University of Liege

Faculty of applied sciences

Multiphysics integrated computational project

Project Report

Authors:
Pierre-Olivier Vanberg
Martin Lacroix
Tom Servais

Professors:
Romain Boman
Christophe Geuzaine

May, 2019

Contents

1 Introduction

Finite element methods are widely used for solving linear and nonlinear problems. However, numerical codes allowing the use and taking advantage of high order elements are still an open research field. The discontinuous Galerkin method combines features of the finite element and the finite volume framework and have been successfully applied to hyperbolic, elliptic, parabolic and mixed form problems arising from a wide range of applications. This latter has the particularity of generating discontinuous solution and uses a so called numerical flux as a coupling tool between the elements (Figure 1).

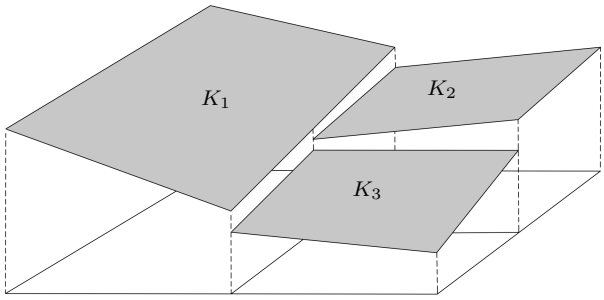


Figure 1: Aspect of the solution obtained by a discontinuous Galerkin finite element algorithm, an element of the mesh is referenced by the letter K.

This project consists in implementing this method applied to the linearized Euler equations and the acoustic perturbation equations in C++ using the finite element mesh generator Gmsh.

- In the first part of this report, a mathematical development of the method as well as an example of discretization will be presented and explained in details.
- The second part will focus on deriving the leading equation of our acoustic problem and the implementation of boundary conditions.
- A brief description of the source code is presented in the the section 4. An important part of the report is the verification of the results, analytical solutions are derived in the section 5 and compared to the numerical solution. In addition, the convergence and stability of this latter are analyzed.
- Finally, a demonstration of the capacities of our solver is presented through simulations of physical phenomena for 2D and 3D cases.

2 Discontinuous Galerkin Finite Elements

2.1 General form

Let us consider the following conservation partial differential equation:

$$\frac{\partial u}{\partial t} + \vec{\nabla} \cdot \vec{f}(u) = 0, \tag{1}$$

where $\vec{f}(u) = f_x(u)\vec{e}_x + f_y(u)\vec{e}_y + f_z(u)\vec{e}_z$ is the vector flux of $u(x, y, z, t)$.

A weak form for (1) is obtained by multiplying the equation by a test function v and integrating over a global domain Ω that will be split into K elements [1] where the solution is defined for a particular element:

$$\int_K v\frac{\partial u}{\partial t}\,dK + \int_K v\vec{\nabla}\cdot\vec{f}(u)\,dK = 0 \qquad \forall\, K \in \Omega.$$

Using Gauss theorem, the equation becomes

$$\int_K v\frac{\partial u}{\partial t}\,dK - \int_K \vec{\nabla}v\cdot\vec{f}(u)\,dK + \int_K v\vec{\nabla}\cdot\vec{f}(u)\,dK = 0,$$

then with Stokes theorem:

$$\int_K v\frac{\partial u}{\partial t}\,dK - \int_K \vec{\nabla}v\cdot\vec{f}(u)\,dK + \int_{\partial K} v\,\vec{n}\cdot\vec{f}(u)\,d\partial K = 0. \tag{2}$$

The functions are discretized in space such as

$$\begin{cases} u(x,y,z,t) = \sum_i N_i(x,y,z)u_i(t) = N_i u_i = \mathbf{N}\mathbf{u}, \\ v(x,y,z,t) = \sum_i N_i(x,y,z)v_i(t) = N_i v_i = \mathbf{N}\mathbf{v}, \\ \left[\vec{f}(u)\right](x,y,z,t) = \sum_i N_i(x,y,z)\left[\vec{f}(u)_i\right](t) = \mathbf{N}\vec{f}(\mathbf{u}), \end{cases} \tag{3}$$

where N_i is the basis function of the node i and $u_i(t) = u(x_i, y_i, z_i, t)$ is the nodal solution, injecting (3) in equation (2) leads to

$$\int_K \mathbf{v}^\top\mathbf{N}^\top\mathbf{N}\frac{\partial\mathbf{u}}{\partial t}\,dK - \int_K \mathbf{v}^\top\vec{\nabla}\mathbf{N}^\top\cdot\vec{f}(\mathbf{N}\mathbf{u})\;dK + \int_{\partial K}\mathbf{v}^\top\mathbf{N}^\top\vec{n}\cdot\vec{f}(\mathbf{N}\mathbf{u})\;d\partial K = 0.$$

This formulation being verified for all test functions v and $\mathbf{u}$ being independent of (x, y, z), $\mathbf{v}^\top$ can be simplified and the equation reduces to

$$\int_K \mathbf{N}^\top\mathbf{N}\,dK\,\frac{\partial\mathbf{u}}{\partial t} - \int_K \vec{\nabla}\mathbf{N}^\top\mathbf{N}\cdot\vec{f}(\mathbf{u})\;dK + \int_{\partial K}\mathbf{N}^\top\mathbf{N}\,\vec{n}\cdot\vec{f}(\mathbf{u})\;d\partial K = 0.$$

Performing a change of variable in the reference space $(x, y, z) \Rightarrow (\xi, \eta, \varphi) \in [-1, 1]^3$:

$$\int_{K'}\mathbf{N}^\top\mathbf{N}\det\mathbf{J}\,dK'\,\frac{\partial\mathbf{u}}{\partial t} - \int_{K'}\vec{\nabla}\mathbf{N}^\top\mathbf{N}\cdot\vec{f}(\mathbf{u})\det\mathbf{J}\,dK' + \int_{\partial K'}\mathbf{N}^\top\mathbf{N}\,\vec{n}\cdot\vec{f}(\mathbf{u})\det\mathbf{J}\,d\partial K' = 0, \tag{4}$$

where $\mathbf{J}$ is the Jacobian matrix. The integrals over the elements are then approximated by a Gaussian quadrature rule:

$$\int_K f(x,y,z)\,dK = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta,\varphi)J\,d\xi\,d\eta\,d\varphi \simeq \sum_g w_g\,f(\xi_g,\eta_g,\varphi_g)\det J_g, \tag{5}$$

where $f(\xi_g, \eta_g, \varphi_g)$ is the function evaluated at the Gauss point g and w_g its weight.

2.2 First term

Replacing (5) into the first integral of (4) leads to

$$\int_{K'}\mathbf{N}^\top\mathbf{N}\det\mathbf{J}\,dK' \simeq \sum_g w_g\mathbf{N}_g^\top\mathbf{N}_g\det\mathbf{J}_g = \sum_g\mathbf{M}_g = \mathbf{M}_k,$$

where $\mathbf{M}_k$ is called the mass matrix of the element K and $\mathbf{N}_g = \mathbf{N}(\xi_g, \eta_g, \varphi_g)$ is the evaluation of the basis function vector at a Gauss point g of the reference element K'.

2.3 Second term

The second integral in (4) can be rewritten

$$\begin{aligned}
&\int_{K'} \vec{\nabla}\mathbf{N}^\top \mathbf{N} \cdot \vec{f}\,(\mathbf{u}) \det \mathbf{J}\, dK' \\
&= \int_{K'} \left[\frac{\partial \mathbf{N}^\top}{\partial x}\mathbf{N}\vec{e}_x + \frac{\partial \mathbf{N}^\top}{\partial y}\mathbf{N}\vec{e}_y + \frac{\partial \mathbf{N}^\top}{\partial z}\mathbf{N}\vec{e}_z \right] \cdot \left[f_x\,(\mathbf{u})\,\vec{e}_x + f_y\,(\mathbf{u})\,\vec{e}_y + f_z\,(\mathbf{u})\,\vec{e}_z \right] \det \mathbf{J}\, dK' \\
&= \int_{K'} \left[\frac{\partial \mathbf{N}^\top}{\partial x}\mathbf{N} f_x\,(\mathbf{u}) + \frac{\partial \mathbf{N}^\top}{\partial y}\mathbf{N} f_y\,(\mathbf{u}) + \frac{\partial \mathbf{N}^\top}{\partial z}\mathbf{N} f_z\,(\mathbf{u}) \right] \det \mathbf{J}\, dK'. \qquad (6)
\end{aligned}$$

$\mathbf{N}(\xi, \eta, \varphi)$ being defined in the reference space, its derivative in (x, y, z) is obtained using the chain rule and the components of the Jacobian matrix:

$$\mathbf{J}^{-1} = \frac{\partial(\xi,\eta,\varphi)}{\partial(x,y,z)} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \xi}{\partial y} & \frac{\partial \xi}{\partial z} \\ \frac{\partial \eta}{\partial x} & \frac{\partial \eta}{\partial y} & \frac{\partial \eta}{\partial z} \\ \frac{\partial \varphi}{\partial x} & \frac{\partial \varphi}{\partial y} & \frac{\partial \varphi}{\partial z} \end{bmatrix} \quad \Leftrightarrow \quad \begin{cases} \frac{\partial \mathbf{N}^\top}{\partial x} = \frac{\partial \mathbf{N}^\top}{\partial \xi} J_{11} + \frac{\partial \mathbf{N}^\top}{\partial \eta} J_{21} + \frac{\partial \mathbf{N}^\top}{\partial \varphi} J_{31}, \\ \frac{\partial \mathbf{N}^\top}{\partial y} = \frac{\partial \mathbf{N}^\top}{\partial \xi} J_{12} + \frac{\partial \mathbf{N}^\top}{\partial \eta} J_{22} + \frac{\partial \mathbf{N}^\top}{\partial \varphi} J_{32}, \\ \frac{\partial \mathbf{N}^\top}{\partial z} = \frac{\partial \mathbf{N}^\top}{\partial \xi} J_{13} + \frac{\partial \mathbf{N}^\top}{\partial \eta} J_{23} + \frac{\partial \mathbf{N}^\top}{\partial \varphi} J_{33}. \end{cases} \qquad (7)$$

Replacing (5) and (7) in the equation (6) leads to

$$\begin{aligned}
&\int_{K'} \vec{\nabla}\mathbf{N}^\top \mathbf{N} \cdot \vec{f}\,(\mathbf{u}) \det \mathbf{J}\, dK' \\
&\simeq \sum_g w_g \left[\frac{\partial \mathbf{N}_g^\top}{\partial \xi} J_{11g} + \frac{\partial \mathbf{N}_g^\top}{\partial \eta} J_{21g} + \frac{\partial \mathbf{N}_g^\top}{\partial \varphi} J_{31g} \right] \mathbf{N}_g f_x\,(\mathbf{u}_k) \det \mathbf{J}_g \\
&+ \sum_g w_g \left[\frac{\partial \mathbf{N}_g^\top}{\partial \xi} J_{12g} + \frac{\partial \mathbf{N}_g^\top}{\partial \eta} J_{22g} + \frac{\partial \mathbf{N}_g^\top}{\partial \varphi} J_{32g} \right] \mathbf{N}_g f_y\,(\mathbf{u}_k) \det \mathbf{J}_g \\
&+ \sum_g w_g \left[\frac{\partial \mathbf{N}_g^\top}{\partial \xi} J_{13g} + \frac{\partial \mathbf{N}_g^\top}{\partial \eta} J_{23g} + \frac{\partial \mathbf{N}_g^\top}{\partial \varphi} J_{33g} \right] \mathbf{N}_g f_z\,(\mathbf{u}_k) \det \mathbf{J}_g \\
&= \mathbf{S}_x(\mathbf{u}) + \mathbf{S}_y(\mathbf{u}) + \mathbf{S}_z(\mathbf{u}) = \mathbf{S}(\mathbf{u})_k,
\end{aligned}$$

where $\mathbf{S}(\mathbf{u})_k$ is the stiffness vector of the element K.

