A parallel matrix-free conservative solution interpolation on unstructured tetrahedral meshes

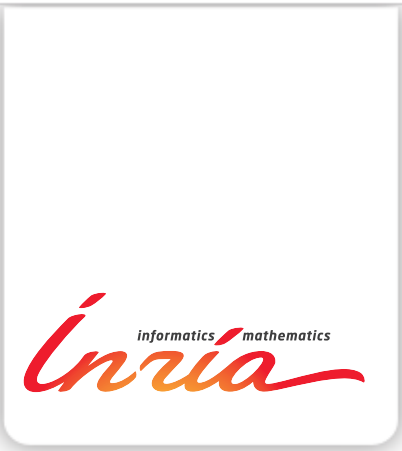
Frédéric Alauzet

A parallel matrix-free conservative solution interpolation on unstructured tetrahedral meshes

Frédéric Alauzet

RESEARCH REPORT
N° 8785
Octobre 2015
Project-Team Gamma3

ISSN 0249-6399 ISRN INRIA/RR--8785--FR+ENG



A parallel matrix-free conservative solution interpolation on unstructured tetrahedral meshes

Frédéric Alauzet*

Project-Team Gamma3

Research Report n° 8785 — Octobre 2015 — 32 pages

Abstract: This document presents an interpolation operator on unstructured tetrahedral meshes that satisfies the properties of mass conservation, $\mathbb{P}_1$-exactness (order 2) and maximum principle. Interpolation operators are important for many applications in scientific computing. For instance, in the context of anisotropic mesh adaptation for time-dependent problems, the interpolation stage becomes crucial as the error due to solution transfer accumulates throughout the simulation. This error can eventually spoil the overall solution accuracy. When dealing with conservation laws in CFD, solution accuracy requires enforcement of mass preservation throughout the computation, in particular in long time scale computations. In the proposed approach, the conservation property is achieved by local mesh intersection and quadrature formulae. Derivatives reconstruction is used to obtain a second order method. Algorithmically, our goal is to design a method which is robust and efficient. The robustness is mandatory to obtain a reliable method on real-life applications and to apply the operator to highly anisotropic meshes. The efficiency is achieved by designing a matrix-free operator which is highly parallel. A multi-thread parallelization is given in this work. Several numerical examples are presented to illustrate the efficiency of the proposed approach.

Key-words: Solution interpolation, matrix-free conservative interpolation, parallel interpolation, unstructured mesh, mesh adaptation, conservation laws

* INRIA, Équipe-projet Gamma3, Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex, France. email: frederic.alauzet@inria.fr

RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

A parallel matrix-free conservative solution interpolation on unstructured tetrahedral meshes

Résumé : Ce document présente un opérateur d'interpolation sur des maillages tétraédriques non-structurés qui satisfait les propriétés de conservation de la masse, $\mathbb{P}_1$-exactitude (ordre 2) et principe du maximum.

Mots-clés : Interpolation de solution, interpolation conservative sans matrice, interpolation parallèle, maillage non-structuré, adaptation de maillage, adaptation de maillage, loi de conservation

Contents

1 Introduction

Solution interpolation or solution transfer is an important stage for several applications in scientific computing. For instance, this stage is important for coupled problems, e.g. multi-physics simulations or fluid-structure interaction (FSI) problems, where specific meshes are considered for each sub-problem, see [1, 2]. The preservation of the conservation property during the interpolation stage is also crucial for the accuracy in long time scale simulations, see [3, 4]. It is an essential component of Arbitrary Lagrangian-Eulerian (ALE) methods as well. An accurate remapping must satisfy several properties such as conservation, high order accuracy, bound preserving, etc. This is also a key point in the context of time-accurate mesh adaptation. Indeed, it links the mesh generation and the numerical flow solver allowing the simulation to be restarted from the previous state. More precisely, after generating a new adapted mesh, called *current mesh*, the aim is to recover the solution field defined on the previous mesh, called *background mesh*, on this new mesh to pursue the computation. This recurrent stage in adaptive simulations is crucial for time-accurate unsteady problems as errors introduced by the interpolation procedure accumulate throughout the computations. The negative impact of such errors on solution accuracy was pointed out in [5] where standard linear interpolation is applied. In [6], it has been demonstrated - for the 2D case - the importance of a conservative interpolation operator on the accuracy of anisotropic time-accurate mesh adaptation for conservation laws.

A conservative interpolation based on a Galerkin projection has been proposed in [1, 7, 8]. The Galerkin projection requires the assembling and the resolution of a linear system. This method can as well suffer from oscillations. To bound the projected quantities specific treatments have to be done [7]. A global supermesh construction is used in [7, 8] but it limits considerably the efficiency of the method and the maximal size of the problem in 3D due

to memory usage. A local approach has been proposed in [6] which improves the algorithm efficiency. It is based on a local mesh intersection procedure and local quadrature. This local approach has been followed in [9] coupled with local Galerkin projection. In [6], the maximum principle is strictly enforced using L^2-optimal gradient correction, the method is thus free from oscillations.

In this paper, we consider the 3D solution interpolation for anisotropic adapted tetrahedral meshes where the background and the current meshes are distinct, in the sense that the number of entities and the connectivities can be completely different. Flows are modeled by the conservative compressible Euler equations and resolved by a second order finite volume scheme. Therefore, to obtain a consistent mesh adaptation loop, the proposed interpolation scheme must satisfy the following properties:

- mass conservation
- $\mathbb{P}_1$ exactness implying an order 2 for the method
- maximum principle.

Moreover, this method has to be algorithmically very robust as we deal with highly stretched elements and it has to be very efficient to be applied to real-life applications. The word efficient signifies that it requires low memory storage and that the additional required CPU time over that for standard linear interpolation is acceptable. In consequence, we propose a *matrix-free* approach based on local mesh intersections and appropriate local reconstructions.

The mass conservation property of the interpolation operator is achieved by local mesh intersection, *i.e.,* intersections are performed at the element level. The use of mesh intersection for conservative interpolation seems natural for unconnected meshes and has already been alluded in [10] or applied in [11] for order 1 reconstruction. The locality is inherent for efficiency and robustness. Once again for efficiency purposes, the proposed intersection algorithm is especially designed for simplicial meshes. The idea is to compute the intersection between two simplexes, mesh this intersection, and use a quadrature formulae to exactly compute the transferred mass. Moreover, the designed algorithm is highly scalable in parallel due to its locality.

The high-order accuracy is obtained by a solution gradient reconstruction from the discrete data and the use of Taylor formulae. This high-order interpolation can lead to loss of monotonicity. The maximum principle is then enforced by correcting the interpolated solution, thus the interpolated solution is free from any oscillations. Notice that much care has been taken while designing the localization algorithm as it is also critical for efficiency.

The proposed $\mathbb{P}_1$-conservative interpolation operator is suitable for solutions defined at elements or vertices.

This paper is the extension of [6] to the three-dimensional case. As compared to [6], several novelties are presented in this work. They are required to ensure the robustness [1], efficiency and accuracy of the method when dealing with 3D highly anisotropic meshes on complex geometries. These novelties concern:

- A specific tetra-tetra intersection procedure where floating point arithmetic is treated with a particular care, as it must be extremely robust and able to deal with highly anisotropic tetrahedra (anisotropic ratio up to 10^5)
- A new method to mesh the tetra-tetra intersection ensuring extra numerical accuracy
- Solutions to deal with the case of non-matching boundaries between meshes (this is always the case for complex geometries), this is crucial to not introduce artifacts in the solution during the interpolation stage and preserve accuracy
- The parallelization of the interpolation stage for efficiency
- A convergence analysis pointing out the loss of convergence using $\mathbb{P}^1$ interpolation which is recovered using the $\mathbb{P}^1$-conservative interpolation, thus advocating the use of the latter one.

[1] Ensuring the robustness of the local mesh intersection and the meshing of these intersections is considerably more difficult in 3D than in 2D. Indeed, volume positivity is harder to satisfy than area positivity when dealing with degenerated cases.

As regards the differences and similarities between this work and [9]. In [9], they use local approach and an advancing front method for the intersection identification similar to the ones proposed in [6]. The first difference lies in the computation and the meshing of the intersection between two elements. In [9], they follow the Eberly clipping algorithm [12] (?which is not described in 3D while being the core of the algorithm?) while we propose specific algorithm to achieve greater numerical accuracy and to have a relative mass variation of the order of the round-off ($\approx 10^{-14}$). This algorithm is still accurate when dealing with highly anisotropic elements (ratio up to 10^5). The second difference is the projection operator. A local Galerkin projection is used in [9] while this work considers quadrature formulae and high-order solution reconstruction. As last, we observe that the proposed algorithm achieve better efficiency in serial, *i.e.,* one order of magnitude faster, and it achieves good scaling in parallel.

The paper is outlined as follows. Section 2 introduces the main definitions and Section 3 recalls the standard linear interpolation operator. Then in Section 4, the proposed $\mathbb{P}_1$-conservative interpolation operator is described. First, the mesh intersection algorithm is presented and at a second stage, $\mathbb{P}_1$-conservative reconstruction is discussed. Finally, the accuracy of the proposed approach is emphasized on analytical examples in Section 5, and parallel efficiency is discussed in Section 6. The approach is successfully applied to adaptive numerical simulations in Section 7. Some concluding remarks close the paper.

2 Definitions and notations

This section provides notations, definitions and conventions used in this paper. Let us consider a bounded domain $\Omega \subset \mathbb{R}^3$ and denote by $\partial\Omega$ its boundary. Domain Ω is discretized by a *tetrahedral* mesh $\mathcal{H} = \bigcup K_i$. A tetrahedron K_i is defined by the list of its vertices which are locally numbered in a convenient way. This list, enriched with some conventions, provides the complete definition of the related element, including the definition of its faces, edges and neighbors, together with an orientation. In particular, the oriented local numbering of tetrahedron's vertices enables us to compute its volume while giving a sense to its sign, and to evaluate directional normals for each face.

The local numbering of vertices, edges, faces and neighboring tetrahedra is pre-defined in such a way that some properties are implicitly induced. In the case of a tetrahedron with vertices $[\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3]$, the vertices are ordered such that a positive *signed volume* is obtained while using the scalar triple product Formula (see Figure 1):

$$V_K = |K| = \frac{1}{6}\, \mathbf{e}_0 \cdot (\mathbf{e}_1 \times \mathbf{e}_2)\,. \tag{1}$$

Faces topology is defined as $F_0 = [\mathbf{p}_3, \mathbf{p}_2, \mathbf{p}_1]$, $F_1 = [\mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_0]$, $F_2 = [\mathbf{p}_1, \mathbf{p}_0, \mathbf{p}_3]$, $F_3 = [\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2]$ (Figure 1, left). This numeration is such that the index of the face is the index of the viewing vertex, *i.e.,* the opposite vertex. And, the four faces have the same inward orientation, in other words, the face normal $\mathbf{n}_{F_i} = \mathbf{e}_{i,1} \times \mathbf{e}_{i,2}$ is inward, where $\mathbf{e}_{i,1}$ is the edge composed by the first and the second face vertices and $\mathbf{e}_{i,2}$ is the edge composed by the first and the third face vertices. To improve the face normal computation accuracy, we choose the two edges forming an angle the closest as possible to 90°, *i.e.,* the minimal dot product. The surface area of the face is simply given by:

$$A_{F_i} = |F_i| = \frac{1}{2}\, \|\mathbf{n}_{F_i}\| = \frac{1}{2}\, \|\mathbf{e}_{i,1} \times \mathbf{e}_{i,2}\|\,.$$

Regarding the neighboring tetrahedra, they follow the face convention. We denote by K_i the neighbor viewing vertex $\mathbf{p}_i$ through face F_i. For the edges topology, the following choice has been made: $\mathbf{e}_0 = [\mathbf{p}_0, \mathbf{p}_1]$, $\mathbf{e}_1 = [\mathbf{p}_0, \mathbf{p}_2]$, $\mathbf{e}_2 = [\mathbf{p}_0, \mathbf{p}_3]$, $\mathbf{e}_3 = [\mathbf{p}_1, \mathbf{p}_2]$, $\mathbf{e}_4 = [\mathbf{p}_1, \mathbf{p}_3]$, $\mathbf{e}_5 = [\mathbf{p}_2, \mathbf{p}_3]$.

With the above notations, we now give some definitions used in the sequel. Let $\mathbf{p}$ be a point, we denote by K^i the virtual tetrahedron where vertex $\mathbf{p}_i$ is substituted by $\mathbf{p}$. The

signed volumes V_{K^i}, for $i = 0 \dots 3$, are called the **barycentrics** of $\mathbf{p}$. The four associated **barycentric coordinates** are given by:

$$\beta_i = \frac{V_{K^i}}{V_K} \quad \text{for} \quad i = 0 \dots 3 \,.$$

The sign of the four barycentrics defines explicitly 15 regions of space where point $\mathbf{p}$ can be located with respect to element K. The possible combinations are given in Figure 1 (right).

Now, we recall some definitions relative to the interpolation schemes. Let u be a solution defined on a mesh $\mathcal{H}^1$ of a domain Ω. The **mass** of the solution over the mesh is simply $m = \int_{\mathcal{H}^1} u$. We deduce the notion of mass on an element K given by $m_K = \int_K u$. An interpolation scheme is said to be **conservative** if it preserves the mass when transferring the solution field u from a mesh $\mathcal{H}^1$ to another $\mathcal{H}^2$. Formally speaking, if we denote by Πu the interpolated field on $\mathcal{H}^2$, then such scheme verifies

$$\int_{\mathcal{H}^1} u = \int_{\mathcal{H}^2} \Pi u \,.$$

A scheme is said to be $\mathbb{P}_k$**-exact** if it is exact for polynomial solutions of degree lower than or equal to k. A $\mathbb{P}_k$**-conservative** interpolation scheme is a scheme satisfying both properties. Finally, an interpolation scheme verifies locally the **maximum principle** on a subset $\mathcal{Q} \subset \mathcal{H}^1$ if

$$\text{for } \mathbf{p} \in \mathcal{H}^2, \quad \min_{\mathbf{q} \in \mathcal{Q}} u(\mathbf{q}) \leq \Pi u(\mathbf{p}) \leq \max_{\mathbf{q} \in \mathcal{Q}} u(\mathbf{q}) \,.$$

It verifies globally that property if $\mathcal{Q} = \mathcal{H}^1$.

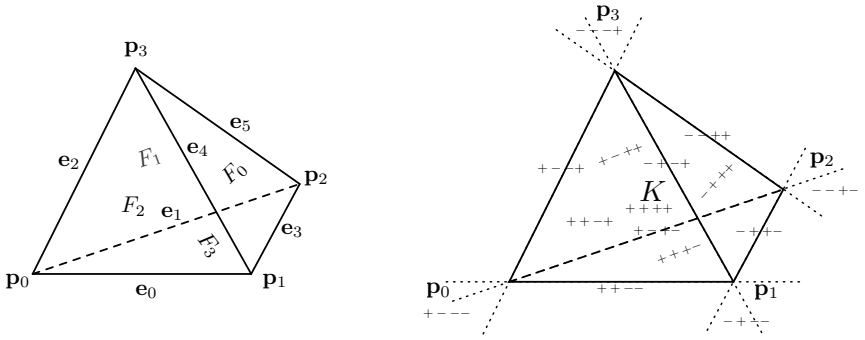


Figure 1: *Left, numeration convention for a tetrahedron K. Right, the* 15 *regions defined by the signs of the four barycentric coordinates of a point* $\mathbf{p}$ *with respect to element K.*

3 Linear interpolation operator

Solution interpolation is a twofold process. First, vertices of the new mesh are located in the background mesh. Second, an interpolation scheme is applied for each vertex.

