Modelling, vol. 39, pp. 7291–7302, dec 2015.

[38] "ISO/IEC 9899:1999," tech. rep., International Organization for Standardization, 1999.

[39] "ISO/IEC 14882:2011," tech. rep., International Organization for Standardization, 2011.

[40] F. Hecht, "New development in freefem++," Journal of Numerical Mathematics, vol. 20, no. 3-4, pp. 251–265, 2012.

[41] M. S. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells, "The fenics project version 1.5," Archive of Numerical Software, vol. 3, no. 100, 2015.

[42] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey, "libMesh : a C++ library for parallel adaptive mesh refinement/coarsening simulations," Engineering with Computers, vol. 22, no. 3-4, pp. 237–254, 2006.

[43] G. Compère, E. Marchandise, and J.-F. Remacle, "Transient adaptivity applied to two-phase incompressible flows," Journal of Computational Physics, vol. 227, pp. 1923–1942, jan 2008.

[44] H. Digonnet, L. Silva, and T. Coupez, "Cimlib: A Fully Parallel Application For Numerical Simulations Based On Components Assembly," in AIP Conference Proceedings, vol. 908, pp. 269–274, AIP, 2007.

[45] "OpenMP Application Program Interface Version 4.5," tech. rep., OpenMP Architecture Review Board, 2015.

[46] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, and F. Rossi, GNU Scientific Library Reference Manual - Third Edition. Network Theory Ltd., 2009.

[47] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, "{PETS}c {W}eb page," 2016.

[48] T. A. Davis, "Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method," ACM Transactions on Mathematical Software, vol. 30, pp. 196–199, jun 2004.

[49] A. Nishida, "Experience in Developing an Open Source Scalable Software Infrastructure in Japan," in Computational Science and Its Applications – ICCSA 2010 (D. Taniar, O. Gervasi, B. Murgante, E. Pardede, and B. Apduhan, eds.), vol. 6017 of Lecture Notes in Computer Science, pp. 448–462, Berlin, Heidelberg: Springer, 2010.

[50] J. W. Demmel, J. R. Gilbert, and X. S. Li, "An asynchronous parallel supernodal algorithm for sparse gaussian elimination," SIAM J. Matrix Analysis and Applications, vol. 20, no. 4, pp. 915–952, 1999.

[51] M. Naumov, M. Arsaev, P. Castonguay, J. Cohen, J. Demouth, J. Eaton, S. Layton, N. Markovskiy, I. Reguly, N. Sakharnykh, V. Sellappan, and R. Strzodka, "AmgX: A Library for GPU Accelerated Algebraic Multigrid and Preconditioned Iterative Methods," SIAM Journal on Scientific Computing, vol. 37, pp. S602–S626, jan 2015.

[52] K. Rupp, P. Tillet, F. Rudolf, J. Weinbub, A. Morhammer, T. Grasser, A. Jüngel, and S. Selberherr, "ViennaCL—Linear Algebra Library for Multi- and Many-Core Architectures," SIAM Journal on Scientific Computing, vol. 38, pp. S412–S439, jan 2016.

[53] U. Ayachit, The ParaView Guide: A Parallel Visualization Application. Kitware, 2015.

[54] J. Schindelin, I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, J.-Y. Tinevez, D. J. White, V. Hartenstein, K. Eliceiri, P. Tomancak, and A. Cardona, "Fiji: an open-source platform for biological-image analysis," Nature Methods, vol. 9, pp. 676–682, jun 2012.

[55] E. Erturk, T. C. Corke, and C. Gökçöl, "Numerical solutions of 2-D steady incompressible driven cavity flow at high Reynolds numbers," International Journal for Numerical Methods in Fluids, vol. 48, pp. 747–774, jul 2005.