2.4 Third term

The vector tangent to the variation of a scalar function f in space is $\vec{\nabla} f$, the normal $\vec{n}_g$ of a surface at an arbitrary Gauss point g of an element is the vector such as $\vec{n}_g \perp \vec{\nabla} N_{ig}\ \forall\ i \in K$ where N_i is the shape function of the node i and $||\vec{n}_g|| = 1$.

$$\vec{n}_g = \frac{\vec{\nabla} N_{ig} \times \vec{\nabla} N_{j_g}}{||\vec{\nabla} N_{ig} \times \vec{\nabla} N_{j_g}||} \text{ in 3D}, \qquad \vec{n}_g = \frac{\vec{\nabla} N_{ig} \times [0,0,1]}{||\vec{\nabla} N_{ig} \times [0,0,1]||} \text{ in 2D}, \qquad \vec{n}_g = \pm 1 \text{ in 1D},$$

for any $i \neq j \in K$, the normal of the surface is then oriented towards the outside of the

element. Replacing (5) in the third integral of (4) leads to

$$
\begin{aligned}
&\int_{\partial K'} \mathbf{N}^\top \mathbf{N}\, \vec{n} \cdot \vec{f}\,(\mathbf{u}) \det \mathbf{J}\, d\partial K' \qquad (8)\\
&= \int_{\partial K'} \mathbf{N}^\top \mathbf{N} \left[n_x \vec{e}_x + n_y \vec{e}_y + n_z \vec{e}_z \right] \cdot \left[f_x\,(\mathbf{u})\, \vec{e}_x + f_y\,(\mathbf{u})\, \vec{e}_y + f_z\,(\mathbf{u})\, \vec{e}_z \right] \det \mathbf{J}\, d\partial K' \\
&= \int_{\partial K'} \mathbf{N}^\top \mathbf{N} \left[n_x f_x\,(\mathbf{u}) + n_y f_y\,(\mathbf{u}) + n_z f_z\,(\mathbf{u}) \right] \det \mathbf{J}\, d\partial K' \\
&\simeq \sum_g w_g \mathbf{N}_g^\top \mathbf{N}_g \left[n_{xg}\, f_x\,(\mathbf{u}_k) + n_{y_g}\, f_y\,(\mathbf{u}_k) + n_{zg}\, f_z\,(\mathbf{u}_k) \right] \det \mathbf{J}_g = \mathbf{F}(\mathbf{u})_k,
\end{aligned}
$$

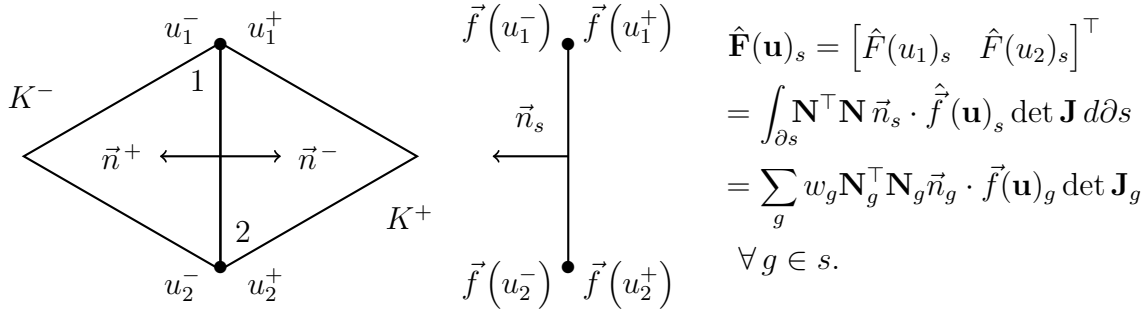
where $\mathbf{F}(\mathbf{u})_k$ is the nodal flux vector of the element K, the Gauss points i are taken at the surface $\partial K'$ and are thus different from the ones used to compute the two volume integrals. With this definition, the flux (and the solution) at a node is not unique. Practically, the physical flux $\vec{f}(u)$ from equation (8) can be replaced by a centered numerical flux

$$\hat{\vec{f}}(u) = \frac{\vec{f}\,(u^+) + \vec{f}\,(u^-)}{2},$$

where the $\pm$ indices denote the solutions from element for which the outer normal is in the same or in the opposite direction to the normal of the face entity (Figure 2). In the particular case of a wave propagation problem, a clever choice could be the use the Rusanov flux [2] for its simplicity and the relatively small communication stencil:

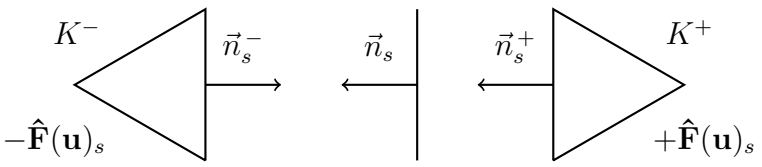
$$\hat{\vec{f}}(u)_g = \frac{\vec{f}\,(u^+) + \vec{f}\,(u^-) + \vec{n}_g\, c_0\,(u^+ - u^-)}{2},$$

where c_0 is the maximum wave speed and $\vec{n}_g$ is the normal of the face entity separating the elements K^+ and K^- at the considered integration point g.



$$
\begin{aligned}
\hat{\mathbf{F}}(\mathbf{u})_s &= \begin{bmatrix} \hat{F}(u_1)_s & \hat{F}(u_2)_s \end{bmatrix}^\top \\
&= \int_{\partial s} \mathbf{N}^\top \mathbf{N}\, \vec{n}_s \cdot \hat{\vec{f}}\,(\mathbf{u})_s \det \mathbf{J}\, d\partial s \\
&= \sum_g w_g \mathbf{N}_g^\top \mathbf{N}_g \vec{n}_g \cdot \vec{f}\,(\mathbf{u})_g \det \mathbf{J}_g \\
&\forall\, g \in s.
\end{aligned}
$$

Figure 2: Example in 2D of numerical flux $\hat{\mathbf{F}}(\mathbf{u})_s$ computed for a surface entity s separating two elements K. Finally, the flux computed at each interface is added or subtracted to the total flux of the element if this latter has its own normal in the same or opposite direction to the one of the face entity (Figure 3).



$$\hat{\mathbf{F}}(\mathbf{u})_k = \sum_s \pm \hat{\mathbf{F}}(\mathbf{u})_s = \begin{bmatrix} \hat{F}(u_1)_k & \hat{F}(u_2)_k & \hat{F}(u_3)_k \end{bmatrix}^\top \quad \forall\, s \in K.$$

Figure 3: Contribution of a surface s to the total flux in an element K according to the orientation of its outer normal.

2.5 Time integration

The original equation (1) has now the following form:

$$\mathbf{M}_k \frac{\partial \mathbf{u}_k}{\partial t} - \mathbf{S}(\mathbf{u})_k + \hat{\mathbf{F}}(\mathbf{u})_k = 0. \tag{9}$$

The time discretization can be performed using an explicit forward Euler scheme, the time derivative is approximated by

$$\frac{\partial \mathbf{u}}{\partial t} \simeq \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t}. \tag{10}$$

Injecting (10) into (9) leads to

$$\mathbf{u}_k^{n+1} = \mathbf{u}_k^n + \Delta t\, \mathbf{M}_k^{-1} \Big[\mathbf{S}\left(\mathbf{u}^n\right)_k - \hat{\mathbf{F}}\left(\mathbf{u}^{\mathbf{n}}\right)_k \Big].$$

for each element in the domain Ω. A more accurate but slower explicit iterative method is known as Runge-Kutta algorithm and consists in computing a weighted average of four intermediate increments to obtain the solution at the next time step:

$$\begin{cases} \mathbf{h}_1 = \Delta t\, \mathbf{M}_k^{-1} \Big[\mathbf{S}\left(\mathbf{u}^n\right)_k - \hat{\mathbf{F}}\left(\mathbf{u}^{\mathbf{n}}\right)_k \Big], \\ \mathbf{h}_2 = \Delta t\, \mathbf{M}_k^{-1} \Big[\mathbf{S}\left(\mathbf{u}^n + \mathbf{h}_1/2\right)_k - \hat{\mathbf{F}}\left(\mathbf{u}^{\mathbf{n}} + \mathbf{h}_1/2\right)_k \Big], \\ \mathbf{h}_3 = \Delta t\, \mathbf{M}_k^{-1} \Big[\mathbf{S}\left(\mathbf{u}^n + \mathbf{h}_2/2\right)_k - \hat{\mathbf{F}}\left(\mathbf{u}^{\mathbf{n}} + \mathbf{h}_2/2\right)_k \Big], \\ \mathbf{h}_4 = \Delta t\, \mathbf{M}_k^{-1} \Big[\mathbf{S}\left(\mathbf{u}^n + \mathbf{h}_3\right)_k - \hat{\mathbf{F}}\left(\mathbf{u}^{\mathbf{n}} + \mathbf{h}_3\right)_k \Big], \end{cases}$$

and the solution of the next time step is

$$\mathbf{u}_k^{n+1} = \mathbf{u}_k^n + \frac{1}{6}\left(\mathbf{h}_1 + 2\mathbf{h}_2 + 2\mathbf{h}_3 + \mathbf{h}_4\right).$$

3 Acoustic Problem

3.1 Linearized Euler equations

1) The conservation of mass expressed in differential form is given by

$$\frac{\partial \rho}{\partial t} + \vec{\nabla} \cdot (\rho \vec{v}) = 0, \tag{11}$$

where ρ is the density and $\vec{v} = v_x\vec{e}_x + v_y\vec{e}_y + v_z\vec{e}_z$ the velocity field. Assuming small acoustic perturbations, the velocity, density and pressure fields can be decomposed into a mean quantity x_0 and a small variations x' around this latter:

$$\vec{v} = \vec{v}_0 + \vec{v}', \qquad \rho = \rho_0 + \rho', \qquad p = p_0 + p',$$

where p is the acoustic pressure field. The conservation of mass (11) becomes

$$\begin{aligned} &\frac{\partial(\rho_0+\rho')}{\partial t} + \vec{\nabla}\cdot[(\rho_0+\rho')(\vec{v}_0+\vec{v}')] = 0 \\ \Leftrightarrow\ &\frac{\partial\rho_0}{\partial t} + \frac{\partial\rho'}{\partial t} + \vec{\nabla}\cdot(\rho_0\vec{v}_0 + \rho_0\vec{v}' + \rho'\vec{v}_0 + \rho'\vec{v}') = 0, \end{aligned} \tag{12}$$

where the mean flux satisfies the continuity equation and quadratic term of the perturbation may be neglected:

$$\begin{cases} \dfrac{\partial p_0}{\partial t} + \vec{\nabla}\cdot(\rho_0\vec{v}_0) = 0, \\ \vec{\nabla}\cdot(\rho'\vec{v}') \simeq 0. \end{cases} \tag{13}$$

For a barotropic flow and isotropic process, the pressure $p = f(\rho)$ can be expressed as a function of the density leading to the following decomposition:

$$p_0 + p' = f(\rho_0+\rho') = f(\rho_0) + \frac{\partial f}{\partial\rho}(\rho_0)\rho' + \mathcal{O}({\rho'}^2) \quad \Leftrightarrow \quad p' \simeq \frac{\partial f}{\partial\rho}(\rho_0)\rho' = c_0^2\rho', \tag{14}$$

where $c = \sqrt{\partial p/\partial\rho}$ is the speed of sound. Injecting (13) and (14) into equation (12) gives

$$\frac{\partial p'}{\partial t} + \vec{\nabla}\cdot\left(c_0^2\rho_0\vec{v}' + \vec{v}_0\right) = 0. \tag{15}$$