3.1 Localization algorithm

The localization problem or research of a point location consists in identifying the element of a simplicial mesh containing a given point. Here, we consider the simplified problem where the background and the new meshes are discretizations of the same domain Ω. This problem has to be dealt with great care in the case of simplicial meshes to handle difficult configurations. Indeed, background and current meshes can be non-convex and can contain holes. It is also possible that the overlapping of the current mesh does not coincide with the background mesh since their boundary discretization can differ. Consequently, some vertices of the current mesh can be outside of the background mesh and conversely. Moreover, efficient

localization algorithms have to be implemented to avoid a quadratic complexity obtained with a naive algorithm.

The localization can be solved efficiently by traversing the background mesh using its topology, *i.e.,* the neighboring elements of each element, thanks to a barycentric coordinates-based [13, 14]. The considered algorithm is presented in details for the 2D case in [6] with a discussion on how to death with pathological configurations Its extension to 3D is straightforward.

3.2 Classical linear interpolation

Once the localization has been performed, we can apply an interpolation algorithm. The easiest interpolation scheme is the classical $\mathbb{P}_1$ interpolation:

$$\Pi_1 u(\mathbf{p}) = \sum_{i=0}^{3} \beta_i(\mathbf{p})\, u(\mathbf{p}_i)\,.$$

where $\mathbf{p}$ is a vertex of the new mesh that has been localized in tetrahedron $K = [\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3]$ of the background mesh. β_i are the barycentric coordinates of $\mathbf{p}$ w.r.t. K. This scheme is $\mathbb{P}_1$-exact and it is thus order 2. This scheme is monotone and satisfies the maximum principle. However, this scheme does not conserve the mass.

4 Matrix-free $\mathbb{P}_1$-exact conservative interpolation

In this section, a $\mathbb{P}_1$-exact conservative interpolation scheme is presented. The provided solution is considered to be piecewise (continuous or discontinuous) linear by element. The idea of the conservative interpolation is to compute the mass of each element of the new mesh $\mathcal{H}^{\text{new}}$ knowing the mass of each element of the background mesh $\mathcal{H}^{\text{back}}$. For efficiency purposes, a local mesh intersection algorithm is utilized coupled with a matrix-free interpolation. Then, in the case of vertex-centered solution, the solution is transferred accurately and conservatively from elements to vertices using the mass of the elements of its ball. This process is summarized in Algorithm 1. The mesh intersection procedure corresponding to steps 3*a* and 3*b* is exposed in Section 4.1 and the conservative reconstruction, steps 3*c*, 4 and 5, is described in Section 4.2.

Algorithm 1 Conservative Interpolation Process

Piecewise linear (continuous or discontinuous) representation of the solution on $\mathcal{H}^{\text{back}}$

1. Localize all vertices of $\mathcal{H}^{\text{new}}$ in $\mathcal{H}^{\text{back}}$
2. $\forall K^{\text{back}} \in \mathcal{H}^{\text{back}}$, compute solution mass $m_{K^{\text{back}}}$ and gradient $\nabla_{K^{\text{back}}}$
3. $\forall K^{\text{new}} \in \mathcal{H}^{\text{new}}$, recover solution mass $m_{K^{\text{new}}}$ and gradient $\nabla_{K^{\text{new}}}$:
   (a) compute the intersection of K^{new} with all $K_i^{\text{back}} \in \mathcal{H}^{\text{back}}$ it overlaps
   (b) mesh the intersection polyhedron of each pair $(K^{\text{new}}, K_i^{\text{back}})$
   (c) compute $m_{K^{\text{new}}}$ and $\nabla_{K^{\text{new}}}$ using Gauss quadrature formulae

   $\Longrightarrow$ get a piecewise linear discontinuous representation of the mass on $\mathcal{H}^{\text{new}}$
4. Correct the gradient to enforce the maximum principle
5. Set the solution values to vertices by an averaging procedure

4.1 Mesh intersection algorithm

The mesh intersection algorithm consists in intersecting each tetrahedron of the current mesh with all the background mesh tetrahedra that it overlaps and in meshing the intersec-

tion region. The 3D mesh intersection is a lot harder than in 2D, thus the method of [6] cannot be applied. Specific attention to numerical accuracy is required to deal accurately with degenerated cases (occurring frequently for highly anisotropic meshes). To this end, the tetrahedron-tetrahedron intersection topology consistency is checked to detect numerical accuracy issues. If an inconsistency is detected a perturbation method is applied. Moreover, a dedicated meshing method of the intersection is proposed where extra accuracy is obtained for the orientation predicate.

In the following, we first describe our generic intersection algorithm between any pair of tetrahedra and how we discretize the intersection polyhedron (Section 4.1.1). Secondly, the algorithm to locate all background tetrahedra that are overlapped by the current element is presented (Section 4.1.2).

4.1.1 Tetrahedron-tetrahedron intersection

The tetrahedron-tetrahedron intersection procedure computes the intersection of two tetrahedra and meshes the intersection region if it is not empty. Notice that if the intersection exists, the intersection region of two tetrahedra is always a convex polygon given by the convex hull of the intersection points. This is a three steps procedure:

1. check all the degenerated intersection cases
2. evaluate the 48 possible edge-face intersections
3. steps 1. and 2. result in a cloud of points which is triangulated resulting in a mesh of the intersection polyhedron.

In the following, these three steps are described. But, first we give a few words about the robustness of the proposed algorithm as in 3D numerical accuracy and floating point arithmetic is of utmost importance.

Accuracy and robustness of the algorithm Accuracy and robustness of algorithms when implemented with floating point arithmetic is a major topic of study in computational geometry [15, 16]. However, this work follows the choice of the authors of [13] where extended accuracy, interval analysis, arithmetic filters and sign or orientation predicates are discussed.

In our context, the most important is *consistency* meaning that the algorithm always deliver the same answer on the same configuration and the topology of the final result is correct. The consistency is crucial to avoid cycling and to have a unique answer in the localization process. It is also essential to obtain the same intersection point for a given pair edge-edge or edge-face as such intersection generally occurs many times, *i.e.,* for many tetrahedron-tetrahedron intersections, in the presented algorithm. Checking the topology of the intersection is an efficient way to detect and to correct inconsistency due to floating point arithmetic. If the algorithm is consistent (and the initial data base correct) then the orientation predicate is reliable. The design of a consistent algorithm is based on handling properly geometric degeneracy and verifying the correctness of the topology of the intersection.

First, an adequate local ε is chosen to carefully handle degenerated cases without loss of accuracy. It needs to be consistent for all intersections with the current tetrahedron. Therefore, it is only based on the current tetrahedron size.

Second, the topology of the tetrahedron-tetrahedron intersection is stored and checked throughout and at the end of the process. Topology is powerful because it is exact as it only involves boolean operation. Figure 2 presents three degenerated intersections where floating point arithmetic is involved. The two left intersections are not valid topologically while the intersection on the right is valid. For all edges, we store all the entities (vertex, edge, face) they intersect. For instance, if an edge of K^{new} intersects an edge of K^{back}, then we store that these edges intersect each other and that they also intersect the faces they share. Moreover, for each intersection point, the entities (vertex, edge, face) of the current and of the background tetrahedra on which it lies on are stored. This set of data provides a complete view of the intersection polyhedron. To verify the topology validity, the number of intersection points on each entity is also checked. In particular, an edge and a face cannot

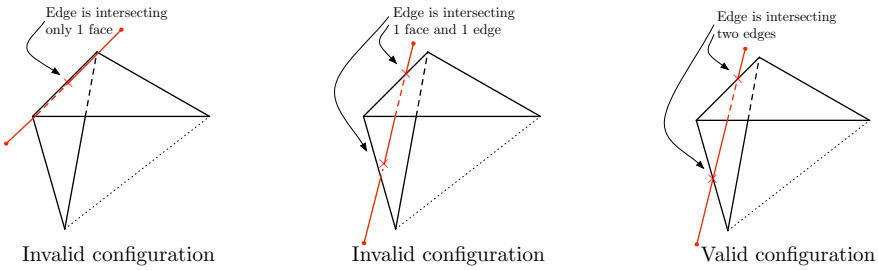


Figure 2: *Three degenerated intersections are presented. The two intersections on the left are inconsistent topologically while the one on the right is valid. Indeed, for the left one, red edge* $\mathbf{e}$ *has an intersection with the top face but it has no other intersection with tetrahedron* K*. In such case, the red edge must intersect* K *twice or have a degenerated intersection with an edge of* K*. For the middle one, red edge* $\mathbf{e}$ *has an intersection with one edge of the side face and an intersection inside the side face which is not valid. Such an edge must either be coplanar with the side face, resulting in the configuration presented on the right, or have a non-degenerated intersection with the top face.*

have more than 2 and 6 intersection points, respectively. And, the intersection polyhedron contains at most 12 intersection points [17]. Cases with 12 intersection points are obtained, for instance, for the stella octangula polyhedron or by taking two opposite tetrahedra from Cundy and Rollett's construction of the tetrahedron 5-compound, [18].

During the tetrahedron-tetrahedron intersection process, the topology consistency is checked to correct pathological cases due to floating point arithmetic and to update missed degenerated cases. After completing the tetrahedron-tetrahedron intersection, the topology consistency is analyzed. If an inconsistency occurs, then (*i*) the intersection is reset, (*ii*) a tiny perturbation of the background vertices position is performed and (*iii*) the intersection is resumed. The perturbation helps to get rid of unresolved degenerated cases that create the inconsistency. As the perturbation is of the order of ε, it doesn't affect the accuracy.

Remark: *The proposed choice to cope with floating point arithmetic is very efficient as it is able to accurately handle the intersection of highly anisotropic tetrahedra with aspect ratio of 1:100 000. and to recover the tetrahedron volume at the order of the round-off* ($\approx 10^{-14}$).

Dealing with degenerated cases We first introduce the definition of the **signed distance**, also called **power**, of point $\mathbf{p}$ with respect to face $F_i = [\mathbf{p}_{i,0}, \mathbf{p}_{i,1}, \mathbf{p}_{i,2}]$:

$$\mathcal{P}(\mathbf{p}, F_i) = \mathbf{p}_{i,k}\mathbf{p} \cdot \frac{\mathbf{n}_{F_i}}{\|\mathbf{n}_{F_i}\|} \text{ where } k \text{ is either } 0, 1 \text{ or } 2\,. \tag{2}$$

Notice that the barycentrics and the powers are linked by the relation: $V_{K^i} = \frac{1}{6}\|\mathbf{n}_{F_i}\|\,\mathcal{P}(\mathbf{p}, F_i)$. The distance of point $\mathbf{p}$ with respect to edge $\mathbf{e}_i = \mathbf{p}_{i,0}\mathbf{p}_{i,1}$ is:

$$\mathcal{P}(\mathbf{p}, e_i) = \frac{\|\mathbf{p}_{i,0}\mathbf{p}_{i,1} \times \mathbf{p}_{i,0}\mathbf{p}\|}{\|\mathbf{p}_{i,0}\mathbf{p}_{i,1}\|} = \frac{\|\mathbf{p}_{i,0}\mathbf{p} \times \mathbf{p}_{i,1}\mathbf{p}\|}{\|\mathbf{p}_{i,0}\mathbf{p}_{i,1}\|}\,.$$

For the given pair of tetrahedra, the 32 vertices powers with respect to the faces, $\mathcal{P}(\mathbf{p}_j, F_i)$, are computed according to Equation (2). Then, all possible vertex degenerated cases are checked according to $\mathcal{P}(\mathbf{p}_j, F_i)$ values:

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

If such a case happen, the vertex is snapped on the appropriate entity and added to the intersection points list. The intersection topology table is updated accordingly.

Afterward, edge-edge intersections - which are degenerated cases - are tested. Let $\mathbf{p}_0\mathbf{p}_1$ and $\mathbf{q}_0\mathbf{q}_1$ the two considered lines. A necessary condition is that the two lines are coplanar. If it is the case, the intersection point $\mathbf{x}$ is evaluated following Hill's approach [19]:

$$\begin{aligned} \mathbf{x} &= \mathbf{p}_0 + \frac{(\mathbf{p}_0\mathbf{q}_0 \times \mathbf{q}_0\mathbf{q}_1)\cdot(\mathbf{p}_0\mathbf{p}_1 \times \mathbf{q}_0\mathbf{q}_1)}{\|\mathbf{p}_0\mathbf{p}_1 \times \mathbf{q}_0\mathbf{q}_1\|^2}\,\mathbf{p}_0\mathbf{p}_1 = \mathbf{p}_0 + s\,\mathbf{p}_0\mathbf{p}_1\,, \\ \text{or}\quad \mathbf{x} &= \mathbf{q}_0 + \frac{(\mathbf{p}_0\mathbf{p}_1 \times \mathbf{q}_0\mathbf{p}_0)\cdot(\mathbf{p}_0\mathbf{p}_1 \times \mathbf{q}_0\mathbf{q}_1)}{\|\mathbf{p}_0\mathbf{p}_1 \times \mathbf{q}_0\mathbf{q}_1\|^2}\,\mathbf{q}_0\mathbf{q}_1 = \mathbf{q}_0 + t\,\mathbf{q}_0\mathbf{q}_1\,. \end{aligned}$$

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

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

If there is intersection, point $\mathbf{x}$ is added to the intersection points list and the intersection topology table is updated.

Edge-face intersection Dealing with degenerated cases first simplifies the following edge-face intersection procedure. Indeed, the computation of all the above geometric degeneracy treats in particular all the possible coplanar edge-face intersections as depicted in Figure 3.

Now, only non-coplanar edge-face intersection remains where the intersection point lies inside the face. The algorithm checks the 48 possible edge-face intersections. Let us denote the considered edge by $\mathbf{e} = [\mathbf{p}_0, \mathbf{p}_1]$ and face by $F = [\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2]$. As it is a non-coplanar case, the edge's vertex powers with respect to the face are not zero: $\mathcal{P}(\mathbf{p}_0, F) \neq 0$ and $\mathcal{P}(\mathbf{p}_1, F) \neq 0$. A necessary condition for the edge-face intersection is:

$$\mathcal{P}(\mathbf{p}_0, F)\,\mathcal{P}(\mathbf{p}_1, F) \;<\; 0\,.$$

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

$$\mathbf{x} = \mathbf{p}_0 + \frac{\mathcal{P}(\mathbf{p}_0, F)}{\mathcal{P}(\mathbf{p}_0, F) - \mathcal{P}(\mathbf{p}_1, F)}\mathbf{p}_0\mathbf{p}_1\,.$$

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

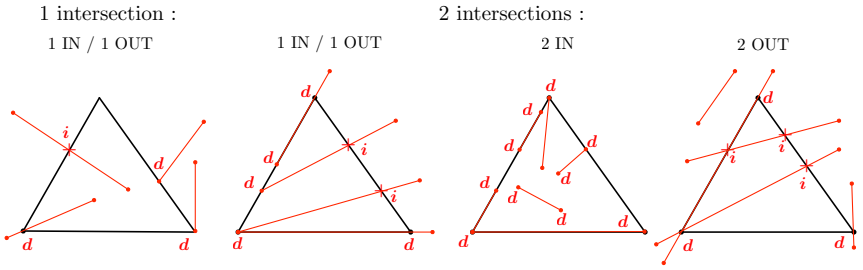


Figure 3: *All possible intersection configurations for coplanar pair edge-face (all kinds of intersection are gathered by edge-face configurations). Four configurations can occur. One intersection with an edge's vertex in the face and the other vertex out. Two intersections with zero (resp. one or two) edge's vertex in the face and two (resp. one or zero) edge's vertices outside of the face. In the pictures, an intersection marked by "i" or "d" represents a pure or a degenerated intersection, respectively.*

Meshing the intersection polyhedron The intersection procedure results in a cloud of points formed by the intersection points list. This list is analyzed to mesh the intersection polyhedron.