2) The Cauchy momentum equation in its general convective form is given by

$$\rho\frac{D\vec{v}}{Dt} = \vec{\nabla}\cdot\underline{\underline{\sigma}} + \rho\vec{g} \qquad \Leftrightarrow \qquad \rho\frac{\partial\vec{v}}{\partial t} + \rho(\vec{v}\cdot\vec{\nabla})\vec{v} = \vec{\nabla}\cdot\underline{\underline{\tau}} - \vec{\nabla}\cdot(p\underline{\underline{I}}) + \rho\vec{g},$$

where the operator $D/Dt = \partial/\partial t + \vec{v}\cdot\vec{\nabla}$ is the material derivative and the stress tensor $\underline{\underline{\sigma}}$ can be expressed as the sum of its deviatoric part $\underline{\underline{\tau}}$ and a pressure term $-p\underline{\underline{I}}$. For a non-viscous fluid without body forces, $\underline{\underline{\tau}} = \vec{0}$ and $\vec{g} = \vec{0}$, the equation (3.1) can thus be rewritten as

$$\frac{\partial\vec{v}}{\partial t} + (\vec{v}\cdot\vec{\nabla})\vec{v} + \frac{1}{\rho}\vec{\nabla}\cdot(p\underline{\underline{I}}) = \vec{0},$$

or, assuming small acoustic perturbations:

$$\frac{\partial(\vec{v}_0+\vec{v}')}{\partial t} + \left[(\vec{v}_0+\vec{v}')\cdot\vec{\nabla}\right](\vec{v}_0+\vec{v}') + \frac{1}{\rho_0+\rho'}\vec{\nabla}\cdot\left[(p_0+p')\underline{\underline{I}}\right] = \vec{0}, \tag{16}$$

the three terms of the left hand side can be decomposed as

$$\begin{cases} \dfrac{\partial(\vec{v}_0+\vec{v}')}{\partial t} = \dfrac{\partial\vec{v}_0}{\partial t} + \dfrac{\partial\vec{v}'}{\partial t}, \\ \left[(\vec{v}_0+\vec{v}')\cdot\vec{\nabla}\right](\vec{v}_0+\vec{v}') = (\vec{v}_0\cdot\vec{\nabla})\vec{v}_0 + (\vec{v}_0\cdot\vec{\nabla})\vec{v}' + (\vec{v}'\cdot\vec{\nabla})\vec{v}_0 + (\vec{v}'\cdot\vec{\nabla})\vec{v}', \\ \frac{1}{\rho_0+\rho'}\vec{\nabla}\cdot\left[(p_0+p')\underline{\underline{I}}\right] = \frac{1}{\rho_0}\vec{\nabla}\cdot(p_0\underline{\underline{I}}) + \frac{1}{\rho_0}\vec{\nabla}\cdot(p'\underline{\underline{I}}) - \frac{\rho'}{\rho_0^2}\vec{\nabla}\cdot(p_0\underline{\underline{I}}) - \frac{\rho'}{\rho_0^2}\vec{\nabla}\cdot(p'\underline{\underline{I}}), \end{cases} \tag{17}$$

where the mean flux satisfies Navier-Stokes equation and quadratic term of the perturbation may be neglected:

$$\begin{cases} \dfrac{\partial \vec{v}_0}{\partial t} + (\vec{v}_0 \cdot \vec{\nabla})\vec{v}_0 + \dfrac{1}{\rho_0}\vec{\nabla} \cdot (p_0 \underline{\underline{I}}) = \vec{0}, \\ \vec{v}\,' \cdot \vec{\nabla}\vec{v}\,' \simeq \vec{0}, \\ \dfrac{\rho'}{\rho_0^2}\vec{\nabla} \cdot (p' \underline{\underline{I}}) \simeq \vec{0}. \end{cases} \tag{18}$$

Injecting (17) and (18) into the equation (16) leads to

$$\frac{\partial \vec{u}\,'}{\partial t} + (\vec{v}_0 \cdot \vec{\nabla})\vec{v}\,' + (\vec{v}\,' \cdot \vec{\nabla})\vec{v}_0 + \frac{1}{\rho_0}\vec{\nabla} \cdot (p' \underline{\underline{I}}) - \frac{\rho'}{\rho_0^2}\vec{\nabla} \cdot (p_0 \underline{\underline{I}}) = \vec{0}. \tag{19}$$

Using (14) and the general relation

$$\vec{\nabla} \cdot (\vec{a} \otimes \vec{b}) = \vec{a}(\vec{\nabla} \cdot \vec{b}) + (\vec{b} \cdot \vec{\nabla})\vec{a}$$

allows us to express (19) as

$$\begin{aligned} & \frac{\partial \vec{v}\,'}{\partial t} + \vec{v}\,'(\vec{\nabla} \cdot \vec{v}_0) - \vec{v}\,'(\vec{\nabla} \cdot \vec{v}_0) + (\vec{v}_0 \cdot \vec{\nabla})\vec{v}\,' + (\vec{v}\,' \cdot \vec{\nabla})\vec{v}_0 = \frac{p'}{c_0^2\rho_0^2}\vec{\nabla} \cdot (p_0 \underline{\underline{I}}) \\ \Leftrightarrow\ & \frac{\partial \vec{v}\,'}{\partial t} + \vec{\nabla} \cdot \left(\vec{v}\,' \otimes \vec{v}_0 + \frac{1}{\rho_0} p' \underline{\underline{I}} \right) = \vec{v}\,'(\vec{\nabla} \cdot \vec{v}_0) - (\vec{v}\,' \cdot \vec{\nabla})\vec{v}_0 + \frac{p'}{c_0^2\rho_0^2}\vec{\nabla} \cdot (p_0 \underline{\underline{I}}). \end{aligned} \tag{20}$$

Finally, (15) and (20) form together the governing equations of our problem:

$$\begin{aligned} & \begin{cases} \dfrac{\partial p'}{\partial t} + \vec{\nabla} \cdot (c_0^2\rho_0\vec{v}\,' + \vec{v}_0) = 0, \\ \dfrac{\partial \vec{v}\,'}{\partial t} + \vec{\nabla} \cdot \left(\vec{v}\,' \otimes \vec{v}_0 + \dfrac{1}{\rho_0} p' \underline{\underline{I}} \right) = \vec{v}\,'(\vec{\nabla} \cdot \vec{v}_0) - (\vec{v}\,' \cdot \vec{\nabla})\vec{v}_0 + \dfrac{p'}{c_0^2\rho_0^2}\vec{\nabla} \cdot (p_0 \underline{\underline{I}}), \end{cases} \\ \Leftrightarrow\ & \begin{cases} \dfrac{\partial p'}{\partial t} + \vec{\nabla} \cdot \vec{f}(\vec{v}) = 0, \\ \dfrac{\partial \vec{v}\,'}{\partial t} + \vec{\nabla} \cdot \underline{\underline{F}}(\vec{v}, p) = \vec{s}(\vec{v}, p). \end{cases} \end{aligned} \tag{21}$$

3.2 Conservative form

In order to write the system (21) into a more compact form [3], we introduce an array of field variables $\vec{u}$, an array of flux vectors $\underline{\underline{F}}(\vec{u})$ and an array $\vec{s}(\vec{u})$ such as

$$\vec{u} = \begin{bmatrix} p' \\ v'_x \\ v'_y \\ v'_z \end{bmatrix}, \qquad \underline{\underline{F}}(\vec{u}) = \begin{bmatrix} c_0^2\rho_0 v'_x + v_{0x}p' & c_0^2\rho_0 v'_y + v_{0y}p' & c_0^2\rho_0 v'_z + v_{0z}p' \\ v'_x v_{0x} + \frac{p'}{\rho_0} & v'_x v_{0y} & v'_x v_{0z} \\ v'_y v_{0x} & v'_y v_{0y} + \frac{p'}{\rho_0} & v'_y v_{0z} \\ v'_z v_{0x} & v'_z v_{0y} & v'_z v_{0z} + \frac{p'}{\rho_0} \end{bmatrix},$$

$$\vec{s}(\vec{u}) = \begin{bmatrix} 0 \\ v'_x \left(\frac{\partial v_{0x}}{\partial x} + \frac{\partial v_{0y}}{\partial y} + \frac{\partial v_{0z}}{\partial z} \right) - v'_x \frac{\partial v_{0x}}{\partial x} - v'_y \frac{\partial v_{0x}}{\partial y} - v'_z \frac{\partial v_{0x}}{\partial z} + \frac{p'}{c_0^2\rho_0^2} \frac{\partial p_0}{\partial x} \\ v'_y \left(\frac{\partial v_{0x}}{\partial x} + \frac{\partial v_{0y}}{\partial y} + \frac{\partial v_{0z}}{\partial z} \right) - v'_x \frac{\partial v_{0y}}{\partial x} - v'_y \frac{\partial v_{0y}}{\partial y} - v'_z \frac{\partial v_{0y}}{\partial z} + \frac{p'}{c_0^2\rho_0^2} \frac{\partial p_0}{\partial y} \\ v'_z \left(\frac{\partial v_{0x}}{\partial x} + \frac{\partial v_{0y}}{\partial y} + \frac{\partial v_{0z}}{\partial z} \right) - v'_x \frac{\partial v_{0z}}{\partial x} - v'_y \frac{\partial v_{0z}}{\partial y} - v'_z \frac{\partial v_{0z}}{\partial z} + \frac{p'}{c_0^2\rho_0^2} \frac{\partial p_0}{\partial z} \end{bmatrix},$$

leading to the following system of 4 equations:

$$\frac{\partial \vec{u}}{\partial t} + \vec{\nabla} \cdot \underline{\underline{F}}(\vec{u}) = \vec{s}(\vec{u}).$$

For a constant mean flow $\vec{v}_0$ and pressure field p_0 in space, one has $\vec{\nabla} \cdot \vec{v}_0 = 0$, $\vec{\nabla} \cdot (p_0 \underline{\underline{I}}) = \vec{0}$ and $(\vec{v}' \cdot \vec{\nabla})\vec{v}_0 = \vec{0}$ so that the right hand side disappears:

$$\frac{\partial \vec{u}}{\partial t} + \vec{\nabla} \cdot \underline{\underline{F}}(\vec{u}) = 0, \tag{22}$$

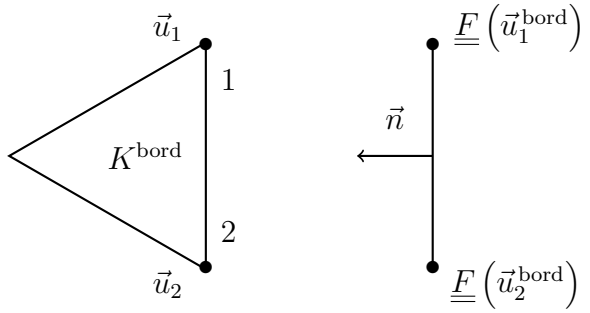
where the 4 unknowns are coupled by the physical flux $\underline{\underline{F}}(p', v'_x, v'_y, v'_z)$. The equations are thus solved simultaneously for each time step at the end of which the flux is updated. As mentioned in the previous chapter, the Rusanov flux is used to couple two adjacent elements:

$$\underline{\underline{\hat{F}}}\left(\vec{u}\right)_g = \frac{\underline{\underline{F}}\left(\vec{u}^{\,+}\right) + \underline{\underline{F}}\left(\vec{u}^{\,-}\right) + \vec{n}_g \, c_0 \left(\vec{u}^{\,-} - \vec{u}^{\,+}\right)}{2}. \tag{23}$$

3.3 Boundary conditions

Although the flux coupling between elements is well defined in the domain, one of the neighbour elements sharing a face at the boundary of the domain disappears and the numerical flux (23) is no longer defined, boundary conditions must be implemented.

1) The slip wall condition [4] consists in preventing the wave from penetrating the wall, the methodology used to implement this latter is called the Weak-Riemann approach. The flux coupling is performed by replacing the flux $\underline{\underline{F}}\left(\vec{u}\right)$ at the boundary by the following formula presented in Figure 4.



$$\vec{u}^{\,\text{bord}} = \begin{bmatrix} p'^{\,\text{in}} \\ v_x'^{\,\text{in}} - \left(\vec{v}'^{\,\text{in}} \cdot \vec{n}\right) n_x \\ v_y'^{\,\text{in}} - \left(\vec{v}'^{\,\text{in}} \cdot \vec{n}\right) n_y \\ v_z'^{\,\text{in}} - \left(\vec{v}'^{\,\text{in}} \cdot \vec{n}\right) n_z \end{bmatrix},$$

Figure 4: Example of slip wall condition at a boundary in 2D with T3 elements. For curved surfaces, $\vec{n}$ is not constant over this latter and $\underline{\underline{F}}\left(\vec{u}^{\,\text{bord}}\right)$ is thus computed at each Gauss point for a correct evaluation of the surface integral.

where $\vec{n}$ denotes the normal of the face entity and $\vec{v}$ is the the velocity field at the considered Gauss point. In practice, the flux $\underline{\underline{F}}(\vec{u}^{\,\text{bord}})_g$ at the integration point g is computed and stored in a ghost element before being replaced in the boundary element K^{bord}.

2) Open boundary conditions for acoustic waves in the time domain are still an open problem, a simple approach [3] is the application of a one-dimensional characteristic boundary condition, although producing small reflections in the domain of interest. Let us define R and R^{-1} as

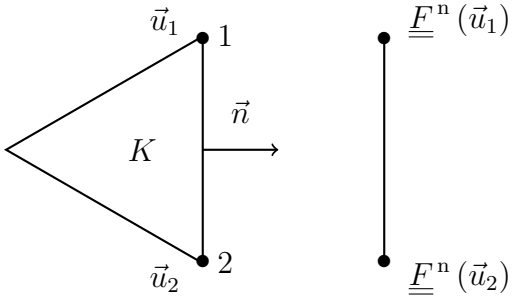
$$R = \frac{1}{2}\begin{bmatrix} \rho_0 c_0 & 0 & 0 & 0 \\ n_x & -n_x & t_x & s_x \\ n_y & -n_y & t_y & s_y \\ n_z & -n_z & t_z & s_z \end{bmatrix}, \qquad R^{-1} = \frac{1}{2}\begin{bmatrix} \frac{1}{\rho_0 c_0} & n_x & n_y & n_z \\ \frac{1}{\rho_0 c_0} & -n_x & -n_y & -n_z \\ 0 & -t_x & -t_y & -t_z \\ 0 & -s_x & -s_y & -s_z \end{bmatrix},$$

where $\vec{n}$ is the outward-pointing normal to the boundary, i.e. the normal of the element's face, $\vec{t}$ and $\vec{s}$ two vectors such that $(\vec{n}, \vec{t}, \vec{s})$ form an orthonormal basis on the boundary face. Computing the characteristic variables requiring the multiplication $R^{-1}\vec{u}$, the characteristic fluxes eliminating the ingoing mode can then be set to zero by further multiplying by

$$A = \begin{bmatrix} c_0 + \vec{v}_0 \cdot \vec{n} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda \vec{v}_0 \cdot \vec{n} & 0 \\ 0 & 0 & 0 & \lambda \vec{v}_0 \cdot \vec{n} \end{bmatrix} \quad \text{with} \quad \begin{cases} \lambda = 0 & \text{if } \vec{v}_0 \cdot \vec{n} < 0, \\ \lambda = 1 & \text{otherwise.} \end{cases}$$

Finally, if $\vec{v}_0 = \vec{0}$, the normal flux at a node of the boundary surface relative to the element K (Figure 5) is obtained by

$$\underline{\underline{F}}^{\mathrm{n}}(\vec{u}) = RAR^{-1}\vec{u} = \frac{1}{4}\begin{bmatrix} c_0 & c_0^2 n_x \rho_0 & c_0^2 n_y \rho_0 & c_0^2 n_z \rho_0 \\ n_x/\rho_0 & c_0 n_x^2 & c_0 n_x n_y & c_0 n_x n_z \\ n_y/\rho_0 & c_0 n_x n_y & c_0 n_y^2 & c_0 n_y n_z \\ n_z/\rho_0 & c_0 n_x n_z & c_0 n_y n_z & c_0 n_z^2 \end{bmatrix}\begin{bmatrix} p' \\ {v_x}' \\ {v_y}' \\ {v_z}' \end{bmatrix}.$$



$$\underline{\underline{F}}^{\mathrm{out}}(\vec{u}) = \frac{1}{4}\begin{bmatrix} p'c_0 + c_0^2\rho_0\left(n_x v_x' + n_y v_y' + n_z v_z'\right) \\ \frac{p' n_x}{\rho_0} + c_0 n_x\left(n_x v_x' + n_y v_y' + n_z v_z'\right) \\ \frac{p' n_y}{\rho_0} + c_0 n_y\left(n_x v_x' + n_y v_y' + n_z v_z'\right) \\ \frac{p' n_z}{\rho_0} + c_0 n_z\left(n_x v_x' + n_y v_y' + n_z v_z'\right) \end{bmatrix}.$$

Figure 5: Example of numerical flux for an open boundary condition in 2D and a T3 element. For curved surfaces, the normal fluxes have to be imposed at the Gauss points g since the normal $\vec{n}_g$ is not constant over the boundary.

4 Implementation

4.1 Main function

The main function of the implementation is contained in `dgalerkin.cpp`. It requires two arguments: a `.msh` and a `.conf` file. In the `.conf` file, the parameters needed for the simulation can be modified easily by the user. These are summarized below:

- the time step;
- the time duration of the simulation;
- the saving rate of the data;
- the element type (Lagrange, Isoparametric);
- the time integration method (Euler order 1, Runge-Kutta order 4);
- the name of the boundary (Gmsh physical name) and the corresponding type of boundary condition;
- the number of threads used OpenMP;
- the mean flow parameters (velocity, density, speed of sound);
- the source parameters (position, size, frequency, amplitude);
- the initial condition (position, size, amplitude);
- the name of the output file (`.msh` readable by Gmsh).

The `.msh` file contains the mesh of the geometry on which the simulation will be performed and is generated with the Gmsh software. The boundary conditions are specified as physical entities named as in the `.conf` file.

The first task of the main function is to load the `.conf` file thanks to the configuration class in `configParser.cpp`. A `Mesh` object is then created in order to compute and collect the mesh data and parameters. Secondly, the main function initializes the vector solution and enforce the initial conditions. For the moment, only Gaussian shaped initial conditions are fully parametrizable.

```
std::vector<std::vector<double>> u(4,std::vector<double>(mesh.getNumNodes(), 0));

for(int i=0; i<config.initConditions.size(); ++i){
    double x = config.initConditions[i][1];
    double y = config.initConditions[i][2];
    double z = config.initConditions[i][3];
    double size = config.initConditions[i][4];
    double amp = config.initConditions[i][5];

    for(int n=0; n<mesh.getNumNodes(); n++){
        std::vector<double> coord, paramCoord;
        gmsh::model::mesh::getNode(mesh.getElNodeTags()[n], coord, paramCoord);
       u[0][n] += amp*exp(-((coord[0]-x)*(coord[0]-x) +
                           (coord[1]-y)*(coord[1]-y) +
                           (coord[2]-z)*(coord[2]-z))/size);
    }
}
```

Listing 1: Initial conditions.

Once all the information required to use the DG-FEM is collected, `solveForwardEule r()` or `solveRungeKutta()` implemented in `solver.cpp` are called to perform the DG method over the entire mesh and to time integrate using either the 1^{st} order forward Euler method or the 4^{th} order Runge-Kutta method.

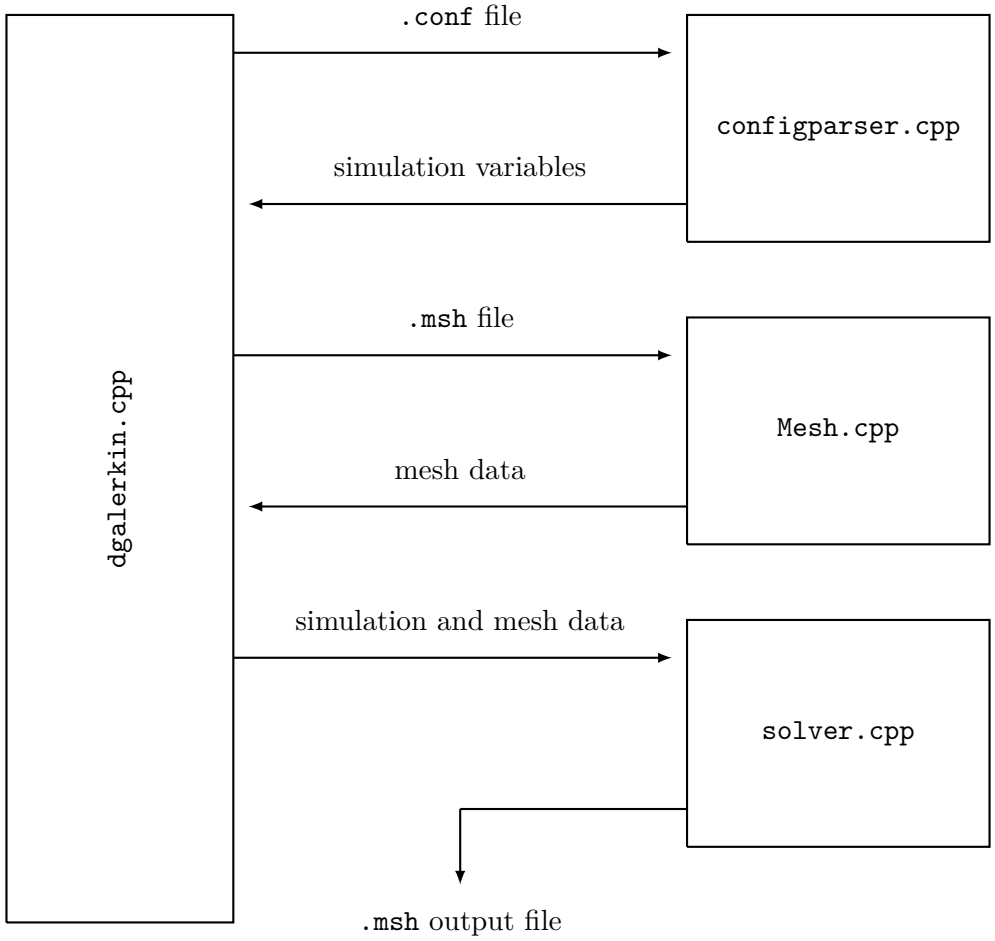


Figure 6: General structure of the implementation.

4.2 Mesh class

The `Mesh` class takes advantage of the Gmsh API to get the parameters and to compute the data needed for the mesh. The `Mesh` constructor starts by saving the dimension, the type and properties of the elements. The integration type (ex: Gauss4), the jacobians, the shape functions and their derivative are also obtained thanks to Gmsh. The derivative of the shape functions is provided along the parametric directions. Their derivative along the physical directions are therefore computed thanks to composed derivative. The system can be expressed as

$$\underline{\underline{J}}^T \frac{d\underline{f}}{d\underline{x}} = \frac{d\underline{f}}{d\underline{u}},$$

where (x, y, z) are the physical coordinates and (u, v, w) are the parametric coordinates. As shown in the piece of code below, instead of transposing the J matrix, the implementation takes advantage of the fact that Gmsh gives row major arrays while Eigen/Lapack assume column major.

```
std::vector<double> jacobian(m_elDim*m_elDim);
m_elGradBasisFcts.resize(m_elNum*m_elNumNodes*m_elNumIntPts*3);

for(int el=0; el<m_elNum; ++el){
    for(int g=0; g<m_elNumIntPts; ++g)
         for(int f=0; f<m_elNumNodes; ++f){
             eigen::solve(jacobian.data(), &elGradBasisFct(el, g, f), m_elDim);
      }
   }
}
```

Listing 2: Derivatives of the shape functions along physical coordinates.

Next, the `Mesh` constructor gets the information related to the faces between the elements. Those are first created using `gmsh::model::mesh::getElementEdgeNodes` or `gmsh::model::mesh::getElementFaceNodes` depending on the dimension of the mesh. Nevertheless, some of the faces are counted twice because it exists common faces between the elements. The purpose of the following function (`Mesh::getUniqueFa ceNodeTags()`) is to remove the identical faces. The faces are first ordered for efficiency using C++14 functionality `std::sort` and then compared to each other.

```
void Mesh::getUniqueFaceNodeTags(){

    // Ordering per face for efficient comparison
    m_elFNodeTagsOrdered = m_elFNodeTags;

    for(int i=0; i<m_elFNodeTagsOrdered.size(); i+=m_fNumNodes)
        std::sort(m_elFNodeTagsOrdered.begin()+i, m_elFNodeTagsOrdered.
                  begin()+(i+m_fNumNodes));

    // Keep gmsh order while ordered array are used for comparison
    m_fNodeTags = m_elFNodeTags;
    m_fNodeTagsOrdered = m_elFNodeTagsOrdered;

    // Remove identical faces by comparing ordered arrays.
    std::vector<int>::iterator it_delete;
    std::vector<int>::iterator it_deleteUnordered;
    std::vector<int>::iterator it_unordered = m_fNodeTags.begin();

    for(std::vector<int>::iterator it_ordered = m_fNodeTagsOrdered.begin();
        it_ordered != m_fNodeTagsOrdered.end();){
        it_deleteUnordered = it_unordered+m_fNumNodes;

        for(it_delete=it_ordered+m_fNumNodes; it_delete != m_fNodeTagsOrdered.end();
                        it_delete+=m_fNumNodes){

            if(std::equal(it_ordered, it_ordered+m_fNumNodes, it_delete)){
                break;
            }
            it_deleteUnordered+=m_fNumNodes;
        }

        if(it_delete != m_fNodeTagsOrdered.end()){
            m_fNodeTagsOrdered.erase(it_delete, it_delete+m_fNumNodes);
            m_fNodeTags.erase(it_deleteUnordered, it_deleteUnordered+
            m_fNumNodes);
        }
```

```
        else {
            it_ordered+=m_fNumNodes;
            it_unordered+=m_fNumNodes;
        }
    }
}
```

Listing 3: Removal of identical faces.