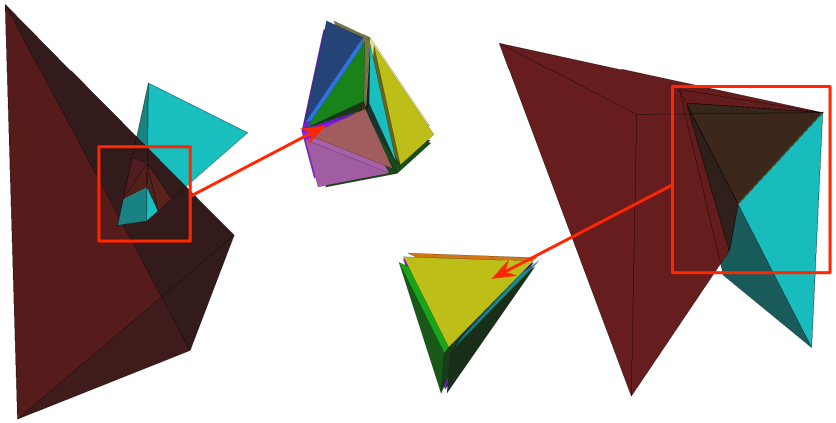


Figure 4: *Two tetrahedron-tetrahedron intersection configurations where the cyan tet intersects the maroon tet. Left, a non-degenerated intersection and, right, a degenerated case where both tets share a common vertex. For each configuration, the mesh of the intersection is depicted. Meshes are composed of* 12 *(left) and* 8 *(right) tets, respectively.*

If the intersection process returns strictly less than 4 intersection points, then we face a degenerated tetrahedron-tetrahedron intersection where there is no geometric intersection or the geometric intersection is either a vertex, a line or a face. In this context, the algorithm has to make the distinction between the case where one tetrahedron is included in the other one, meaning that the intersection is this tetrahedron, and the case where the intersection is empty. Tetrahedron $K_{\mathbf{p}} = [\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3]$ is included inside tetrahedron $K_{\mathbf{q}} = [\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3]$ if and only if $\forall i = 0, .., 3$ and $\forall j = 0, .., 3$, $\mathcal{P}(\mathbf{p}_i, F_j^{\mathbf{q}}) \geq 0$, where $F_j^{\mathbf{q}}$ are the faces of $K_{\mathbf{q}}$. As all the powers are evaluated at the beginning of the intersection process, the included tetrahedron case can be immediately checked. Therefore, if the number of intersection points is strictly less than 4, it implies that no intersection occurs.

Otherwise, if more than 4 intersection points are returned, the convex hull of this cloud of points forms a convex polyhedron representing the region of intersection of the tetrahedra pair.

At this point, many strategies can be employed to mesh the intersection volume. The more intuitive idea is to create the Delaunay triangulation of the cloud of points - similar to the one used in [6] - as it returns the convex hull of the set of points. However, a basic algorithm lacks of robustness due to the presence of many coplanar points inside faces and the frequent nearly-degenerated configurations when two highly anisotropic tetrahedra are involved. In such context, orientation predicate may fail leading to a wrong configuration. To solve this issue, one can consider the use of a very advanced Delaunay triangulation algorithm such as the one proposed by [20]. But, this seems incongruous to call such an evolved algorithm at each intersection to triangulate a dozen of points and it may face the same issues.

Therefore, we propose a clever strategy using the information gathered during the intersection process. Indeed, as an intersection topology table has been created, we know which intersection point belong to which tetrahedron's face. A first step consists in meshing all the faces of the convex polyhedron. Each of these faces corresponds to a face of one of the two tetrahedra. Thus, for each created triangle on these faces the orientation predicate is based on the orientation of the normal which must be identical to the orientation of the corresponding tetrahedron's face normal. This gives extra accuracy because this predicate is based on a cross product whereas a volume predicate is based on a triple product. Once all the tetrahedra's faces have been meshed, an oriented surface mesh of the intersecting polyhedron is obtained. In order to get a simplicial volume mesh of the polyhedron, its gravity center is added and tetrahedra are created by joining the barycenter to each triangle. In other words, the polyhedron surface mesh is star-shaped with respect to the barycenter.

Two examples of tetrahedron-tetrahedron are shown in Figure 4.

4.1.2 Overlapped tetrahedra detection

The conservative interpolation method consists in computing for each tetrahedron K^{new} of the current mesh $\mathcal{H}^{\text{new}}$ its intersection with all tetrahedra K_j^{back} of the background mesh $\mathcal{H}^{\text{back}}$ that it overlaps. We present how this list of background elements is determined.

First of all, all vertices of the new mesh $\mathcal{H}^{\text{new}}$ are localized in the background mesh $\mathcal{H}^{\text{back}}$ using the algorithm presented in Section 3.1. Then, for each tetrahedron K^{new} of $\mathcal{H}^{\text{new}}$, the initial list of background tetrahedra that are overlapped is given by the elements containing the vertices of K^{new}. For degenerated cases where a new vertex lies on a background face or on a background edge or on a background vertex, we add to the initial list the background tetrahedra that are sharing the face or are in the edge's shell or are in the vertex's ball, respectively.

Then, the intersections between K^{new} and the tetrahedra of the initial list are computed. New tetrahedra are added to the list during the intersection procedure as follows:

- if face F_j of K^{back} is intersected by K^{new} then neighbor K_j^{back} of K^{back} sharing face F_j is added to the list
- if K^{new} is intersected by an edge of K^{back} then background tetrahedra of the edge's shell are added to the list
- if a vertex of K^{back} lies inside K^{new} then background tetrahedra of the vertex's ball are added to the list.

With this simple procedure, all overlapped elements are automatically detected while computing intersections. Overlapped elements are detected without any additional cost as powers and intersections have already been computed in the tetrahedron-tetrahedron intersection process.

4.2 $\mathbb{P}_1$-conservative reconstruction

In this section, we describe the $\mathbb{P}_1$-conservative solution reconstruction process. It mainly follows [6], but choices made to cope with the case of non-matching boundaries between meshes (this is always the case for complex geometries) are detailed. We consider a bounded domain Ω of $\mathbb{R}^3$ and two tetrahedral meshes of this domain $\mathcal{H}^{\text{back}} = \bigcup K_i^{\text{back}}$ and $\mathcal{H}^{\text{new}} = \bigcup K_i^{\text{new}}$. For sake of simplicity, we first make the assumption that the discrete boundaries of both meshes are the same, *i.e.,* both meshes are discretization of the same polyhedral domain Ω_h: $|\mathcal{H}^{\text{back}}| = |\mathcal{H}^{\text{new}}|$ where $|\mathcal{H}| = \int_{\Omega_h} \mathrm{d}\mathbf{x}$. The case of non-matching discrete boundaries is addressed in Section 4.2.4. For each mesh, a dual partition of the domain is defined by associating with each vertex a control volume or cell (which is defined by some rules): $\mathcal{H}^{\text{back}} = \bigcup C_i^{\text{back}}$ and $\mathcal{H}^{\text{new}} = \bigcup C_i^{\text{new}}$. A $\mathbb{P}_1$ discrete solution field u is given on the background mesh $\mathcal{H}^{\text{back}}$.

Now, we have to define a projection operator Π_1^c from $\mathcal{H}^{\text{back}}$ to $\mathcal{H}^{\text{new}}$ with the following properties:

- Π_1^c is conservative: $\displaystyle\int_{\mathcal{H}^{\text{back}}} u = \int_{\mathcal{H}^{\text{new}}} \Pi_1^c u$
- Π_1^c is $\mathbb{P}_1$-exact: if u is affine then $\Pi_1^c u = u$.

The projection operator is presented for solutions defined at elements and solutions defined at vertices.

4.2.1 Solution defined at elements

In this case, the solution is piecewise linear by elements and can be discontinuous. We have for each background tetrahedron K^{back}:

- mass $m_{K^{\text{back}}} = \displaystyle\int_{K^{\text{back}}} u = |K^{\text{back}}|\, u(\mathbf{g})$, where $\mathbf{g}$ is the barycenter of K^{back}
- constant gradient $\nabla u_{K^{\text{back}}}$.

For each tetrahedron K^{new} of the current mesh, we compute the intersection with all tetrahedra of the background mesh $\{K_j^{\text{back}}\}_j$ it overlaps as described in the previous section. Each pair of tetrahedra K^{new} and K_j^{back} provides a simplicial mesh of their intersection region denoted $\mathcal{T}_j = K^{\text{new}} \cap K_j^{\text{back}}$. The integrals $\displaystyle\int_{\mathcal{T}_j} u$ and $\displaystyle\int_{\mathcal{T}_j} \nabla u$ are computed exactly using Gauss quadrature formulae. Consequently, we obtain for each tetrahedron of the current mesh a mass and a gradient given by:

$$m_{K^{\text{new}}} = \int_{K^{\text{new}}} \Pi_1^c u = \sum_j \int_{\mathcal{T}_j} u \qquad \text{and} \qquad (\nabla \Pi_1^c u)_{|K^{\text{new}}} = \frac{\sum_j \int_{\mathcal{T}_j} \nabla u}{|K^{\text{new}}|}\,.$$

This reconstruction is conservative and $\mathbb{P}_1$-exact. It gives a $\mathbb{P}_1$ by element discontinuous solution. A specific treatment of the reconstruction is carried out to verify the maximum principle.

4.2.2 Verifying the maximum principle

Let K be a tetrahedron of the new mesh. In the following, for the sake of clarity, we denote by u_K the $\mathbb{P}_1$-conservative interpolated solution $\Pi_1^c u$ on K. The value at the barycenter and the gradient of the interpolated solution on K are given by:

$$u_K(\mathbf{g}_K) = \frac{1}{|K|}\int_K \Pi_1^c u \quad \text{and} \quad \nabla u_K = (\nabla \Pi_1^c u)_{|K}\,.$$

Consequently, for each vertex $\mathbf{p}_i$ of the new mesh, a value of the solution is obtained using Taylor expansion for each element K of its ball:

$$u_K(\mathbf{p}_i) = u_K(\mathbf{g}_K) + \nabla u_K \cdot \mathbf{g}_K\mathbf{p}_i\,. \tag{3}$$

A correction is applied to the linear representation of the solution on each element in order to verify the maximum principle. The interpolated solution is thus free from any oscillations. To this end, let $\mathcal{K}$ be the set of elements of the background mesh that K overlaps and let $\mathcal{Q}$ be the set of vertices of $\mathcal{K}$:

$$\mathcal{K} = \{K_j^{\text{back}} \,|\, K \cap K_j^{\text{back}} \neq \emptyset\} \qquad \text{and} \qquad \mathcal{Q} = \{\mathbf{q}_j \,|\, \mathbf{q}_j \in K^{\text{back}} \text{ such that } K^{\text{back}} \in \mathcal{K}\}\,.$$

Then, for each vertex $\mathbf{p}_i$ of each element K of the new mesh, the nodal value $u_K(\mathbf{p}_i)$ verify the maximum principle if:

$$u_{\min} = \min_{\mathbf{q}\in\mathcal{Q}} u(\mathbf{q}) \leq u_K(\mathbf{p}_i) \leq \max_{\mathbf{q}\in\mathcal{Q}} u(\mathbf{q}) = u_{\max}\,.$$

Notice that $u_K(\mathbf{g}_K)$ always satisfies the maximum principle. If a vertex does not verify the maximum principle on an element K then the gradient value of this element is corrected. The proposed approach results from a minimization problem[2]. We first reorder the indices such that $u_K(\mathbf{p}_0) \leq u_K(\mathbf{p}_1) \leq u_K(\mathbf{p}_2) \leq u_K(\mathbf{p}_3)$. Then, we set:

$$\begin{aligned}
u_K^M(\mathbf{p}_3) &= \min\big(u_K(\mathbf{p}_3)\,,\, u_{\max}\big) & \tilde{u}_K(\mathbf{p}_0) &= \max\big(u_K^M(\mathbf{p}_0)\,,\, u_{\min}\big)\\
u_K^M(\mathbf{p}_2) &= \min\big(u_K(\mathbf{p}_2) + \frac{1}{3}\,(u_K(\mathbf{p}_3) - u_K^M(\mathbf{p}_3))\,,\, u_{\max}\big) & \tilde{u}_K(\mathbf{p}_1) &= \max\big(u_K^M(\mathbf{p}_1) + \frac{1}{3}\,(u_K^M(\mathbf{p}_0) - \tilde{u}_K(\mathbf{p}_0))\,,\, u_{\min}\big)\\
u_K^M(\mathbf{p}_1) &= \min\big(u_K(\mathbf{p}_1) + \frac{1}{2}\sum_{i=2}^{3}(u_K(\mathbf{p}_i) - u_K^M(\mathbf{p}_i))\,,\, u_{\max}\big) & \tilde{u}_K(\mathbf{p}_2) &= \max\big(u_K^M(\mathbf{p}_2) + \frac{1}{2}\sum_{i=0}^{1}(u_K^M(\mathbf{p}_i) - \tilde{u}_K(\mathbf{p}_i))\,,\, u_{\min}\big)\\
u_K^M(\mathbf{p}_0) &= u_K(\mathbf{p}_0) + \sum_{i=1}^{3}(u_K(\mathbf{p}_i) - u_K^M(\mathbf{p}_i)) & \tilde{u}_K(\mathbf{p}_3) &= u_K^M(\mathbf{p}_3) + \sum_{i=0}^{2}(u_K^M(\mathbf{p}_i) - \tilde{u}_K(\mathbf{p}_i))\,.
\end{aligned}$$

These new nodal values $\tilde{u}_K(\mathbf{p}_i)$ define the corrected linear representation of the solution on K. For any points $\mathbf{x}$ included in K, its solution value is then given by: $\tilde{u}_K(\mathbf{x}) =$

[2] A method based on the notion of slope limiter widely used in numerical schemes has also been implemented but it turns out to be less accurate than the current method on all validation cases.

$\sum_{i=0}^{3} \beta_i(\mathbf{x}) \tilde{u}_K(\mathbf{p}_i)$, where $\beta_i(\mathbf{x})$ are the barycentric coordinates of $\mathbf{x}$ with respect to K. The final interpolated solution verifies all required properties:

PROPOSITION 1: *The reconstruction $\tilde{u}_K$ satisfies the maximum principle, is linear preserving and is conservative. Moreover, we have:*

$$u_K(\mathbf{p}_0) \leq u_K(\mathbf{p}_1) \leq u_K(\mathbf{p}_2) \leq u_K(\mathbf{p}_3) \quad \Longrightarrow \quad \tilde{u}_K(\mathbf{p}_0) \leq \tilde{u}_K(\mathbf{p}_1) \leq \tilde{u}_K(\mathbf{p}_2) \leq \tilde{u}_K(\mathbf{p}_3)$$

and if we have $u_{\min} \leq u_K(\mathbf{p}_i) \leq u_{\max}$ for $i = 0..3$ then $\tilde{u}_K(\mathbf{p}_i) = u_K(\mathbf{p}_i)$ for $i = 0..3$.