Subsequently, a single entity containing all the unique faces is created. The Gmsh function `gmsh::model::mesh::setElementsByType` with empty face tags is called and the auto-generated tags are retrieved directly after.

The following step is to create the normal to each face. The normals are defined based on the gradient of the shape functions. This allows to define normals even for curved faces. Note that the normals are all normalized in the end of the loop.

```
std::vector<double> normal(m_Dim);

for(int f=0; f<m_fNum; ++f){
    for(int g=0; g<m_fNumIntPts; ++g){
        switch (m_fDim){
            case 0:{
                normal = {1, 0, 0};
                break;
            }
            case 1:{
                std::vector<double> normalPlane = {0, 0, -1};
                eigen::cross(&fGradBasisFct(f, g, 0), normalPlane.data(),
                             normal.data());

                if(eigen::dot(&fGradBasisFct(f, g), &fGradBasisFct(f, 0), m_Dim)<0){
                    for(int x=0; x<m_Dim; ++x){
                        normal[x] = -normal[x];
                    }
                }
                break;
            }
            case 2:{
                eigen::cross(&fGradBasisFct(f, g, 0), &fGradBasisFct(f, g, 1),
                             normal.data());

                if(g!=0 && eigen::dot(&fNormal(f,0), normal.data(), m_Dim)<0){
                    for(int x = 0; x<m_Dim; ++x){
                        normal[x] = -normal[x];
                    }
                }
                break;
            }
        }
        eigen::normalize(normal.data(), m_Dim);
        m_fNormals.insert(m_fNormals.end(), normal.begin(), normal.end());
    }
}
```

Listing 4: Definition of the normals.

Now that the elements and the faces are correctly introduced in the code, the faces have to be associated with their related elements. This part is facilitated by the fact that the node tags per face were previously ordered.

```
m_fNbrElIds.resize(m_fNum);

for(int el=0; el<m_elNum; ++el){
    for(int elF=0; elF<m_fNumPerEl; ++elF){
        for(int f=0; f<m_fNum; ++f){

            if(std::equal(&fNodeTagOrdered(f), &fNodeTagOrdered(f)+
               m_fNumNodes, &elFNodeTagOrdered(el, elF))){

                m_elFIds.push_back(f);
                m_fNbrElIds[f].push_back(el);
            }
        }
    }
}
```

Listing 5: Assignation of the faces to each element.

For efficiency purposes, the mapping between face node tags and the element node tags is stored. For example, the 3^{rd} node of the face corresponds to the 7^{th} of the element.

```
m_fNToElNIds.resize(m_fNum);

for(int f=0; f<m_fNum; ++f){
    for(int nf=0; nf<m_fNumNodes; ++nf){
        for(int el : m_fNbrElIds[f]){
            for(int nel=0; nel<m_elNumNodes; ++nel){

                if(fNodeTag(f, nf) == elNodeTag(el, nel)){
                    m_fNToElNIds[f].push_back(nel);
                }
            }
        }
    }
}
```

Listing 6: Assignation of the faces to each element.

Up to now, the normals are defined on each faces, but the orientation of the latter with respect to the neighbouring elements is not known. The following algorithm associates the value +1 when the normal goes out of the element and -1 otherwise.

```
double dotProduct;
std::vector<double> m_elBarycenters, fNodeCoord(3), elOuterDir(3), paramCoords;
gmsh::model::mesh::getBarycenters(m_elType[0], -1, false, true, m_elBarycenters);

for(int el=0; el<m_elNum; ++el){
    for(int f=0; f<m_fNumPerEl; ++f){
        dotProduct = 0.0;
        gmsh::model::mesh::getNode(elFNodeTag(el, f), fNodeCoord, paramCoords);

        for(int x=0; x<m_Dim; x++){
```

```
            elOuterDir[x] = fNodeCoord[x] - m_elBarycenters[el*3+x];
            dotProduct += elOuterDir[x]*fNormal(elFId(el, f), 0, x);
        }
        if(dotProduct >= 0){m_elFOrientation.push_back(1);}
        else{m_elFOrientation.push_back(-1);}
    }
}
```

Listing 7: Normal orientation.

Now that the orientation is saved, the elements are reclassified to impose that the first is the neighbouring element with the associated value +1.

```
int elf;
for(int f=0; f<m_fNum; ++f){
    for(int lf=0; lf<m_fNumPerEl; ++lf){
        if(elFId(fNbrElId(f, 0), lf) == f)
            elf = lf;
    }
    if(m_fNbrElIds.size() == 2){
        if(elFOrientation(fNbrElId(f, 0), elf) <= 0){
            std::swap(m_fNbrElIds[f][0], m_fNbrElIds[f][1]);

            for(int nf=0; nf<m_fNumNodes; ++nf)
                std::swap(fNToElNId(f, nf, 0), fNToElNId(f, nf, 1));
        }
    }
}
```

Listing 8: Reclassification of the neighbouring elements.

The code now iterates over the faces to find the boundaries of the mesh. The normals of the boundary faces are then oriented in the outward direction in order to facilitate the boundary conditions definition.

```
for(int f=0; f<m_fNum; ++f){
    if(m_fNbrElIds[f].size() < 2){
        m_fIsBoundary.push_back(true);

        for(int lf=0; lf<m_fNumPerEl; ++lf) {
            if(elFId(fNbrElId(f, 0), lf) == f){
                for(int g=0; g<m_fNumIntPts; ++g){
                    fNormal(f, g, 0) *= elFOrientation(fNbrElId(f, 0), lf);
                    fNormal(f, g, 1) *= elFOrientation(fNbrElId(f, 0), lf);
                    fNormal(f, g, 2) *= elFOrientation(fNbrElId(f, 0), lf);
                }
                elFOrientation(fNbrElId(f, 0), lf) = 1;
            }
        }
    }
    else {m_fIsBoundary.push_back(false);}
}
```

Listing 9: Detection of the boundary elements.

The physical entities defined as boundary in `.msh` file are then browsed as well as the nodes belonging to those entities. The associated faces can thus be retrieved and a unique integer can be assigned representing the type of boundary condition chosen by the user.

The value 1 is associated with the reflecting condition while the value 2 is for the absorbing one. Note that in the absence of assignation, the absorbing condition is chosen by default.

```
m_fBC.resize(m_fNum);
std::vector<int> nodeTags;
std::vector<double> coord;

for (auto const& physBC : config.physBCs){
    auto physTag = physBC.first;
    auto BCtype = physBC.second.first;
    auto BCvalue = physBC.second.second;
    gmsh::model::mesh::getNodesForPhysicalGroup(m_fDim, physTag, nodeTags, coord);

    if(BCtype == "Reflecting"){
        for(int f=0; f<m_fNum; ++f){
            if(m_fIsBoundary[f] && std::find(nodeTags.begin(), nodeTags.end(),
               fNodeTag(f)) != nodeTags.end()){
                m_fBC[f] = 1;
            }
        }
    }
    else{
        for(int f=0; f<m_fNum; ++f){
            if(m_fIsBoundary[f] && std::find(nodeTags.begin(), nodeTags.end(),
               fNodeTag(f)) != nodeTags.end()){
                m_fBC[f] = 0;
            }
        }
    }
}
```

Listing 10: Boundary condition assignment.

The final part of the `Mesh` constructor consists in the computation of the matrices related to the absorbing boundary condition in the specific context of acoustic waves. It is mainly used to suppress the outgoing solution along the characteristics lines.

```
RKR.resize(m_fNum*m_fNumIntPts);

for(int f=0; f<m_fNum; ++f){
    if(m_fBC[f] == 0){
        for(int g=0; g<m_fNumIntPts; ++g){

            int i = f*m_fNumIntPts+g;
            RKR[i].resize(16);
            RKR[i][0] = 0.25*config.c0;
            RKR[i][1] = 0.25*config.c0*config.c0*config.rho0*fNormal(f,g,0);
            RKR[i][2] = 0.25*config.c0*config.c0*config.rho0*fNormal(f,g,1);
            RKR[i][3] = 0.25*config.c0*config.c0*config.rho0*fNormal(f,g,2);

            RKR[i][4] = 0.25*fNormal(f,g,0)/config.rho0;
            RKR[i][5] = 0.25*config.c0*fNormal(f,g,0)*fNormal(f,g,0);
            RKR[i][6] = 0.25*config.c0*fNormal(f,g,0)*fNormal(f,g,1);
            RKR[i][7] = 0.25*config.c0*fNormal(f,g,0)*fNormal(f,g,2);

            RKR[i][8] = 0.25*fNormal(f,g,1)/config.rho0;
            RKR[i][9] = 0.25*config.c0*fNormal(f,g,1)*fNormal(f,g,0);
```

```
            RKR[i][10] = 0.25*config.c0*fNormal(f,g,1)*fNormal(f,g,1);
            RKR[i][11] = 0.25*config.c0*fNormal(f,g,1)*fNormal(f,g,2);

            RKR[i][12] = 0.25*fNormal(f,g,2)/config.rho0;
            RKR[i][13] = 0.25*config.c0*fNormal(f,g,2)*fNormal(f,g,0);
            RKR[i][14] = 0.25*config.c0*fNormal(f,g,2)*fNormal(f,g,1);
            RKR[i][15] = 0.25*config.c0*fNormal(f,g,2)*fNormal(f,g,2);
        }
    }
}
```

Listing 11: Matrices computation for the absorbing condition.

Outside the `mesh` constructor, the methods are dedicated to the assembly of the mass matrices, the stiffness matrices and their update at the element level.

4.3 Solver

As mentioned before, the `solver.cpp` contains the functions used to solve the acoustic system using the forward Euler order 1 method (`forwardEuler()`) or the Runge-Kutta order 4 method (`rungeKutta()`). Those two functions are essentially the same, only the time integration part differs.

The implementation starts with the memory allocation for the node numbers, the fluxes, the stiffness vectors and also for the solution (pressure, density and velocity). Right before going in the time iteration loop, the mass matrix is precomputed using `mesh.precomputeMassMatrix()` and the initial indices for the source are saved thanks to the parameters input by the user in the `.conf` file (size and position of the source).

```
std::vector<std::vector<int>> srcIndices;

for(int i=0; i<config.sources.size(); ++i){
    std::vector<int> indice;

    for(int n=0; n<mesh.getNumNodes(); n++){
        std::vector<double> coord, paramCoord;
        gmsh::model::mesh::getNode(mesh.getElNodeTags()[n], coord, paramCoord);

        if(pow(coord[0]-config.sources[i][1], 2) +
           pow(coord[1]-config.sources[i][2], 2) +
           pow(coord[2]-config.sources[i][3], 2) <
           pow(config.sources[i][4],2)){

                indice.push_back(n);
          }
    }
    srcIndices.push_back(indice);
}
```

Listing 12: Initial source indices.

Then, the time iteration is initiated with the saving of the solution and the message printing informing the user of the progress of the simulation. As it is commented in the code below, the Gmsh software uses a proper 2D array formatting (version 4.14) while the solution vector is previously stored as a continuous 1D vector.