Notice that this reconstruction comes from a minimization problem. Indeed, we have

PROPOSITION 2: *Suppose that $u_K(\mathbf{p}_0) \leq u_K(\mathbf{p}_1) \leq u_K(\mathbf{p}_2) \leq u_K(\mathbf{p}_3)$ and that $u_{\max} < u_K(\mathbf{p}_3)$. Then, we have*

$$\sum_{i=0}^{3} |u_K(\mathbf{p}_i) - u_K^M(\mathbf{p}_i)|^2 \leq \sum_{i=0}^{3} |u_K(\mathbf{p}_i) - v_K(\mathbf{p}_i)|^2$$

for all the linear reconstructions v_K satisfying $v_K(\mathbf{p}_3) = u_{\max}$, $v_K(\mathbf{p}_i) \leq u_{\max}$ for $i = 0,..,2$ and $\int_K v_K = \int_K u_K$.

The proofs of these propositions are identical to the 2D proofs given in [6].

4.2.3 Solution defined at vertices

When the solution is given at vertices of the background mesh, *i.e.,* nodal values are provided, it implicitly defines a piecewise linear continuous representation of the solution at the elements. Therefore, the $\mathbb{P}_1$-conservative interpolation defined in the previous section can be applied. However, a piecewise linear solution by elements, which is generally discontinuous, is obtained on the new mesh. Therefore, one more stage is required to retrieve a solution at vertices of the new mesh which consists in transferring this solution from elements to vertices while preserving the properties of the interpolation scheme. The solution is simply re-distributed to each vertex $\mathbf{p}$ of the new mesh by averaging:

$$\tilde{u}(\mathbf{p}) = \frac{\sum_{K_i^{\text{new}} \ni \mathbf{p}} |K_i^{\text{new}}| \; \tilde{u}_{K_i^{\text{new}}}(\mathbf{p})}{\sum_{K_i^{\text{new}} \ni \mathbf{p}} |K_i|},$$

where $\tilde{u}$ is the interpolated solution on the new mesh. Notice that after re-distribution to vertices the interpolated solution still satisfies the maximum principle, is linear preserving and is conservative.

REMARK: *As the mass of the solution is linked to the topology of the mesh, the $\mathbb{P}_1$-conservative interpolation operator Π_1^c depends on the mesh topology on which it is applied. In consequence, it cannot be applied to interpolate solution at any points of a given domain.*

4.2.4 Non-matching discrete boundaries

Let Ω be a bounded domain of $\mathbb{R}^3$ and $\mathcal{H}^{\text{back}}$ and $\mathcal{H}^{\text{new}}$ two meshes of Ω. We consider the case where the discrete boundaries of $\mathcal{H}^{\text{back}}$ and $\mathcal{H}^{\text{new}}$ do not match. In other words, $\mathcal{H}^{\text{back}}$ and $\mathcal{H}^{\text{new}}$ are meshes of two different polyhedral domains Ω_h^{back} and Ω_h^{new} the boundary of which differs: $\Gamma_h^{\text{back}} \neq \Gamma_h^{\text{new}}$. Therefore, the volume of each mesh differs: $|\mathcal{H}^{\text{back}}| \neq |\mathcal{H}^{\text{new}}|$. For instance, when dealing with complex geometries, the volume of the domain may change due to the change in the geometric approximation of the surface between both meshes.

For the conservative interpolation, the non-matching discrete boundaries are handled differently depending on the solution behavior and the geometric configuration. This is crucial to not introduce artifacts in the solution during the interpolation stage and preserve accuracy. Let's take the simple example of a sphere in a constant flow field delimited by a box domain. If the geometric approximation of the sphere is increased (the surface mesh size on the sphere is divided by a factor 2), then the volume of the sphere increases (because it is convex) leading to a decrease of the domain volume. In that case, we obviously want

to preserve the constant flow field when the solution is transferred from one mesh to the other, hence the conservation property must be violated. Satisfying the $\mathbb{P}_1$-exactness is more important than being conservative. Three specific solution cases may occur: the solution is constant, linear or something else. And, for the geometry configuration, we may have either a background tetrahedron or a new tetrahedron which is not entirely overlapped.

When a tetrahedron of the background mesh is not completely overlapped, we choose to do nothing specific thus some of the global mass is not recovered due to the volume change. The algorithm is no longer conservative but it still preserves the $\mathbb{P}_1$-exactness and the maximum principle properties.

When a tetrahedron of the new mesh is not completely overlapped, we have :

$$|K^{\text{new}}| \neq \sum_j |K^{\text{new}} \cap K_j^{\text{back}}| = \sum_j |\mathcal{T}_j| = |\mathcal{T}_\cap| \,.$$

This case is treated based on the solution behavior as follows.

If the solution is constant, the operator must recover a constant solution to satisfy the $\mathbb{P}_1$-exactness and the maximum principle properties. Indeed, if a constant uniform field is given as the initial set, we have to return an constant uniform solution set to not introduce any artifacts in the solution. The conservation principle must be violated. Formally speaking, we have:

$$m_\cap = c\,|\mathcal{T}_\cap| \quad \text{and} \quad (\nabla\,\Pi_1^c u)_{|K^{\text{new}}} = 0\,,$$

and, the value at barycenter is obtained by dividing the mass by the intersection volume and not the new tetrahedron volume:

$$u_K(\mathbf{g}_K) = \frac{m_\cap}{|\mathcal{T}_\cap|} = c \quad \Rightarrow \quad m_{K^{\text{new}}} = c\,|K^{\text{new}}|\,.$$

If the solution is linear, the operator must recover a linear solution to satisfy the $\mathbb{P}_1$-exactness, once more to not introduce any artifacts in the solution. The above method cannot be applied because the recovered mass is not correct to compute the barycenter value. However, the exact gradient value can be recovered:

$$(\nabla\,\Pi_1^c u)_{|K^{\text{new}}} = \frac{\sum_j \int_{\mathcal{T}_j} c}{|\mathcal{T}_\cap|} = c\,.$$

Therefore, to retrieve the linear solution, we pick one of the new tetrahedron vertex that is inside the domain and a linear extrapolation is applied to obtain the other vertices values. In such case, the maximum principle and the conservation properties are no more verified.

Finally, if the solution is not in the two previous cases, we choose to preserve the maximum principle and to not be conservative. Values and gradients at barycenter are computed as:

$$u_K(\mathbf{g}_K) = \frac{\sum_j \int_{\mathcal{T}_j} u}{|\mathcal{T}_\cap|} \quad \text{and} \quad (\nabla\,\Pi_1^c u)_{|K^{\text{new}}} = \frac{\sum_j \int_{\mathcal{T}_j} \nabla u}{|\mathcal{T}_\cap|}\,.$$

and values at vertices are computed with Relation (3).

5 Accuracy and convergence study on analytical functions

In this section, the behavior of the $\mathbb{P}_1$-conservative interpolation is analyzed on four analytical functions defined on a cubic domain $[-0.5, 0.5]^3$. These functions are representative of several physical phenomena encountered in computational fluid dynamics (CFD). The function's solutions are considered at vertices. The $\mathbb{P}_1$-conservative interpolation is compared to the linear interpolation, in particular, the mass conservation and the convergence order of the schemes are studied.

To perform this analysis, two meshes $\mathcal{H}_1^1$ and $\mathcal{H}_1^2$, composed respectively of 906 and 918 vertices, are considered. These meshes are completely different and unconnected. In order

to study the convergence order of each interpolation method, each of these meshes spans a series of embedded meshes denoted $(\mathcal{H}_i^1)_{i=1...5}$ and $(\mathcal{H}_i^2)_{i=1...5}$. Mesh $\mathcal{H}_{i+1}^j$ is deduced from $\mathcal{H}_i^j$ by splitting each tetrahedra into eight tetrahedra in a Lagrangian fashion, *i.e.,* in an *isoparametric* way. These series of meshes are summarized in Table 1.

For each case, the analytical function is applied on $\mathcal{H}_i^1$ providing a solution field u_i^1. This solution field is transferred from $\mathcal{H}_i^1$ to $\mathcal{H}_i^2$, we get Πu_i^2. This solution transfer is called transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2$. The error is computed by comparing the interpolated solution Πu_i^2 with the analytical function applied on $\mathcal{H}_i^2$, *i.e.,* u_i^2, in L^1-norm:

$$\varepsilon_i = \int_{\mathcal{H}_i^2} |u_i^2 - \Pi u_i^2| \,.$$

The series of errors enable a convergence study. We also analyze the error when the solution field is re-interpolated back to $\mathcal{H}_i^1$. More precisely, the function is applied on $\mathcal{H}_i^1$ giving u_i^1, then it is interpolated on $\mathcal{H}_i^2$ giving Πu_i^2 and finally Πu_i^2 is interpolated from $\mathcal{H}_i^2$ to $\mathcal{H}_i^1$ and we obtain Πu_i^1. The error ϵ_i is obtained by computing the gap in L^1-norm between u_i^1 and Πu_i^1 on $\mathcal{H}_i^1$. This double solution transfer is called transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2 \to \mathcal{H}_i^1$. For completeness, we analyze the behavior of each interpolation operator for many interpolation steps. This will point out how error due to solution interpolation accumulates. This is crucial for anisotropic mesh adaptation application where a large number of interpolation steps are done. To this end, transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2 \to \mathcal{H}_i^1$ is performed five times leading to a total of 10 interpolation steps. It is denoted $5 \times \mathcal{H}_i^1 \to \mathcal{H}_i^2 \to \mathcal{H}_i^1$.

The four studied analytical functions are represented in Figure 5. For each analytical function, a figure is given providing:

- left, relative mass variation $\frac{|m_{\mathcal{H}_i^1} - m_{\mathcal{H}_i^2}|}{m_{\mathcal{H}_i^1}}$ for solution transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2$
- middle, error ε_i for solution transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2$
- and right, error ϵ_i for solution transfers $\mathcal{H}_i^1 \to \mathcal{H}_i^2 \to \mathcal{H}_i^1$ and $5 \times \mathcal{H}_i^1 \to \mathcal{H}_i^2 \to \mathcal{H}_i^1$ for a total of 2 ($\#I = 2$) and 10 ($\#I = 10$) interpolations, respectively.

The linear interpolation scheme is represented by the red (and green) lines and the $\mathbb{P}_1$-conservative interpolation is represented by the blue (and purple) lines.

A Gaussian function The first analytical function is a gaussian given by:

$$u_1(x, y, z) = \exp(-30\,(x^2 + y^2 + z^2))\,.$$

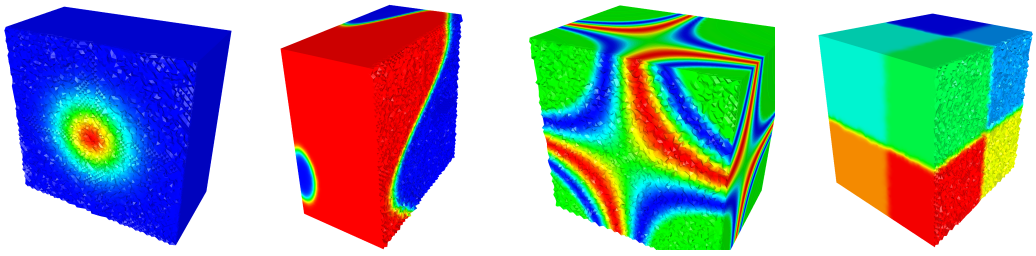


Figure 5: *Representation of the four analytical functions. From left to right, gaussian function u_1, continuous sinusoidal shock function u_2, multi-scales smooth function u_3 and discontinuous function u_4.*

Step	1	2	3	4	5
# vertices $\mathcal{H}_i^1$	906	6 287	42 656	322 656	2 330 670
# vertices $\mathcal{H}_i^2$	918	6 432	46 695	360 543	2 739 492

Table 1: *Mesh sizes of the series of embedded meshes $(\mathcal{H}_i^1)_{i=1...5}$ and $(\mathcal{H}_i^2)_{i=1...5}$.*

This smooth function is representative of the vortices encountered in CFD, Figure 6.

The relative mass variation with the classical linear interpolation reduces up to 40,000 vertices and then stabilizes around 0.005% of variation. Conversely, the relative mass variation with the $\mathbb{P}_1$-conservative interpolation is of the order of the round-off ($\approx 5.\ 10^{-14}$) for all interpolation steps.

As regards the accuracy and the convergence order, both interpolation scheme are converging at order 2 for solution transfers $\mathcal{H}_i^1 \to \mathcal{H}_i^2$, $\mathcal{H}_i^1 \to \mathcal{H}_i^2 \to \mathcal{H}_i^1$ and $5 \times \mathcal{H}_i^1 \to \mathcal{H}_i^2 \to \mathcal{H}_i^1$. This fits to the theory. We notice that the $\mathbb{P}_1$-conservative interpolation is more accurate than the linear one in both cases. The difference in accuracy is almost a factor 1.7 and 2.4 for solution transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2$ and $\mathcal{H}_i^1 \to \mathcal{H}_i^2 \to \mathcal{H}_i^1$, respectively.

However, analyzing the $5\times\mathcal{H}_i^1 \to \mathcal{H}_i^2 \to \mathcal{H}_i^1$ case, we notice that the error level stays almost the same with the $\mathbb{P}_1$-conservative interpolation when the number of interpolations increases while the error increases with the linear interpolation. It means that error accumulates in the classical case and this adverse effect is drastically reduced with the new interpolation scheme. As a consequence, the error with the linear interpolation is 7.7 times higher than the $\mathbb{P}_1$-conservative interpolation one.

A continuous sinusoidal shock This analytical function represents a continuous model of a shock which can be assimilated to the numerical capture of a shock with a dissipative flow solver, *i.e.,* the solver captures the shock on several mesh elements. This smooth function is given by:

$$u_2(x,y,z) = \tanh\big(20\,\big(x + 0.3\,\sin(-10\,y) - 0.3\,\sin(-5(z-0.1))\,\big)\big)\,.$$

It contains two quasi-constant regions that are separated by sinusoidal interfaces in which strong gradient variation occurs continuously.

As previously, the relative mass variation with the $\mathbb{P}_1$-conservative interpolation is of the order of the round-off ($\approx 10^{-14}$) for all interpolation steps. For the linear interpolation the relative mass variation decreases with the mesh size. It varies from 13% for the first meshes to 0.02% for the last meshes with two million vertices.

The $\mathbb{P}_1$-conservative interpolation achieves an order 2 of convergence for all solution transfers whereas the linear interpolation has a convergence order less than 2. The convergence order asymptotically reaches 1.75 for all cases. As regards the accuracy, the $\mathbb{P}_1$-conservative interpolation is more accurate than the linear one in all cases and the difference increases while meshes are refined. For solution transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2$, the error is 2 times smaller with the $\mathbb{P}_1$-conservative interpolation on the finest meshes. This gap increases for multiple solution transfers and rises to 5.8 when ten interpolations are done.

A multi-scales smooth function This function presents smooth sinusoidal variations but at different scales. There are two order of magnitudes between small and large scales variations. This function reads:

$$u_3(x,y,z) = \begin{cases} 0.01\,\sin(200\,x\,y\,z) & \text{if} \quad x\,y\,z \le \dfrac{-\pi}{200} \\ \sin(200\,x\,y\,z) & \text{if} \quad \dfrac{-\pi}{200} < x\,y\,z \le \dfrac{2\,\pi}{200} \\ 0.01\,\sin(200\,x\,y\,z) & \text{if} \quad \dfrac{2\,\pi}{200} < x\,y\,z \end{cases} .$$

The relative mass variation with the $\mathbb{P}_1$-conservative interpolation is of the order of the round-off ($\approx 10^{-14}$) for all interpolation steps. For the linear interpolation, the mass variation is large from 5% for step 1 to 0.01% for step 5 for only one solution transfer. However, the mass variation seems to converge toward zero while the mesh size goes toward zero.

For this smooth case, the $\mathbb{P}_1$-conservative approach reaches an order 2 of convergence for all solution transfers. With the linear interpolation an order 2 is reached for one transfer and it is lower than 2 for multiple transfers. Concerning the accuracy, the $\mathbb{P}_1$-conservative interpolation achieves better accuracy than the standard linear interpolation. The difference in accuracy increases with the number of solution transfers. For the finest meshes (step 4 and 5), the error is 6 times smaller for ten solution transfers while it was only a factor 1.7 for one solution transfer. Again, error accumulates with linear interpolation while the effect is diminished with the conservative one.

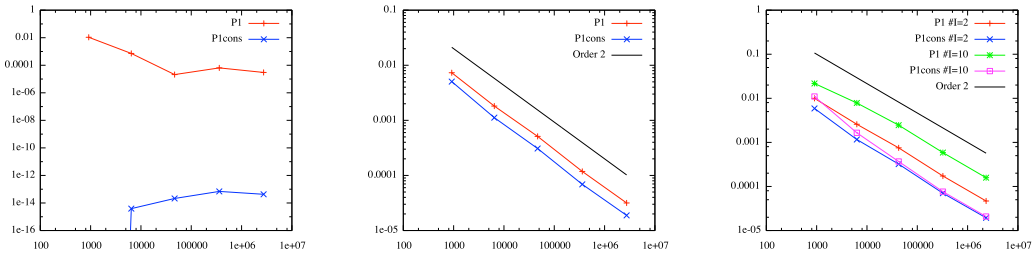


Figure 6: *Gaussian analytical function u_1. Left, mass variation for the transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2$. Middle, error ε_i for the transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2$. Right, error ϵ_i for the transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2 \to \mathcal{H}_i^1$ (#I=2) and $5 \times \mathcal{H}_i^1 \to \mathcal{H}_i^2 \to \mathcal{H}_i^1$ (#I=10).*

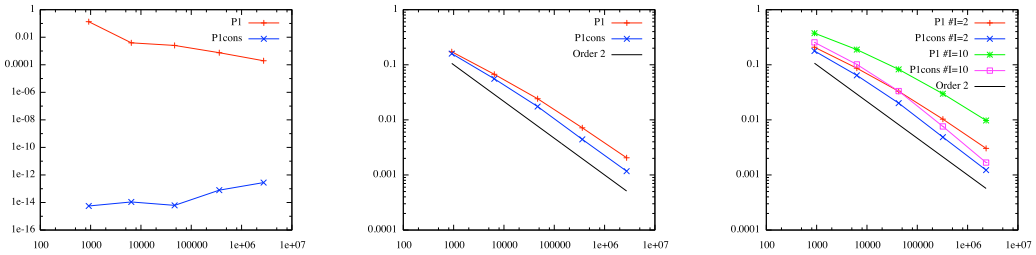


Figure 7: *Continuous sinusoidal shock analytical function u_2. Left, mass variation for the transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2$. Middle, error ε_i for the transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2$. Right, error ϵ_i for the transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2 \to \mathcal{H}_i^1$ and $5 \times \mathcal{H}_i^1 \to \mathcal{H}_i^2 \to \mathcal{H}_i^1$.*

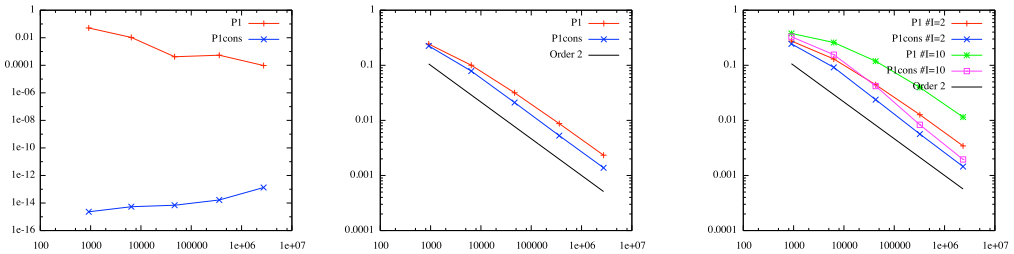


Figure 8: *Multi-scales smooth analytical function u_3. Left, mass variation for the transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2$. Middle, error ε_i for the transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2$. Right, error ϵ_i for the transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2 \to \mathcal{H}_i^1$ (#I=2) and $5 \times \mathcal{H}_i^1 \to \mathcal{H}_i^2 \to \mathcal{H}_i^1$ (#I=10).*

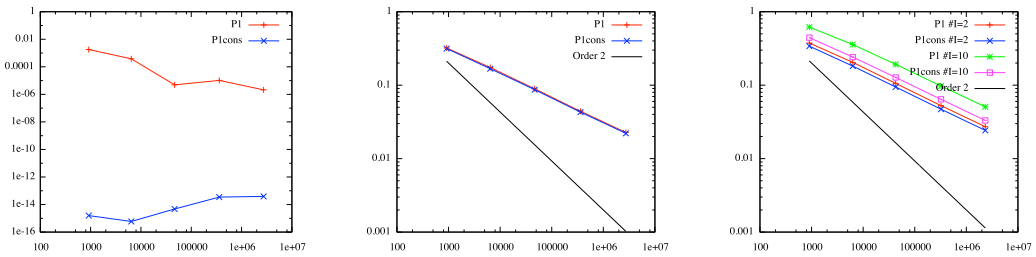


Figure 9: *Discontinuous analytical function u_4. Left, mass variation for the transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2$. Middle, error ε_i for the transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2$. Right, error ϵ_i for the transfer $\mathcal{H}_i^1 \to \mathcal{H}_i^2 \to \mathcal{H}_i^1$ (#I=2) and $5 \times \mathcal{H}_i^1 \to \mathcal{H}_i^2 \to \mathcal{H}_i^1$ (#I=10).*

Nbr. cores	Serial	1 HT	2 HT	4 HT	8 HT	10 HT	20 HT
Timings (sec.)	1,435	1,071	562	301	158	126	72
Speed-up	1.0	1.3	2.6	4.8	9.1	11.3	19.93

Table 2: Timings and speed-up on computer 1 up to 20 cores with hyper-threading.

Nbr. cores	Serial	1 HT	2 HT	4 HT	10 HT	20 HT	40 HT
Timings (sec.)	2,087	1,625	1,011	413	193	115	63
Speed-up	1.0	1.3	2.0	5.0	10.8	18.1	33.1

Table 3: Timings and speed-up on computer 2 up to 40 cores with hyper-threading.

A discontinuous function The last analytical function is discontinuous and represents eight steps:

$$u_4(x,y,z) = \begin{cases} 1 & \text{if} \quad x \geq 0 \quad \text{and} \quad y \geq 0 \quad \text{and} \quad z \geq 0 \\ 2 & \text{if} \quad x \geq 0 \quad \text{and} \quad y < 0 \quad \text{and} \quad z \geq 0 \\ 3 & \text{if} \quad x < 0 \quad \text{and} \quad y \geq 0 \quad \text{and} \quad z \geq 0 \\ 4 & \text{if} \quad x < 0 \quad \text{and} \quad y < 0 \quad \text{and} \quad z \geq 0 \\ 5 & \text{if} \quad x \geq 0 \quad \text{and} \quad y \geq 0 \quad \text{and} \quad z < 0 \\ 6 & \text{if} \quad x \geq 0 \quad \text{and} \quad y < 0 \quad \text{and} \quad z < 0 \\ 7 & \text{if} \quad x < 0 \quad \text{and} \quad y \geq 0 \quad \text{and} \quad z < 0 \\ 8 & \text{if} \quad x < 0 \quad \text{and} \quad y < 0 \quad \text{and} \quad z < 0 \end{cases}.$$

The solution is constant in eight cubic regions and is discontinuous at the interface of each region.

Again, the relative mass variation with the $\mathbb{P}_1$-conservative interpolation is of the order of the round-off for all the interpolation steps. For the linear interpolation, the relative mass variation varies from 0.2% for step 1 to 0.0002% for step 5 for one solution transfer. It seems to converge toward zero while the size approaches zero.

Even if the mass is preserved, the same accuracy is obtained for both approaches while transferring the solution from one mesh to another one, Nevertheless, for the multiple solution transfers, the $\mathbb{P}_1$-conservative interpolation performs better than the classical linear approach as it accumulates less error.

For this purely discontinuous case, the two approaches reach only an order 1 of convergence for all solution transfers.

Conclusions For all those analytical cases, while preserving the mass, the $\mathbb{P}_1$-conservative interpolation obviously achieves better accuracy than the classical linear interpolation and for some cases it converges at a faster rate. We also notice that it gathers less error when multiple solution transfers are performed. This points out the superiority of the conservative interpolation over the standard one.

6 Parallelization of the conservative interpolation algorithm

The CPU time overhead is minor in 2D but a major issue in 3D. In 2D, the conservative interpolation is three times slower than the polynomial linear interpolation [6]. In the context of 2D anisotropic mesh adaptation, the CPU times devoted to the interpolation stage is very small (less than one percent) relative to the total adaptive CPU time. Therefore, the time overhead induced by the conservative interpolation as compared to the polynomial one can be considered as negligible.

In 3D, the conservative interpolation is 50 times slower on average than the polynomial linear interpolation, meaning that it has non-negligible cost in the adaptive process, but numerical results shown in Section 7.2 will point out that it is well worth it. Let us explain why the overhead is so important. For all the performed test cases, there are 16 computed intersections on average for each tetrahedron. For each intersection, 32 barycentrics are computed

to solve the 48 edge-face intersections. Therefore, a mean of 512 barycentrics evaluations is needed for each tetrahedron which is 128 times the cost of the linear interpolation. Then, we have to generate a 3D mesh of these intersections which is composed of 8 tetrahedra on average.

Fortunately, the procedure is easily parallelized and scales very well because most of the CPU time is spent in step 3 of Algorithm 1. For this step, the intersection of each tetrahedron of the new mesh with the background mesh can be done independently. There is no dependency nor communication which makes it efficient to parallelize. In this work, we follow the strategy proposed in [21] where space filling curve based renumbering is used to minimize cache misses and memory contention, and the parallelization uses the p-thread paradigm [22, 23].

Parallel performance has been analyzed on two different multi-cores computers with different processors and memory access speeds:

- Computer 1:
  - 2 chips: Xeon E5-2670 10 cores 2.5 GHz
  - both chips are connected by 2 QPI links with a speed of 16 GB/s
- Computer 2:
  - 4 chips: Xeon E7-4850 10 cores 2 GHz
  - all chips are connected to all by 1 QPI link with a speed of 16 GB/s

The selected test case is the spherical blast of Section 7.2.1 where the state (five solution fields) at a-dimensioned time 0.6945 is interpolated. The background mesh size is $12,993,399$ tetrahedra and the new mesh size is $13,037,975$ tetrahedra. For that case, 226 millions tetrahedron-tetrahedron intersections have been computed and 1.7 billions tetrahedra have been generated to mesh the intersections.

Parallel timings are compared to the CPU time in serial on the same mesh, thus strong speed-ups are analyzed. In the timings analysis, I/Os and initializations are not taken into account. For each parallel run, hyper-threading has been used by launching a number of threads equal to twice the number of cores. For instance, the 4 HT run means than only 4 cores have been used but 8 threads have been launched. Timings and speed-ups for computer 1 and 2 are summarized in Table 2 and 3, respectively.

We notice that the speed-ups are excellent on both computers and even super-linear for a low number of cores thanks to the hyper-threading which reduces memory latency. However, for computer 2, we notice that when more chips are used, the speed-up degrades. This is mainly due to slower memory access between the chips (only one link between each).

We can evaluate the algorithm performance by computing the average time spent to treat one element. On Computer 1, it takes in serial 0.1 ms of CPU time per element and on 20 cores with hyper-threading 0.0055 ms of CPU time per element.

7 Application to mesh adaptation

One of the main application is time-accurate anisotropic mesh adaptation in CFD. Mesh adaptation provides a way to control the accuracy of the numerical solution by modifying the domain discretization according to size and directional constraints. It is well known that mesh adaptation captures accurately physical phenomena in the computational domain while reducing significantly the CPU time, see [5, 24, 25, 26, 27, 28, 29].

7.1 Unsteady mesh adaptation scheme

Our goal is to solve an unsteady PDE which is set in the computational space-time domain $\mathcal{Q} = \Omega \times [0, T]$ where T is the (positive) maximal time and $\Omega \subset \mathbb{R}^3$ is the spatial domain. Let Π_h be the usual $\mathbb{P}^1$ projector. The considered problem of mesh adaptation consists in

finding the space-time mesh $\mathcal{H}$ of $\mathcal{Q}$ that minimizes the space-time linear interpolation error $u - \Pi_h u$ in L^p norm. The problem is thus stated in an *a priori* way:

Find $\mathcal{H}_{opt}$ having N_{st} space-time vertices such that $\mathbf{E}_{L^p}(\mathcal{H}_{opt}) = \min_{\mathcal{H}} \|u - \Pi_h u\|_{L^p(\Omega_h \times [0,T])}$.

In the continuous mesh framework, we rewrite this problem under the continuous form [30]:

$$\text{Find } \mathbf{M}_{L^p} = (\mathcal{M}_{L^p}(\mathbf{x},t))_{(\mathbf{x},t)\in\mathcal{Q}} \text{ such that } \mathbf{E}_{L^p}(\mathbf{M}_{L^p}) = \min_{\mathbf{M}} \|u - \pi_{\mathcal{M}} u\|_{L^p(\Omega\times[0,T])}, \quad (4)$$

under the space-time constraint:

$$\mathcal{C}_{st}(\mathbf{M}) = \int_0^T \tau(t)^{-1} \left(\int_\Omega d_{\mathcal{M}}(\mathbf{x},t)\, \mathrm{d}\mathbf{x} \right) \mathrm{d}t = N_{st}\,. \quad (5)$$

where $\tau(t)$ is the time step used at time t. To find the optimal space-time continuous mesh, Problem (4-5) is solved in two steps. First, a spatial minimization is done for a fixed t. Second, a temporal minimization is performed. The expression of the optimal space-time metric $\mathbf{M}_{L^p}$ for a prescribed time step is [31]:

$$\mathcal{M}_{L^p}(\mathbf{x},t) = N_{ST}^{\frac{2}{3}} \left(\int_0^T \tau(t)^{-\frac{2p}{2p+3}}\, \mathcal{K}(t)\mathrm{d}t \right)^{-\frac{2}{3}} \tau(t)^{\frac{2}{2p+3}} (\det |H_u(\mathbf{x},t)|)^{-\frac{1}{2p+3}} |H_u(\mathbf{x},t)|\,, \quad (6)$$

where H_u is the Hessian of sensor u.