A copy operation is therefore required.

```
if(tDisplay>=config.timeRate || step==0){
    tDisplay = 0;

    // [1] Copy solution to match GMSH format
    for(int el=0; el<mesh.getElNum(); ++el){
        for(int n=0; n<mesh.getElNumNodes(); ++n){
            int elN = el*elNumNodes+n;
            g_p[el][n] = u[0][elN];
            g_rho[el][n] = u[0][elN]/(config.c0*config.c0);
            g_v[el][3*n+0] = u[1][elN];
            g_v[el][3*n+1] = u[2][elN];
            g_v[el][3*n+2] = u[3][elN];
        }
    }
    gmsh::view::addModelData(gp_viewTag,step,g_names[0],"ElementNodeData",
                             elTags, g_p, t, 1);
    gmsh::view::addModelData(grho_viewTag, step, g_names[0], "ElementNodeData",
                             elTags, g_rho, t, 1);
    gmsh::view::addModelData(gv_viewTag, step, g_names[0], "ElementNodeData",
                             elTags, g_v, t, 3);

    // [2] Print and compute iteration time
    auto end = std::chrono::system_clock::now();
    auto elapsed = std::chrono::duration_cast<std::chrono::seconds>(end - start);
    gmsh::logger::write("["+std::to_string(t)+"/"+std::to_string(config.timeEnd)+"s]
                        Step number : "+std::to_string((int) step)+", Elapsed time: "
                        +std::to_string(elapsed.count())+"s");
}
```

Listing 13: Saving of the solution and message printing.

The next step of the solver implementation consists in updating the source. In the context of acoustic, the initial source is modulated by an amplitude and a sinus of the frequency chosen by the user (Listing 14).

```
for(int src=0; src<config.sources.size(); ++src){
     double amp = config.sources[src][5];
     double freq = config.sources[src][6];
     double phase = config.sources[src][7];
     double duration = config.sources[src][8];

     if(t<duration)
          for(int n=0; n<srcIndices[src].size(); ++n)
               u[0][srcIndices[src][n]] =  amp*sin(2*M_PI*freq*t+phase);
}
```

Listing 14: Source updating.

Finally, the first order forward Euler or fourth order Runge-Kutta algorithm is performed and the solution is stored in the output file.

5 Verification

5.1 Analytical solution in 1D

An important step before using the solver to illustrate physical phenomena is to validate this latter, first with an analytical solution, then by analysing the behaviour of the numerical scheme such as its convergence. By considering a one-dimensional problem along x, the system (22) becomes

$$\begin{cases} \dfrac{\partial p'}{\partial t} + v_0 \dfrac{\partial p'}{\partial x} + c_0^2 \rho_0 \dfrac{\partial v'}{\partial x} = 0, \\ \dfrac{\partial v}{\partial t} + \dfrac{1}{\rho} \dfrac{\partial p'}{\partial x} + v_0 \dfrac{\partial v'}{\partial x} = 0, \end{cases} \tag{24}$$

where v'_x and v_{0x} have been shortened v' and v_0 for readability reasons. The matrix A of the system (24) has two left eigenvectors $\vec{\varphi}$, $\vec{\psi}$ and two left eigenvalues λ_1, λ_2 given by

$$A = \begin{bmatrix} v_0 & c_0^2\rho_0 \\ 1/\rho & v_0 \end{bmatrix}, \qquad \begin{cases} \lambda_1 = v_0 + c_0, & \vec{\varphi} = \begin{bmatrix} 1/c_0\rho_0 & 1 \end{bmatrix}, \\ \lambda_2 = v_0 - c_0, & \vec{\psi} = \begin{bmatrix} -1/c_0\rho_0 & 1 \end{bmatrix}. \end{cases} \tag{25}$$

The system (24) can thus be rewritten as

$$\begin{cases} \dfrac{\partial}{\partial t}\left(\varphi_1 p' + \varphi_2 v'\right) + \lambda_1 \dfrac{\partial}{\partial x}\left(\varphi_1 p' + \varphi_2 v'\right) = 0, \\ \dfrac{\partial}{\partial t}\left(\psi_1 p' + \psi_2 v'\right) + \lambda_2 \dfrac{\partial}{\partial x}\left(\psi_1 p' + \psi_2 v'\right) = 0, \end{cases} \Leftrightarrow \begin{cases} \dfrac{\partial f}{\partial t}(x,t) + \lambda_1 \dfrac{\partial f}{\partial x}(x,t) = 0, \\ \dfrac{\partial g}{\partial t}(x,t) + \lambda_2 \dfrac{\partial g}{\partial x}(x,t) = 0, \end{cases}$$

where $f = (\varphi_1 p' + \varphi_2 v')$ and $g = (\psi_1 p' + \psi_2 v')$ form two uncoupled transport equations that can be solved independently according to the initial perturbations p'_0 and v'_0:

$$\begin{cases} f(x,t) = f_0(x - \lambda_1 t), \\ g(x,t) = g_0(x - \lambda_2 t), \end{cases} \Leftrightarrow \begin{cases} \varphi_1 p'(x,t) + \varphi_2 v(x,t) = \varphi_1 p'_0(x - \lambda_1 t) + \varphi_2 v'_0(x - \lambda_1 t), \\ \psi_1 p'(x,t) + \psi_2 v'(x,t) = \psi_1 p'_0(x - \lambda_2 t) + \psi_2 v'_0(x - \lambda_2 t). \end{cases}$$

Injecting (25) leads to the solution

$$\begin{cases} p'(x,t) = \dfrac{p'_0(x - \lambda_1 t) + p'_0(x - \lambda_2 t)}{2} + \dfrac{\left[v'_0(x - \lambda_1 t) - v'_0(x - \lambda_2 t)\right] c_0\rho_0}{2}, \\ v'(x,t) = \dfrac{p'_0(x - \lambda_1 t) - p'_0(x - \lambda_2 t)}{2c_0\rho_0} + \dfrac{v'_0(x - \lambda_1 t) + v'_0(x - \lambda_2 t)}{2}. \end{cases}$$

A comparison between the solution generated by the finite element code and the analytical derivation is displayed in Figures 7 and 8.

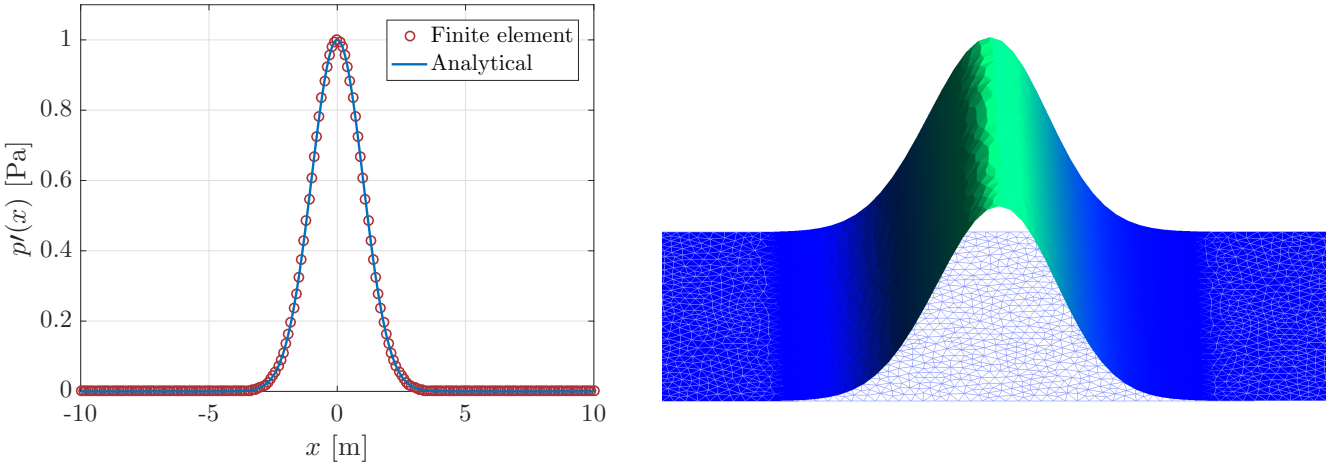


Figure 7: Analytical and numerical solutions of a 1D wave at $t = 0$ [s] for $\rho_0 = 1$ [kg/m^3], $c_0 = 5$ [m/s], $v_0 = 1$ [m/s] and a Gaussian shaped p_0, with T6 elements.

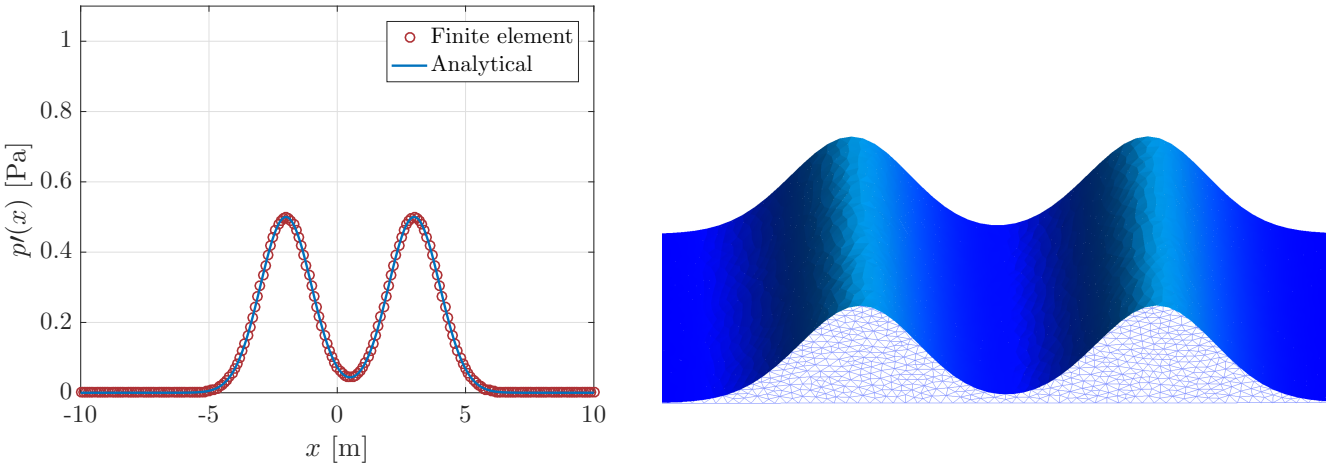


Figure 8: Analytical and numerical solutions of a 1D wave at $t = 0.5$ [s] for $\rho_0 = 1$ [kg/m^3], $c_0 = 5$ [m/s], $v_0 = 1$ [m/s] and a Gaussian shaped p_0, with T6 elements.

In addition to the results presented previously, the convergence of the method can be highlighted by computing an error (Figures 9 and 10), where p' is the analytical solution and $\hat{p}'$ the numerical one

$$\text{error} = \int_\Omega |\, p'(x) - \hat{p}'(x) \,|^2 \, d\Omega \simeq \sum_g w_g \, |\, p'(x_g, y_g) - \hat{p}'(x_g, y_g) \,|^2 \det \mathbf{J}_g.$$

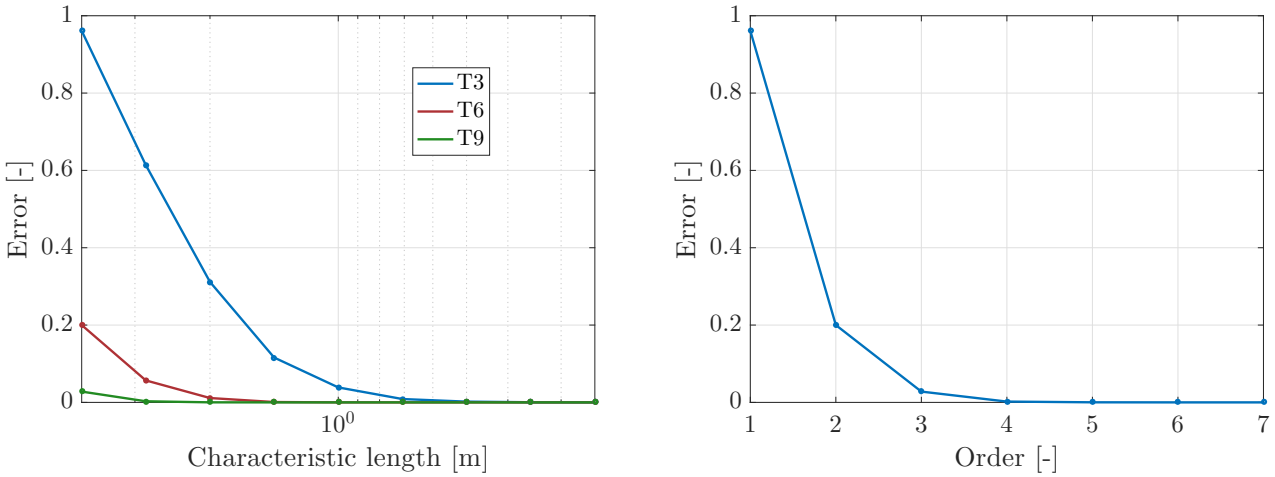


Figure 9: Error on p' at time $t = 1.5$ [s] for $\vec{v}_0 = [1, 0, 0]$ [m/s], $c_0 = 5$ [m/s], $\rho_0 = 1$ [kg/m^3] and Gaussian initial pressure field along $\vec{e}_x$. The left graph displays the error according to the number of elements in the mesh through their characteristic length and the right graph displays the error according to the order of elements while keeping the same mesh density.

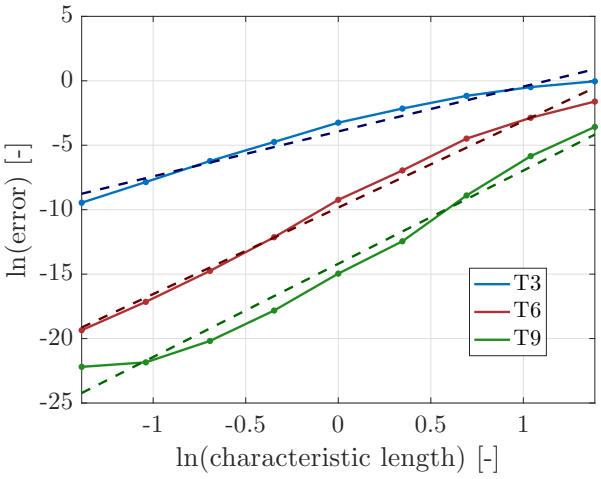


Figure 10: Logarithmic plot of the error according to the characteristic length L of the mesh. The linear relation $\ln(\text{error}) \simeq \alpha \ln(L)$ leads to $\text{error}(L) \simeq L^\alpha$ where $\alpha \simeq 3.5,\ 6.7,\ 7.2$ [-] is the order of convergence for T3, T6 and T9 elements respectively.

5.2 Analytical solution in 2D

By considering a two-dimensional problem along $(\vec{e}_x, \vec{e}_y)$ and taking a mean velocity field $\vec{v}_0 = \vec{0}$, the system (22) becomes

$$\begin{cases} \dfrac{\partial p'}{\partial t} = c_0^2 \rho_0 \left(\dfrac{\partial v'_x}{\partial x} + \dfrac{\partial v'_y}{\partial y} \right), \\ \dfrac{\partial v'_x}{\partial t} = \dfrac{1}{\rho_0} \dfrac{\partial p'}{\partial x}, \\ \dfrac{\partial v'_y}{\partial t} = \dfrac{1}{\rho_0} \dfrac{\partial p'}{\partial y}. \end{cases} \tag{26}$$

Differentiating the first equation of (26) with respect to t leads to a common formulation of a wave equation:

$$\begin{aligned}\frac{\partial^2 p'}{\partial t^2} &= c_0^2\rho_0\left(\frac{\partial}{\partial t}\frac{\partial v'_x}{\partial x}+\frac{\partial}{\partial t}\frac{\partial v'_y}{\partial y}\right) = c_0^2\rho_0\left(\frac{\partial}{\partial x}\frac{\partial v'_x}{\partial t}+\frac{\partial}{\partial y}\frac{\partial v'_y}{\partial t}\right)\\ &= c_0^2\rho_0\left[\frac{\partial}{\partial x}\left(\frac{1}{\rho_0}\frac{\partial p'}{\partial x}\right)+\frac{\partial}{\partial y}\left(\frac{1}{\rho_0}\frac{\partial p'}{\partial y}\right)\right] = c_0^2\left(\frac{\partial^2 p'}{\partial x^2}+\frac{\partial^2 p'}{\partial y^2}\right),\end{aligned} \tag{27}$$

A truncated solution can be obtained by separation of variables and superposition principle on a simple squared domain $[0,a]\times[0,b]$ with the following boundary and initial conditions:

$$\begin{aligned} &p'(0,y,t) = p'(a,y,t) = 0 && \text{on } \{x=0, x=a\}\times[0,b]\times[0,\infty],\\ &p'(x,0,t) = p'(x,b,t) = 0 && \text{on } [0,a]\times\{y=0, y=b\}\times[0,\infty],\end{aligned} \tag{28}$$

$$\begin{aligned} &p'(x,y,0) = p'_0(x,y) && \text{in } [0,a]\times[0,b]\times\{t=0\},\\ &\frac{\partial p'}{\partial t}(x,y,0) = 0 && \text{in } [0,a]\times[0,b]\times\{t=0\}.\end{aligned} \tag{29}$$

We will firstly look for a solution of the form $p' = X(x)Y(y)T(t)$, injecting it into (27) gives

$$XY\frac{d^2T}{dt^2} = c_0^2\left(\frac{d^2X}{dx^2}YT + X\frac{d^2Y}{dy^2}T\right) \quad\Leftrightarrow\quad \frac{d^2T/dt^2}{c_0^2T} = \frac{d^2X/dx^2}{X}+\frac{d^2Y/dy^2}{Y}.$$

Since the two sides are functions of different independent variables, they must be constant.

$$\begin{cases}\frac{d^2T/dt^2}{c_0^2T} = C_1\\ \frac{d^2X/dx^2}{X} = C_1 - \frac{d^2Y/dy^2}{Y}\end{cases} \Leftrightarrow \begin{cases}\frac{d^2T/dt^2}{c_0^2T} = C_1\\ \frac{d^2X/dx^2}{X} = C_2\\ C_1 - \frac{d^2Y/dy^2}{Y} = C_2\end{cases} \Leftrightarrow \begin{cases}\frac{d^2T}{dt^2} - C_1c_0^2T = 0,\\ \frac{d^2X}{dx^2} - C_2X = 0,\\ \frac{d^2Y}{dy^2} - C_3Y = 0,\end{cases}$$

where $C_3 = C_1 - C_2$. In the same way, injecting the solution into the boundary conditions (28) leads to particular conditions on X and Y for having a non-trivial solution:

$$\begin{cases}X(0)Y(y)T(t) = X(a)Y(y)T(t) = 0\\ X(x)Y(0)T(t) = X(x)Y(b)T(t) = 0\end{cases} \Rightarrow \begin{cases}X(0) = X(a) = 0,\\ Y(0) = Y(b) = 0.\end{cases}$$

The non-trivial solutions to the boundary value problems for X and Y are thus

$$\begin{aligned} X_m(x) &= \sin\left(\sqrt{-C_2}ax\right), && C_2 = \frac{-m^2\pi^2}{a^2}, && m = 1,2,3,...\\ Y_n(y) &= \sin\left(\sqrt{-C_3}bx\right), && C_3 = \frac{-n^2\pi^2}{b^2}, && n = 1,2,3,...\end{aligned}$$

Recall that $C_1 = C_2 + C_3$, a general solution for T is obtained as

$$\frac{d^2T}{dt^2} - C_1c_0^2T = 0 \quad\Leftrightarrow\quad T_{mn}(t) = B_1\cos\left(\sqrt{-C_1}c_0t\right) + B_2\sin\left(\sqrt{-C_1}c_0t\right).$$

Assembling our results, for any pair $m,n\geq 1$, the normal mode $p'_{mn} = X_mY_nT_{mn}$ is

$$p'_{mn} = \sin\frac{m\pi x}{a}\sin\frac{n\pi y}{b}\left[B_1\cos\left(\pi\sqrt{\frac{m^2}{a^2}+\frac{n^2}{b^2}}c_0t\right) + B_2\sin\left(\pi\sqrt{\frac{m^2}{a^2}+\frac{n^2}{b^2}}c_0t\right)\right].$$

The general solution is obtained according to the principle of superposition:

$$p' = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty} \sin\frac{m\pi x}{a}\sin\frac{n\pi y}{b}\left[B_1\cos\left(\pi\sqrt{\frac{m^2}{a^2}+\frac{n^2}{b^2}}c_0 t\right) + B_2\sin\left(\pi\sqrt{\frac{m^2}{a^2}+\frac{n^2}{b^2}}c_0 t\right)\right]$$

and satisfies the boundary conditions (28). Finally, we must determine the values of the coefficients B_1 and B_2 that are required to satisfy the initial conditions (29), assuming the following decomposition in double Fourier series:

$$p'(x,y,0) = p'_0(x,y) = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty} B_1 \sin\frac{m\pi x}{a}\sin\frac{n\pi y}{b}$$

$$\frac{\partial p'}{\partial t}(x,y,0) = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty},\left(\pi\sqrt{\frac{m^2}{a^2}+\frac{n^2}{b^2}}c_0 t\right) B_2 \sin\frac{m\pi x}{a}\sin\frac{n\pi y}{b} = 0 \quad \Leftrightarrow \quad B_2 = 0.$$

Since the functions Z_{mn} are pairwise orthogonal relative to the inner product $\langle\,\cdot,\cdot\,\rangle$

$$Z_{mn} = \int_0^b\int_0^a \sin\frac{m\pi x}{a}\sin\frac{n\pi y}{b}\,dxdy \qquad \forall\, m,n\in\mathbb{N},$$

the coefficient B_1 can be computed as

$$B_1 = \frac{\langle\, p'_0, Z_{mn}\,\rangle}{\langle\, Z_{mn}, Z_{mn}\,\rangle} = \frac{\int_0^b\int_0^a p'_0(x,y)\sin\frac{m\pi x}{a}\sin\frac{n\pi y}{b}\,dxdy}{\int_0^b\int_0^a \sin^2\frac{m\pi x}{a}\sin^2\frac{n\pi y}{b}\,dxdy}$$
$$= \frac{4}{ab}\int_0^b\int_0^a p'_0(x,y)\sin\frac{m\pi x}{a}\sin\frac{n\pi y}{b}\,dxdy.$$

The effect of the boundary conditions is expected to be negligible far from the borders of the domain. A comparison between the analytical and numerical solutions with an initial Gaussian pressure field is presented in Figure 11.

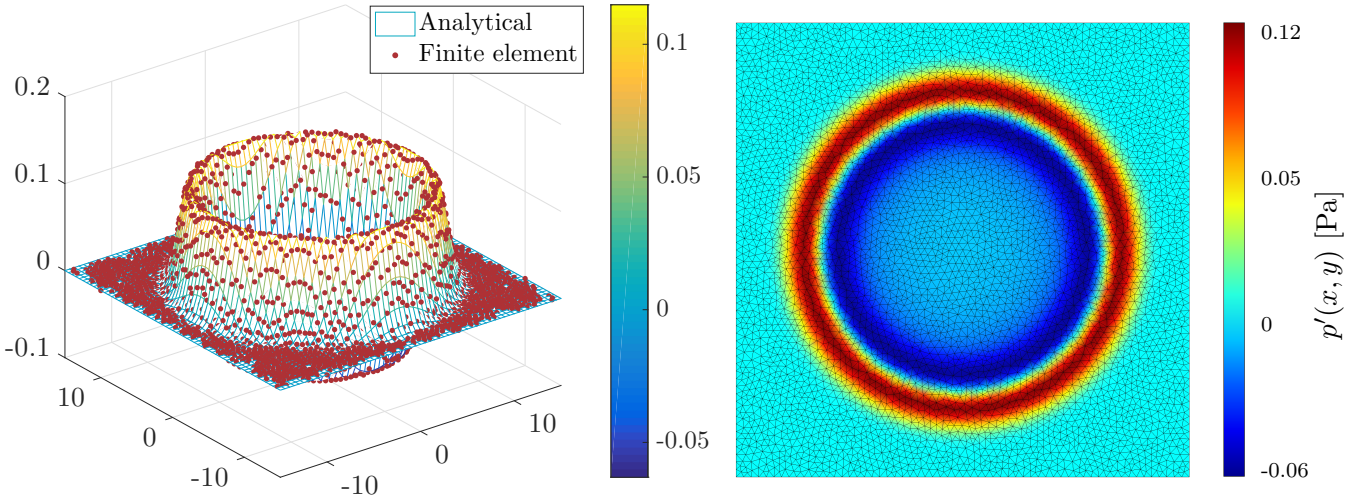


Figure 11: Comparison between the analytical pressure field (on the left) for domain $[0,30]\times[0,30]$ [m^2], then translated to the origin, and the numerical simulation (on the right) at $t = 2$ [s] with $\vec{v}_0 = \vec{0}$ [m/s], $c_0 = 5$ [m/s], $\rho_0 = 1$ [kg/m^3].