The previous analysis provides the optimal size of the adapted meshes for each time level. Hence, this analysis requires the mesh to be adapted at each flow solver time step which is inconceivable. Now, we want to extend the previous analysis to the fixed-point mesh adaptation algorithm context [5]. The idea consists in splitting the simulation time frame $[0,\, T]$ into n_{adap} adaptation sub-intervals:

$$[0,\, T] \;=\; [0 = t_0,\, t_1] \cup \ldots \cup [t_i,\, t_{i+1}] \cup \ldots \cup [t_{n_{adap}-1},\, t_{n_{adap}}]\,, \quad (7)$$

and to keep the same adapted spatial mesh $\mathbf{M}^i$ for each time sub-interval $[t_{i-1}, t_i]$. On each sub-interval, the mesh is adapted to control the solution accuracy from t_i to t_{i+1}. Consequently, the time-dependent simulation is performed with n_{adap} different adapted meshes. This drastically reduces the number of remeshing during the simulation, hence the number of solution transfers. This can been seen as a coarse adapted discretization of the time axis, the spatial mesh being kept constant for each sub-interval when the global space-time mesh is visualized, thus providing a first answer to the adaptation of the whole space-time mesh.

Algorithm 2 Mesh Adaptation Loop for Unsteady Flows

Initial mesh and solution$(\mathcal{H}_0, \mathcal{S}_0^0)$ and set targeted space-time complexity N_{st}

 # Fixed-point loop to converge the global space-time mesh adaptation problem
For $j = 1, n_{ptfx}$

 # Adaptive loop to advance the solution in time on time frame $[0, T]$

1. For $i = 1, n_{adap}$
   (a) $\mathcal{S}_{0,i}^j$ = Interpolate conservatively next sub-interval initial sol. from $(\mathcal{H}_{i-1}^j, \mathcal{S}_{i-1}^j, \mathcal{H}_i^j)$;
   (b) $\mathcal{S}_i^j$ = Compute solution on sub-interval from pair $(\mathcal{S}_{0,i}^j, \mathcal{H}_i^j)$;
   (c) $|\mathbf{H}|_i^j$ = Compute sub-interval Hessian-metric from sol. sample $(\mathcal{H}_i^j, \{\mathcal{S}_i^j(k)\}_{k=1,nk})$;
   EndFor
2. $\mathcal{C}^j$ = Compute space-time complexity from all Hessian-metrics $(\{|\mathbf{H}|_i^j\}_{i=1,n_{adap}})$;
3. $\{\mathcal{M}_i^j\}_{i=1,n_{adap}}$ = Compute all sub-interval unsteady metrics $(\mathcal{C}^j, \{|\mathbf{H}|_i^j\}_{i=1,n_{adap}})$;
4. $\{\mathcal{H}_i^{j+1}\}_{i=1,n_{adap}}$ = Generate all sub-interval adapted meshes $(\{\mathcal{H}_i^j,\ \mathcal{M}_i^j\}_{i=1,n_{adap}})$;

EndFor

To converge the non-linear mesh adaptation problem, *i.e.,* converging the mesh-solution couple, a fixed-point mesh adaptation algorithm is used. This is also a way to predict the solution evolution and to adapt the mesh accordingly. The previous error analysis can be also carried out in this context [31]. The time-accurate fixed-point mesh adaptation algorithm is schematized in Algorithm 2. where $\mathcal{H}$, $\mathcal{S}$, $\mathbf{H}$ and $\mathcal{M}$ denote respectively meshes, solutions, Hessian-metrics and metrics.

Flow solver In all the examples, the flow is modeled by the conservative Euler equations. Assuming that the gas is perfect, inviscid and that there is no thermal diffusion, the Euler equations for mass, momentum and energy conservation read:

$$\frac{\partial W}{\partial t} + \nabla \cdot F(W) = 0\,,$$

where $W = {}^t(\rho, \rho\mathbf{u}, \rho E)$ is the conservative variables vector and vector F represents the convective operator:

$$F(W) = {}^t\left(\rho\mathbf{u}, \rho u\mathbf{u} + pe_x, \rho v\mathbf{u} + pe_y, \rho w\mathbf{u} + pe_z, \mathbf{u}(\rho E + p)\right)\,.$$

We have noted ρ the density, $\mathbf{u} = (u, v, w)$ the velocity vector, $E = T + \frac{\|\mathbf{u}\|^2}{2}$ the total energy and $p = (\gamma - 1)\rho T$ the pressure with $\gamma = 1.4$ the ratio of specific heats and T the temperature.

The Euler system is solved by means of a Finite Volume technique on unstructured meshes composed of tetrahedra. The proposed scheme is vertex-centered, achieved a second order accuracy in space and a third order accuracy in time with an explicit Runge-Kutta scheme. More details can be found in [32].

Local remesher The generation of the adapted anisotropic meshes is done using a metric-based adaptive local remeshing strategy [33] where the surface mesh is adapted conjointly with the volume mesh using local mesh modifications. One main advantage of this method is to be extremely robust. Indeed, if an invalid operation occurs, it is simply rejected. The core of the algorithm uses a unique cavity-based operator [34]. Thus, each meshing operateor is equivalent to a node insertion or reinsertion. It is the cavity initialization (definition) which defines the scope of the underlying local mesh modification. The node insertion is based on the extension of the Delaunay kernel to the metric-based anisotropic context. This strategy is very efficient to generate high-quality adapted meshes with a very high level of anisotropy $O(1:10^6)$ [33].

Remark: *For polynomial interpolation, it has been observed that interpolating physical variable $(\rho, \mathbf{u}, p)$ is more accurate than dealing with conservative variable $(\rho, \rho\mathbf{u}, \rho E)$. As ρ and p are interpolated, the positivity is preserved. But, the conservative interpolation works on $(\rho, \rho\mathbf{u}, \rho E)$ and, thus, there is no guarantee that $p = (\gamma - 1)\left(\rho E - \frac{\rho\|\mathbf{u}\|^2}{2}\right)$ remains positive after the process even if it should be. Indeed, for 3D problems involving strong physics (e.g. blast wave) and complex geometry (e.g. sharp ridges in the city) the positivity issue may occur. This is very rare and concerns a few points over million. In that case, the resulting interpolated field contains negative pressure which is not admissible for the flow solver. A correction, using clipping or averaging with neighboring vertices, is then applied to restore a positive pressure.*

7.2 Numerical simulations

The goal of the presented examples is to emphasized the benefits in terms of accuracy and convergence of using the $\mathbb{P}_1$-conservative interpolation instead of classical $\mathbb{P}_1$ interpolation for item 1 (a) in the unsteady mesh adaptation loop presented in Algorithm 2. For each simulation, identical parameters are used for the flow solver and the local remesher, and a control of the interpolation error of the density variable in L^2-norm is done.

To analyze the accuracy of each simulation and to perform a convergence analysis, we compute the L^1-norm of the space-time error with respect to the reference solution:

$$\begin{aligned} err_{ST} &= \int_0^T \int_\Omega |u_{ref}(\mathbf{x},t) - u(\mathbf{x},t)| \mathrm{d}\mathbf{x}\mathrm{d}t \\ &\approx \sum_{i_{adap}=1}^{n_{adap}^{ref}} \Delta t \left(\sum_{i_{tet}=1}^{N_{tet}^{ref}(i_{adap})} |K_{i_{tet}}| \, |u_{ref}(G_{i_{tet}}, t_{i_{adap}}) - u(G_{i_{tet}}, t_{i_{adap}})| \right) \end{aligned} \tag{8}$$

with the notations:

- n_{adap}^{ref} is the number of reference adapted meshes used for the simulation
- $\Delta t = \frac{T}{n_{adap}^{ref}}$ is the sub-intervals time length
- $N_{tet}^{ref}(i_{adap})$ is the number of tetrahedra of the $i_{i_{adap}}^{th}$ adapted mesh used to compute the reference solution for sub-interval $[t_{i_{adap}-1}, t_{i_{adap}}]$
- $|K_{i_{tet}}|$ is the volume of the $i_{i_{tet}}^{th}$ tetrahedron
- $u_{ref}(G_{i_{tet}}, t_{i_{adap}})$ and $u(G_{i_{tet}}, t_{i_{adap}})$ are the reference solution and the solution at $i_{i_{tet}}^{th}$ tetrahedron barycenter at time $t_{i_{adap}}$.

The total space-time mesh complexity, *i.e.,* total number of vertices, of a simulation ran on n_{adap} meshes is:

$$N_{ST} = \sum_{i_{adap}=1}^{n_{adap}} N_{ver}(i_{adap}) = n_{adap} \times \tilde{N}_{ver} \tag{9}$$

where $N_{ver}(i_{adap})$ is the number of vertices of the $i_{i_{adap}}^{th}$ adapted mesh and $\tilde{N}_{ver}$ is the average number of vertices per mesh.

7.2.1 Spherical blast

The first example is a spherical Riemann problem between two parallel walls simulating a blast. Initially, the gas is at rest with density $\rho_{out} = 1$ and pressure $p_{out} = 1$ everywhere except in a sphere centered at $(0, 0, 0.4)$ with radius 0.2. Inside the sphere the parameters are $\rho_{in} = 1$ and $p_{in} = 5$. For both regions, we have $\gamma = 1.4$. The initial pressure jump results in a strong outward moving shock wave, an outward contact discontinuity and an inward moving rarefaction wave. The main feature of the solution are the interactions between these waves. Another significant feature is the development of a low density region in the center of the domain. The solution remains cylindrically symmetric throughout the simulation and is computed until a-dimensioned time $T = 0.7$.

The adaptive *reference solution* has been computed using $n_{adap}^{ref} = 128$ sub-intervals, *i.e.,* the number of adapted meshes used to run the simulation, each mesh having an average size of $\tilde{N}_{ver}^{ref} = 1,712,282$ vertices. The total space-time number of vertices of the reference solution is $N_{ST}^{ref} = 219$ millions vertices. The final adapted mesh and density solution field are shown in Figure 10.

Four series of adaptive simulations have been run to make the comparison between the two interpolation methods. Two series with the $\mathbb{P}_1$ interpolation and two series with the $\mathbb{P}_1$-conservative interpolation by setting a theoretical target mesh size of $120,000$ and $240,000$ vertices on average per sub-interval mesh. To increase the space-time mesh complexity and to perform a convergence analysis, the number of sub-intervals is increased. Notice that when the simulation complexity increased, the number of interpolation stages increases as well. For each series, six simulations have been run with 4, 8, 16, 32, 64 and 128 sub-intervals. The total theoretical space-time mesh size being the number of sub-intervals n_{adap} multiplied by the average complexity N_{Avg}. The resulting mesh sizes for the simulations with $240,000$ vertices on average per sub-interval mesh are presented in Table 4 using $\mathbb{P}_1$ interpolation and in Table 5 using $\mathbb{P}_1$-conservative interpolation. We notice that for the same set of parameters, we obtain almost the same space-time mesh complexity for the $\mathbb{P}_1$ and the $\mathbb{P}_1$-conservative interpolation.

The improvement in accuracy and convergence of the conservative interpolation with respect to the classical one is clearly pointed out in Figure 14 where the space-time error - given by Equation (8) - vs. the space-time complexity - given by Equation (9) - is plotted for each simulation.

As larger size adapted meshes are generated with $\tilde{N}_{ver}^{240K}$ than $\tilde{N}_{ver}^{120K}$, it results (as expected) in more accurate solutions, therefore the blue and the pink curves are below the red and the green curves, respectively. This accounts for the discretization error that has been reduced by increasing the mesh size.

Besides, the difference between the blue (resp. red) curve and the pink (resp. green) curve points out the accumulation of the error due to the interpolation stage. For the $\mathbb{P}_1$ interpolation, we observe that the accuracy of the solution degrades more and more with the number of interpolation stages. This degradation becomes visible when 16 or more interpolation steps are done. This is due to solution transfer errors which accumulates at each interpolation and spoils the solution accuracy. As this is not the case with the $\mathbb{P}_1$-conservative interpolation even if 128 solution transfers are done, the gap in accuracy between the two approaches grows quickly with the number of adaptations (sub-intervals) and becomes quickly significant. As shown in Section 5, the error introduced by the interpolation stage is linked to the mesh size. The smaller h, the smaller the error. This is why the pink curve is shift to the right with respect to the green curve. Hence, the accumulation of interpolation error is much more dramatic with coarser meshes.

This significant difference is evident when we observe the final density solutions and adapted meshes, shown in Figures 11 and 12, obtained with each method for a theoretical target mesh size of $240,000$ vertices on average per sub-interval mesh and $n_{adap} = 128$. The solution using the $\mathbb{P}_1$-conservative interpolation is obviously more detailed with more shock waves present in the flow, less dissipated shock waves and the low density region at the domain center separated by a contact discontinuity is a lot less diffused. In comparison, we notice that both solutions for $n_{adap} = 16$ - given in Figure 13 - are more similar. Indeed, less error due to the solution transfer has been accumulated. Moreover, we observe a time shift of the shock waves positions with the $\mathbb{P}_1$ interpolation which increases with the number of sub-intervals, while no time shift occurs with $\mathbb{P}_1$-conservative interpolation. These comments are highlighted in Figure 14 by density solutions extraction along a line for the reference and these four cases solutions.

This significant difference is also emphasized by the final adapted mesh size of each simulation. For a given number of sub-intervals and average complexity, we see in Tables 4 and 5 that the average number of vertices per mesh in both cases is similar. However, the larger the number of adapted meshes, the larger the final adapted mesh size using the $\mathbb{P}_1$-conservative interpolation, see Tables 4 and 5. A similar behavior is observed for the simulations where $120,000$ vertices on average per sub-interval mesh has been prescribed. This means that information on the solution is lost during the simulation using the $\mathbb{P}_1$ interpolation.

Tables 4 and 5 also report the CPU time distribution between the interpolation stage, the flow solver stage, and the metric computation and the mesh generation stage. All stages are run in parallel using the same number of processors. Table 4 - for the $\mathbb{P}_1$ interpolation - shows that the 3%/63%/33% (Interpolation/Flow/Metric-Mesh) CPU distribution using 4 adapted meshes evolves to a 11%/31%/58% CPU distribution using 128 adapted meshes. Hence, the flow solver CPU part decreases with respect to the interpolation and meshing stages when the number of sub-intervals increases. This change in the CPU distribution is due to the lower number of flow solver time steps performed on each mesh when a larger number of meshes is used to discretize the simulation time frame[3]. Using a larger number of meshes increases the number of interpolation steps and the number of generated meshes. In this example, the average number of flow solver time steps per sub-interval is 525 for $n_{adap} = 4$ and drops to 60 for $n_{adap} = 128$. For the $\mathbb{P}_1$-conservative interpolation, see Table 5, the same behavior is observed: the 15%/41%/44% CPU distribution using 4 adapted meshes evolves to a 45%/23%/32% CPU distribution using 128 adapted meshes. As the

[3]Indeed, more meshes considered to reach the simulation final time (see Expression (7)) means less time steps performed on each mesh.

$\mathbb{P}_1$-conservative interpolation is more costly in term of CPU time, it takes a larger part of the CPU distribution. An almost balanced distribution between the three stages is obtained for $n_{adap} = 32$. Similar CPU distributions are obtained for the simulations where $120,000$ vertices on average per sub-interval mesh has been prescribed.

In conclusion, for time-accurate compressible flow simulations with a large number of adapted meshes, the conservative interpolation is mandatory even if it has an important CPU cost in the unsteady mesh adaptation loop.

7.2.2 A blast in a town

The second example is the propagation of a blast in a geometry representing a city plaza. The physics of this problem has a lot more energy than the one proposed in [5], it results in stronger and more complex physical phenomena. The main feature is related to the random character of the blast wave propagation due to a large number of waves reflexions on the geometry and the interactions between the numerous blast waves. The computational domain size is $85 \times 85 \times 70$ m^3. Initially, the gas representing the ambient air is at rest with a density $\rho_{out} = 1$ and $p_{out} = 1$. To simulate the blast, a high pressure and density region is introduced in a quarter-circle centered at $(6.5, 0)$ with a radius 0.25. In this region, the relevant parameters are $\rho_{in} = 10$, $p_{in} = 25$ and $\mathbf{u}_{in} = \mathbf{0}$. For both regions, we have $\gamma = 1.4$. The solution is computed until a-dimensioned time $T = 15$.

The adaptive *reference solution* has been computed using $n_{adap}^{ref} = 128$ adapted meshes, each mesh having an average size of $\tilde{N}_{ver}^{ref} = 3,327,382$ vertices. The total space-time number of vertices of the reference solution is $N_{ST}^{ref} = 426$ millions vertices. The final adapted mesh and density solution field are shown in Figure 16.

n_{adap}	$\mathbb{P}_1$				
	$\tilde{N}_{ver}^{240K}$	$N_{ver}^{240K}(n_{adap})$	Itp CPU	Flow CPU	Mesh CPU
4	181,073	202,807	3.48%	63.19%	33.33%
8	198,064	223,082	5.36%	69.00%	25.64%
16	214,477	245,638	7.67%	61.72%	30.61%
32	230,379	265,977	9.51%	51.73%	38.76%
64	245,714	278,025	10.44%	41.53%	48.03%
128	256,374	259,879	10.53%	30.97%	58.50%

Table 4: *Spherical blast. Statistics for the unsteady mesh adaptation algorithm for different number of sub-intervals n_{adap} using the $\mathbb{P}_1$ interpolation. From left to right, average number of vertices per sub-interval $\tilde{N}_{ver}$ and final meshes number of vertices $N_{ver}(n_{adap})$, percentage of the total CPU time for the interpolation (Itp), the flow solver (Flow) and the metric computation and mesh generation (Mesh) stages. The total space-time mesh number of vertices is $n_{adap} \times \tilde{N}_{ver}$.*

n_{adap}	$\mathbb{P}_1$-conservative				
	$\tilde{N}_{ver}^{240K}$	$N_{ver}^{240K}(n_{adap})$	Itp CPU	Flow CPU	Mesh CPU
4	182,330	206,760	14.98%	41.12%	43.90%
8	199,328	229,657	23.62%	53.70%	22.68%
16	217,308	259,689	30.44%	43.94%	25,62%
32	236,141	291,980	35.77%	35.49%	28.74%
64	256,758	327,466	41.11%	28.14%	30.75%
128	279,709	367,101	44.80%	23.05%	32.15%

Table 5: *Spherical blast. Statistics for the unsteady mesh adaptation algorithm for different number of sub-intervals n_{adap} using the $\mathbb{P}_1$-conservative interpolation. From left to right, average number of vertices per sub-interval $\tilde{N}_{ver}$ and final meshes number of vertices $N_{ver}(n_{adap})$, percentage of the total CPU time for the interpolation (Itp), the flow solver (Flow) and the metric computation and mesh generation (Mesh) stages. The total space-time mesh number of vertices is $n_{adap} \times \tilde{N}_{ver}$.*

Two series of adaptive simulations have been run to make the comparison between the two interpolation methods. A theoretical target mesh size of $250,000$ vertices on average per sub-interval mesh has been prescribed. To increase the space-time mesh complexity and to perform a convergence analysis, the number of sub-intervals is increased. Thus, the number of interpolation stages increases with the simulation complexity. For each series, six simulations have been run with 4, 8, 16, 32, 64 and 128 sub-intervals. The resulting mesh size for each simulation are presented in Tables 6 and 7. We notice that for the same set of parameters,

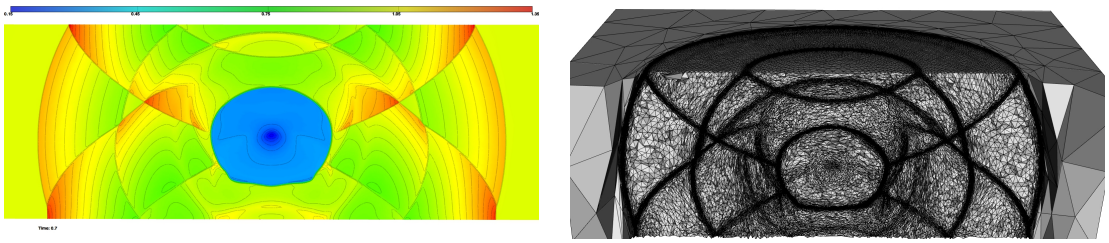


Figure 10: *Spherical blast. Reference density solution (left) at a-dimensioned time* $T = 0.7$ *and final adapted reference mesh (right) containing* $2,173,612$ *vertices and* $13,037,975$ *tetrahedra.*

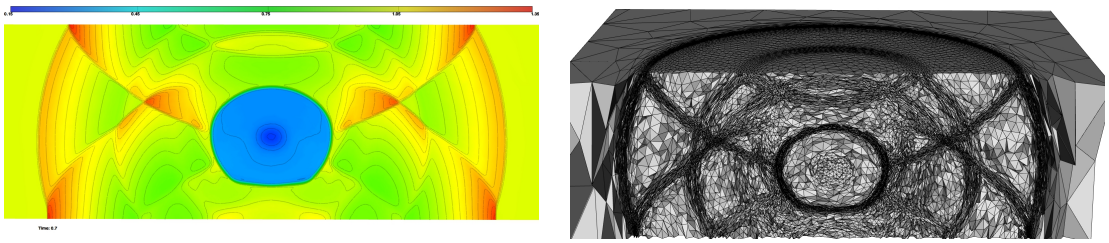


Figure 11: *Spherical blast. Density solution (left) at a-dimensioned time* $T = 0.7$ *and final adapted mesh (right) containing* $367,101$ *vertices and* $2,177,486$ *tetrahedra obtained with the* $\mathbb{P}_1$*-conservative interpolation for* $n_{adap} = 128$ *and* $\tilde{N}_{ver} = 240K$*.*

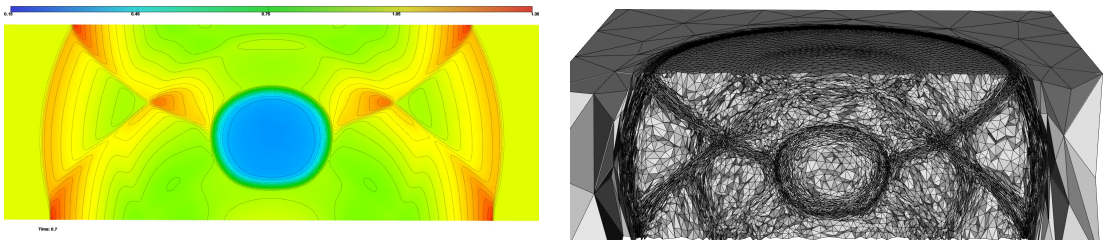


Figure 12: *Spherical blast. Density solution (left) at a-dimensioned time* $T = 0.7$ *and final adapted reference mesh (right) containing* $259,879$ *vertices and* $1,535,198$ *tetrahedra obtained with the* $\mathbb{P}_1$ *interpolation for* $n_{adap} = 128$ *and* $\tilde{N}_{ver} = 240K$*.*

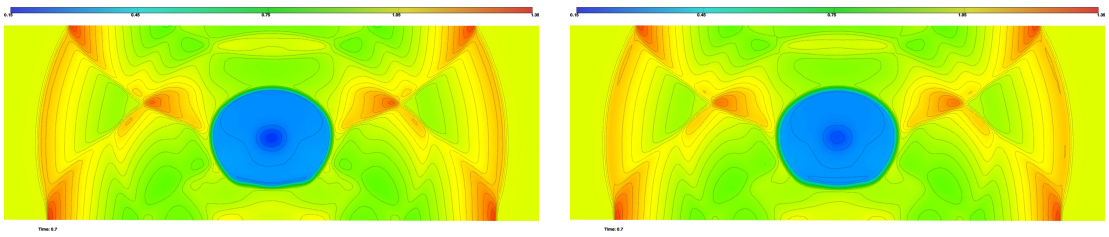


Figure 13: *Spherical blast. Density solution at a-dimensioned time* $T = 0.7$ *obtained with the* $\mathbb{P}_1$*-conservative interpolation (left) and the* $\mathbb{P}_1$ *interpolation (right) for* $n_{adap} = 16$ *and* $\tilde{N}_{ver} = 240K$*.*

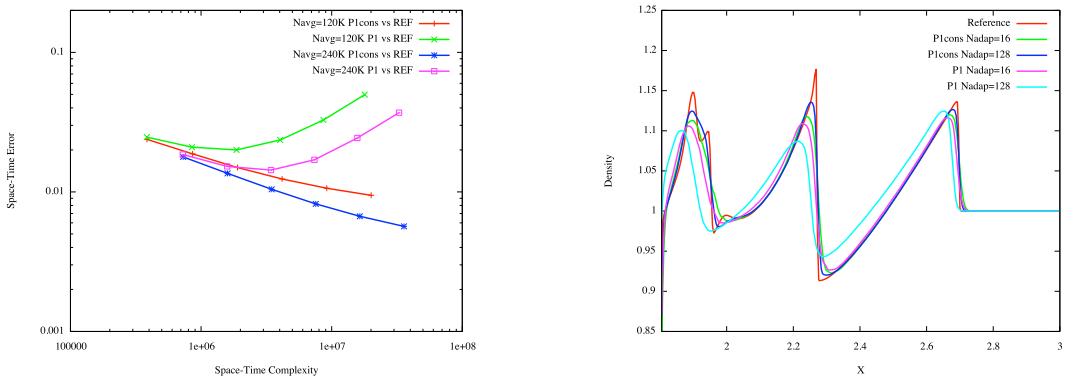


Figure 14: *Spherical blast. Left, space-time error (Equation (8)) vs. space-time complexity (Equation (9)) for the four series of adaptive simulations from* 4 *to* 128 *sub-intervals. In green and pink, adaptive simulations with the* $\mathbb{P}_1$ *interpolation and, in red and blue, with the* $\mathbb{P}_1$*-conservative interpolation. Right, final density solution extraction along a line for the reference simulation and simulations at average complexity* $240,000$ *for both interpolations with* 16 *and* 128 *sub-intervals.*

we obtain almost the same space-time mesh complexity for the $\mathbb{P}_1$ and the $\mathbb{P}_1$-conservative interpolation. Again, when the number of sub-intervals increases, we observe that the size of the final adapted mesh obtained with the $\mathbb{P}_1$-conservative interpolation is (a lot) larger than the one obtained with $\mathbb{P}_1$ interpolation. It denotes a loss of accuracy during the simulation while using the $\mathbb{P}_1$ interpolation.

The space-time error vs. the space-time complexity for each simulation is plotted in Figure 15. Final density solutions and corresponding adapted meshes are shown in Figures 17 and 18. Comments on the results are identical to the previous example. Again, for a large number of adaptation, the difference in accuracy between the two methods is significant. Thus, the conservative interpolation is mandatory to avoid the accumulation of errors due to the solution transfer which spoils considerably the solution accuracy.

Tables 6 and 7 report the CPU time distribution between the interpolation stage, the flow solver stage, and the metric computation and the mesh generation stage. As previously, all stages are run in parallel using the same number of processors. For the $\mathbb{P}_1$ interpolation, see Table 6, the 3%/60%/37% (Interpolation/Flow/Metric-Mesh) CPU distribution using 4 adapted meshes evolves to a 10%/51%/38% CPU distribution using 128 adapted meshes. Therefore, the same observations as Section 7.2.1 can be made. In this example, the average number of flow solver time steps per sub-interval is 750 for $n_{adap} = 4$ and drops to 200 for $n_{adap} = 128$. For the $\mathbb{P}_1$-conservative interpolation, see Table 7, the same behavior is observed: the 12%/55%/33% CPU distribution using 4 adapted meshes evolves to a 34%/37%/29% CPU distribution using 128 adapted meshes. This time, an almost balanced distribution between the three stages is obtained for $n_{adap} = 128$.

8 Conclusion

In this work, we have proposed a matrix-free $\mathbb{P}_1$-conservative interpolation operator that satisfies the maximum principle. This operator is based on local mesh intersections and local operations that make it memory efficient and easy to parallelize. The properties of this new operator have been verified numerically on analytical examples and adaptive simulations. These examples also point out a significative improvement in accuracy and numerical convergence obtained with the conservative interpolation as compared to the classical one when a large number of adapted meshes are considered. The accuracy of the solution is not spoiled by this stage. Consequently, for long-time simulations, the conservative interpolation is mandatory as it is only slightly sensitive to the increase of the number of interpolations.

The proposed conservative interpolation scheme can be extend to $\mathbb{P}^k$-representation of the

n_{adap}	$\mathbb{P}_1$				
	$\tilde{N}_{ver}^{120K}$	$N_{ver}^{120K}(n_{adap})$	Itp CPU	Flow CPU	Mesh CPU
4	210,131	239,090	2.91%	60.34%	36.75%
8	229,860	281,278	4.17%	63.44%	32.39%
16	245,058	316,850	5.16%	68.16%	26.68%
32	264,455	354,053	6.61%	64.35%	29.04%
64	285,301	374,908	7.95%	58.85%	33.20%
128	307,001	314,963	9.93%	51.46%	38.61%

Table 6: *City blast. Statistics for the unsteady mesh adaptation algorithm for different number of sub-intervals n_{adap} using the $\mathbb{P}_1$ interpolation. From left to right, average number of vertices per sub-interval $\tilde{N}_{ver}$ and final meshes number of vertices $N_{ver}(n_{adap})$, percentage of the total CPU time for the interpolation (Itp), the flow solver (Flow) and the metric computation and mesh generation (Mesh) stages. The total space-time mesh number of vertices is $n_{adap} \times \tilde{N}_{ver}$.*

n_{adap}	$\mathbb{P}_1$-conservative				
	$\tilde{N}_{ver}^{120K}$	$N_{ver}^{120K}(n_{adap})$	Itp CPU	Flow CPU	Mesh CPU
4	210,309	241,620	12.02%	54.57%	33.41%
8	230,148	287,236	16.76%	56.11%	27.13%
16	247,354	331,257	19.56%	58.00%	22.44%
32	270,084	384,186	23.69%	51.87%	24.44%
64	296,473	438,169	28.20%	45.04%	26.76%
128	323,835	487,440	33.72%	36.82%	29.46%

Table 7: *City blast. Statistics for the unsteady mesh adaptation algorithm for different number of sub-intervals n_{adap} using the $\mathbb{P}_1$-conservative interpolation. From left to right, average number of vertices per sub-interval $\tilde{N}_{ver}$ and final meshes number of vertices $N_{ver}(n_{adap})$, percentage of the total CPU time for the interpolation (Itp), the flow solver (Flow) and the metric computation and mesh generation (Mesh) stages. The total space-time mesh number of vertices is $n_{adap} \times \tilde{N}_{ver}$.*

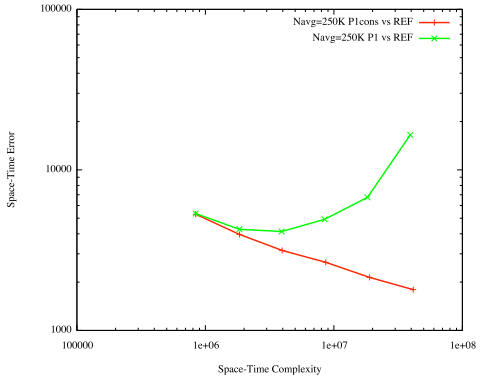


Figure 15: *City blast. Space-time error (Equation (8)) vs. space-time complexity (Equation (9)) for the adaptive simulations with the $\mathbb{P}_1$ interpolation (green) and the $\mathbb{P}_1$-conservative interpolation (red) from 4 to 128 sub-intervals.*

solution if solutions are defined at the elements. In that case, a high-order gauss quadrature is considered to recover the mass of the solution field and all the derivatives (up to order k). However, the extension to $\mathbb{P}^k$-representation of the solution defined at vertices requires more work.

References

References

[1] C. Geuzaine, B. Meys, F. Henrotte, P. Dular, W. Legros, A Galerkin projection method for mixed finite elements, IEEE Transactions on Magnetics 35 (3) (1999) 1438–1441.

[2] X. Jiao, M. Heath, Common-refinement-based data transfer between non-matching meshes in multiphysics simulations, Int. J. Numer. Meth. Engng61 (14) (2004) 2402–2427.

[3] T. Ringler, D. Randall, A potential enstrophy and energy conserving numerical scheme for solution of the shallow-water equations on a geodesic grid, Monthly Weather Review 130 (5) (2002) 1397–1410.

[4] J. Thuburn, Some conservation issues for the dynamical cores of NWP and climate models, J. Comp. Phys.227 (7) (2007) 3715–3730.

[5] F. Alauzet, P. Frey, P. George, B. Mohammadi, 3D transient fixed point mesh adaptation for time-dependent problems: Application to CFD simulations, J. Comp. Phys.222 (2007) 592–623.

[6] F. Alauzet, M. Mehrenberger, P1-conservative solution interpolation on unstructured triangular meshes, Int. J. Numer. Meth. Engng84 (13) (2010) 1552–1588.

[7] P. Farrell, M. Piggott, C. Pain, G. Gorman, Conservative interpolation between unstructured meshes via supermesh construction, Comput. Methods Appl. Mech. Engrg.198 (33-36) (2009) 2632–2642.

[8] D. S. S. Menon, Conservative interpolation on unstructured polyhedral meshes: An extension of the supermesh approach to cell-centered finite-volume variables, Comput. Methods Appl. Mech. Engrg.200 (41-44) (2011) 2797–2804.

[9] P. Farrell, J. Maddison, Conservative interpolation between volume meshes by local galerkin projection, Comput. Methods Appl. Mech. Engrg.200 (1-4) (2011) 89–100.

[10] P. George, H. Borouchaki, Delaunay triangulation and meshing. Application to finite elements, Hermès, Paris, 1998.

[11] J. Grandy, Conservative remapping and regions overlays by intersecting polyhedra, J. Comp. Phys.148 (2) (1999) 433–466.

[12] D. Eberly, 3D game engine design: a practical approach to real-time computer graphics (The Morgan Kaufmann Series in Interactive 3D Technology), 2nd Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[13] P. Frey, P. George, Mesh generation. Application to finite elements, 2nd Edition, ISTE Ltd and John Wiley & Sons, 2008.

[14] R. Löhner, Robust, vectorized search algorithms for interpolation on unstructured grids, J. Comp. Phys.118 (2) (1995) 380–387.

[15] C. Hoffmann, The problems of accuracy and robustness in geometric computation, Computer 22 (3) (1989) 31–39.

[16] A. Stewart, Local robustness and its application to polyhedral intersection, International Journal of Computational Geometry and Applications 4 (1) (1994) 87–118.

[17] R. Seidel, Convex hull computations, in: The Handbook of Discrete and Computational Geometry, Edited by J.E. Goodman and J. O'Rourke, Chapman & Hall/CRC, 2004, Ch. 22, pp. 495–512.

[18] H. Cundy, A. Rollett, Mathematical Models, 3rd Edition, Tarquin Publications, Stradbroke, England, 1989.

[19] F. Hill Jr., The pleasures of 'perp dot' products, in: Graphics Gems IV, Paul S. Heckbert Edition, Academic Press, San Diego, CA, 1994, Ch. II.5, pp. 138–148.

[20] P. George, F. Hermeline, Delaunay's mesh of a convex polygon in dimension d. Application to arbitrary polyedra, Int. J. Numer. Meth. Engng33 (1992) 975–995.

[21] F. Alauzet, A. Loseille, On the use of space filling curves for parallel anisotropic mesh adaptation, in: Proceedings of the 18th International Meshing Roundtable, Springer, 2009, pp. 337–357.

[22] L. Maréchal, A parallelization framework for numerical simulation. The LP3 library, Documentation, INRIA (Jun 2010).

[23] L. Maréchal, Handling unstructured meshes in multithreaded environments with the help of Hilbert renumbering and dynamic scheduling, Parallel ComputingSubmitted.

[24] C. Bottasso, Anisotropic mesh adaption by metric-driven optimization, Int. J. Numer. Meth. Engng60 (2004) 597–639.

[25] L. Formaggia, S. Micheletti, S. Perotto, Anisotropic mesh adaptation in computational fluid dynamics: Application to the advection-diffusion-reaction and the Stokes problems, Appl. Numer. Math.51 (4) (2004) 511–533.

[26] P. Frey, F. Alauzet, Anisotropic mesh adaptation for CFD computations, Comput. Methods Appl. Mech. Engrg.194 (48-49) (2005) 5068–5082.

[27] C. Gruau, T. Coupez, 3D tetrahedral, unstructured and anisotropic mesh generation with adaptation to natural and multidomain metric, Comput. Methods Appl. Mech. Engrg. 194 (48-49) (2005) 4951–4976.

[28] F. Hecht, B. Mohammadi, Mesh adaptation by metric control for multi-scale phenomena and turbulence, in: 35th AIAA Aerospace Sciences Meeting and Exhibit, AIAA-1997-0859, Reno, NV, USA, 1997.

[29] C. Pain, A. Humpleby, C. de Oliveira, A. Goddard, Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations, Comput. Methods Appl. Mech. Engrg.190 (2001) 3771–3796.

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

[31] F. Alauzet, Contributions aux méthodes numériques pour l'adaptation de maillage et le maillage mobile, Habilitation à Diriger des Recherches, Université Pierre et Marie Curie, Paris VI, Paris, France, 2012.

[32] F. Alauzet, Size gradation control of anisotropic meshes, Finite Elem. Anal. Des.46 (2010) 181–202.

[33] A. Loseille, R. Löhner, Adaptive anisotropic simulations in aerodynamics, in: 48th AIAA Aerospace Sciences Meeting, AIAA Paper 2010-169, Orlando, FL, USA, 2010.

[34] A. Loseille, V. Menier, Serial and parallel mesh modification through a unique cavity-based primitive, in: Proceedings of the 22th International Meshing Roundtable, Springer, 2013, pp. 541–558.

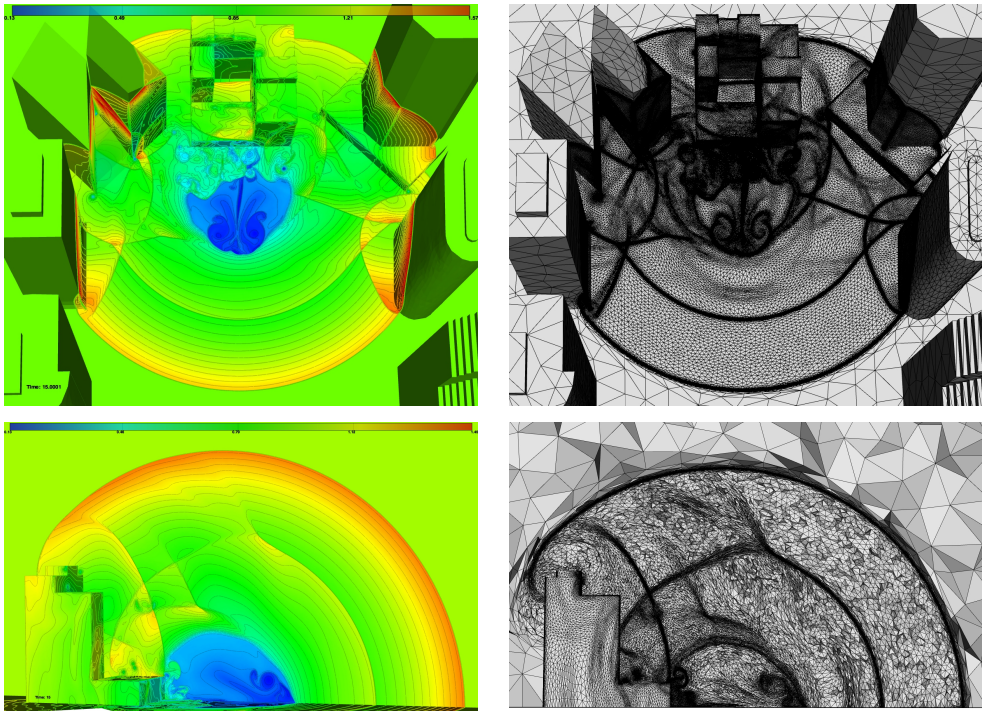


Figure 16: *City blast. Reference density solution (left) at a-dimensioned time* $T = 15$ *and final adapted reference mesh (right) containing* $4,187,548$ *vertices and* $25,249,618$ *tetrahedra.*

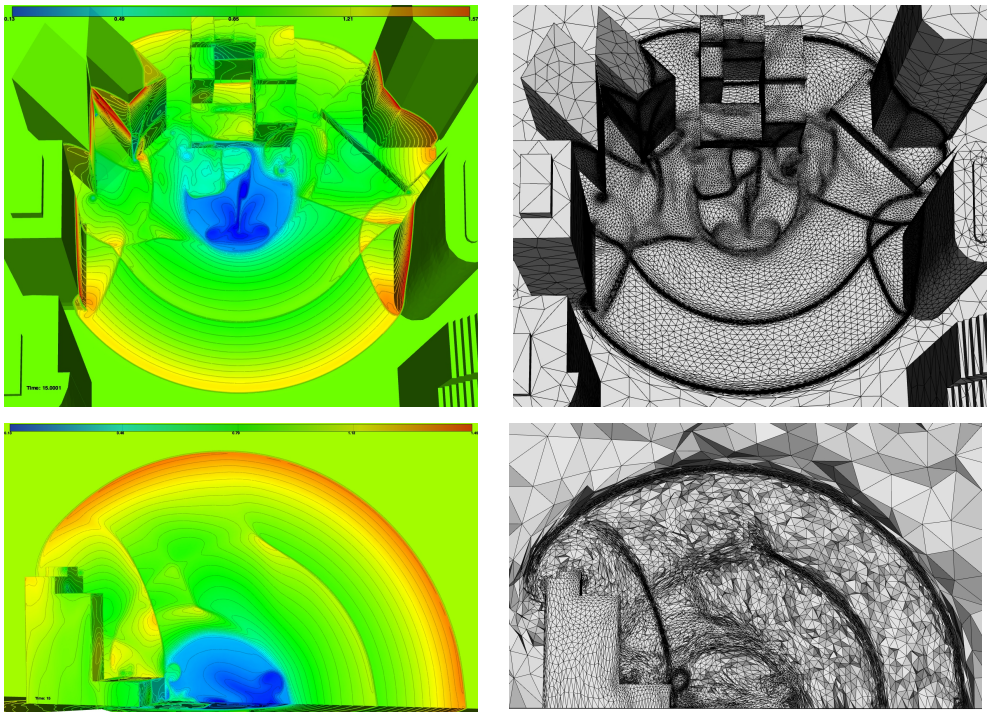


Figure 17: *City blast. Density solution (left) at a-dimensioned time* $T = 15$ *and final adapted mesh (right) containing* $487,440$ *vertices and* $2,802,472$ *tetrahedra obtained with the* $\mathbb{P}_1$*-conservative interpolation for* $n_{adap} = 128$*.*

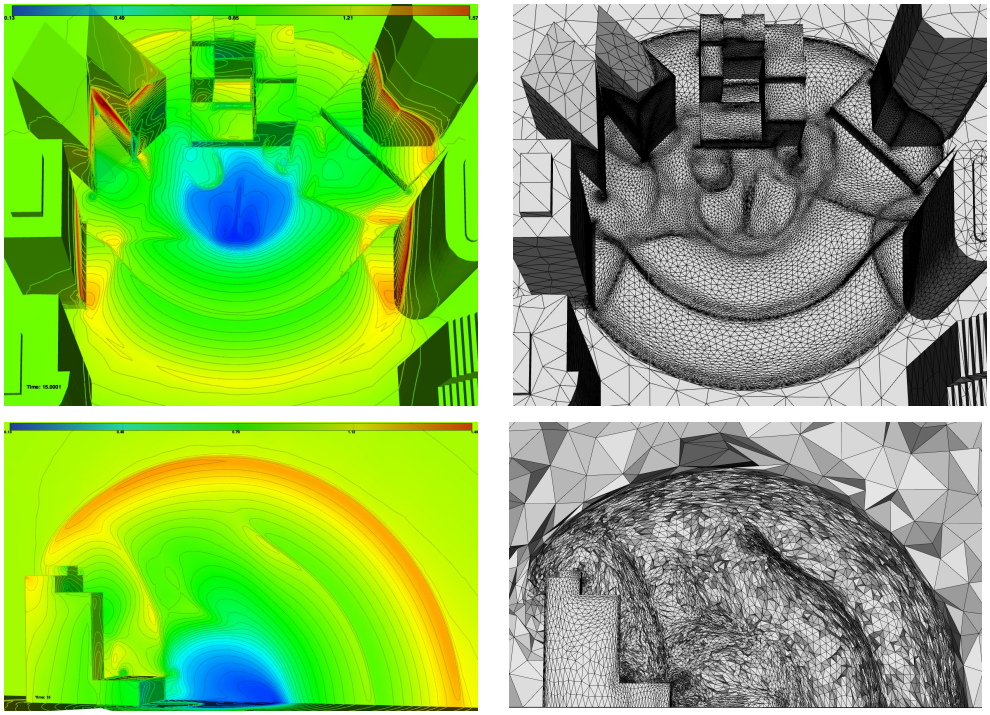


Figure 18: *City blast. Density solution (left) at a-dimensioned time* $T = 15$ *and final adapted mesh (right) containing* $314,963$ *vertices and* $1,788,719$ *tetrahedra obtained with the* $\mathbb{P}_1$ *interpolation for* $n_{adap} = 128$.

RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

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

ISSN 0249-6399