The convergence of the method is displayed in Figures 12 and 13 by computing the error between the analytical solution p' and the numerical one $\hat{p}'$.

$$\text{error} = \int_\Omega |\, p'(x,y) - \hat{p}\,'(x,y) \,|^2 \, d\Omega \simeq \sum_g w_g \,|\, p'(x_g,y_g) - \hat{p}\,'(x_g,y_g) \,|^2 \det \mathbf{J}_g.$$

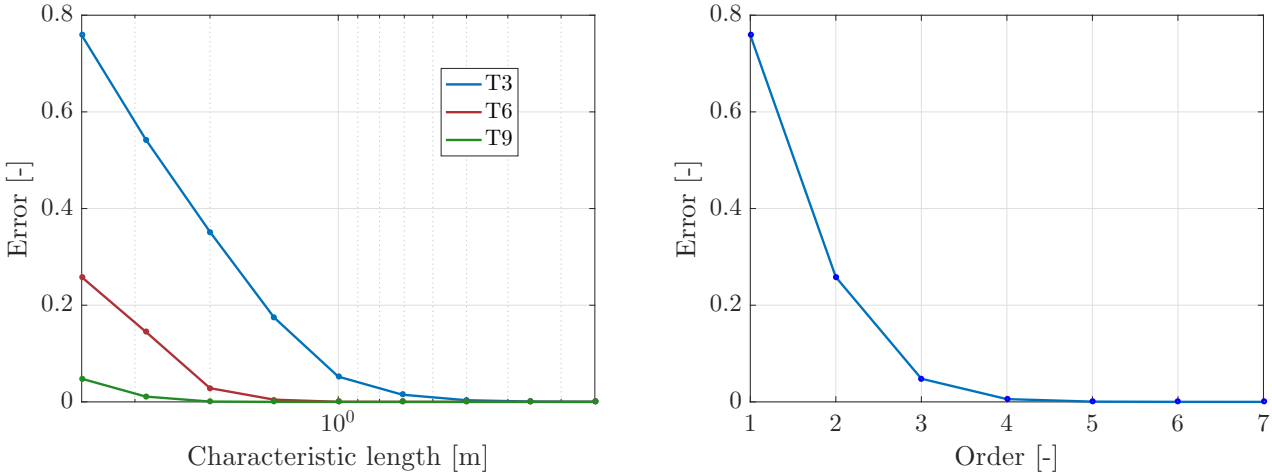


Figure 12: Error on p' at time $t = 2$ [s] for $\vec{v}_0 = [0, 0, 0]$ [m/s], $c_0 = 5$ [m/s], $\rho_0 = 1$ [kg/m^3] and Gaussian initial pressure field along $(\vec{e}_x, \vec{e}_y)$. The left graph displays the error according to the number of elements in the mesh through their characteristic length and right graph displays the error according to the order of the elements while keeping the same mesh density.

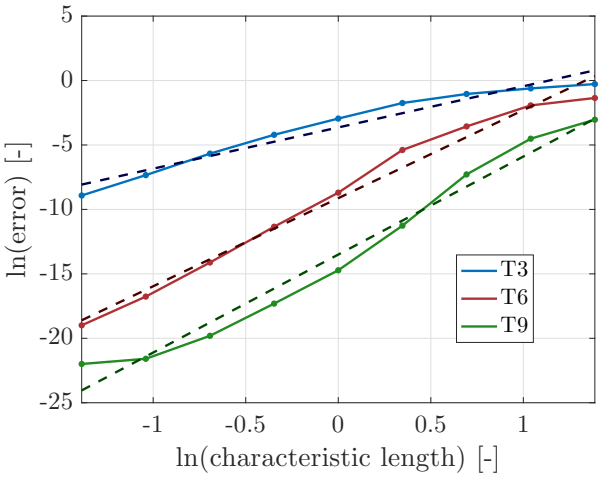


Figure 13: Logarithmic plot of the error according to the characteristic length L of the mesh. The linear relation $\ln(\text{error}) \simeq \alpha \ln(L)$ leads to $\text{error}(L) \simeq L^\alpha$ where $\alpha \simeq 3.2, 6.8, 7.6$ [-] is the order of convergence for T3, T6 and T9 respectively.

5.3 Stability

A general formulation of the Courant–Friedrichs–Lewy condition for a n-dimensional case is given by

$$\Delta t \leq C \sum_{i=1}^{n} \frac{\Delta x_i}{c_i}, \tag{30}$$

where Δx_i and c_i are respectively the distance between two integration points and the magnitude of the velocity along the direction i, Δt is the time discretization step and C is the Courant number that depends on the numerical scheme.

If $\vec{v}_0 = \vec{0}$, the velocity of the wave is c_0 and by taking a regular n-dimensional grid of first order elements with uniform spacing $\Delta x_i = \Delta x$ (Figures 14 and 15), the condition (30) becomes

$$\Delta t \leq C \frac{\Delta x}{n c_0},$$

For higher order polynomials, the minimum distance Δx between two nodes is smaller than the characteristic length of an element. Using regular second order elements, one has $\Delta x = L/2$ (Figures 16 and 17).

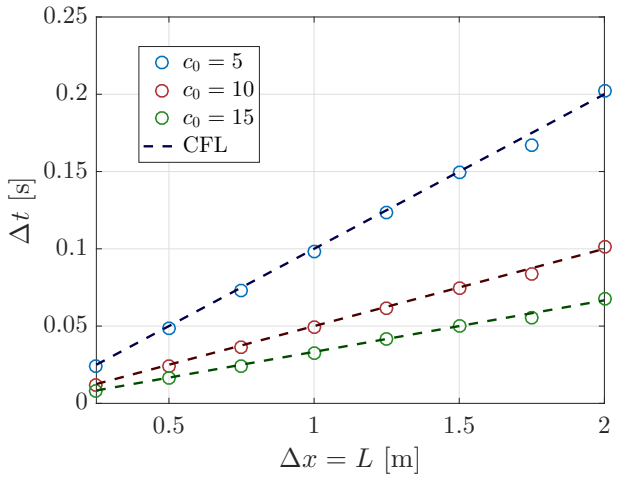


Figure 14: Maximum Δt for a stable numerical scheme according to L on a regular 1D mesh ($n = 1$), with $C = 1/2$, $\vec{v}_0 = \vec{0}$ [m/s] and first order elements.

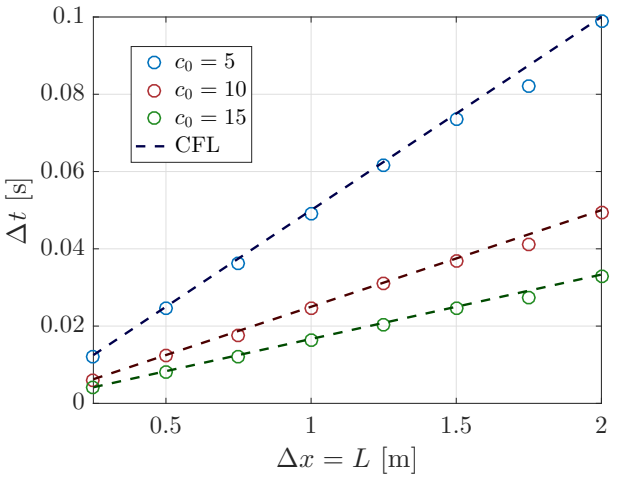


Figure 15: Maximum Δt for a stable numerical scheme according to L on a regular 2D mesh ($n = 2$), with $C = 1/2$, $\vec{v}_0 = \vec{0}$ [m/s] and first order elements.

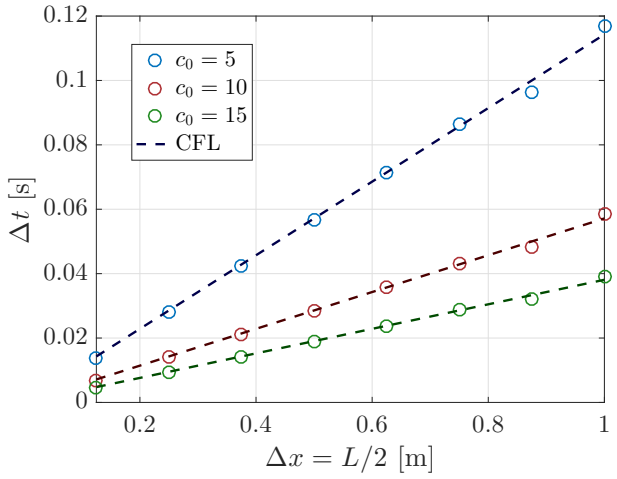


Figure 16: Maximum Δt for a stable numerical scheme according to L on a regular 1D mesh ($n = 1$), with $C = 1/1.75$, $\vec{v}_0 = \vec{0}$ [m/s] and second order elements.

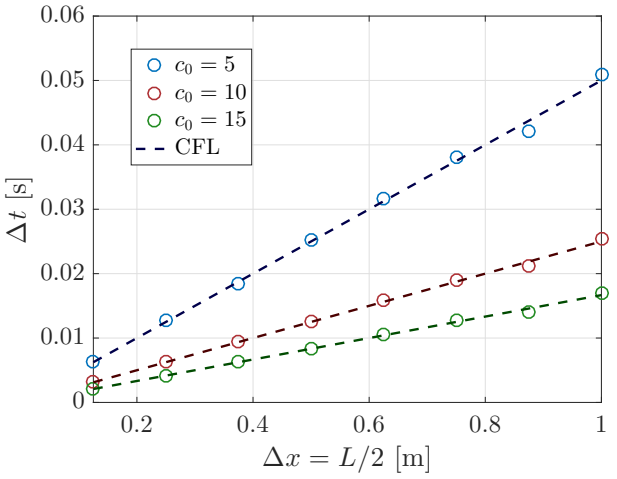


Figure 17: Maximum Δt for a stable numerical scheme according to L on a regular 2D mesh ($n = 2$), with $C = 1/2$, $\vec{v}_0 = \vec{0}$ [m/s] and second order elements.

For the 2D case, the results presented above exploit the regularity of squared elements, when the mesh is unstructured or when using triangular elements, the spacing between the nodes is not uniform and the CFL condition will depend on the shortest distance between these nodes in the direction of the flow. For general T3, a first approximation of CFL condition may be considered by taking Δx as the radius of the inscribed circle in the smallest element of the mesh. The CFL condition is thus necessary but not always sufficient to ensure stability.

5.4 Performance and flops

Finally before presenting the first results, we are going to asses our algorithm performances. We use a reference disk shaped mesh containing 1,063 elements. The test are run on a processor Intel xeon e3-1230v5 3.2 Ghz, 4 cores, 8 threads. The theoretical peak Gflops is calculated as:

$$\begin{aligned}\text{Peak Gflops} = &(\text{CPU speed in GHz}) \times (\text{number of CPU cores}) \times \\ &(\text{floating point numbers in one SIMD register}) \times \\ &(\text{nbr of SIMD operands}) \times (\text{SIMD operands per CPU cycle}).\end{aligned}$$

In particular, Intel Xeon Scalable Processors support AVX2, 2FMA (multiply and add). We therefore get:

$$\text{Peak Gflops} = 3.2 \times 4 \times 6 = 76.8 \text{ Gflops}.$$

The previous computation is in accordance with the export compliance metrics provided by Intel. We hereafter present the performances of our implementation of the DGFEM and by the same occasion we present the openMP scaling.

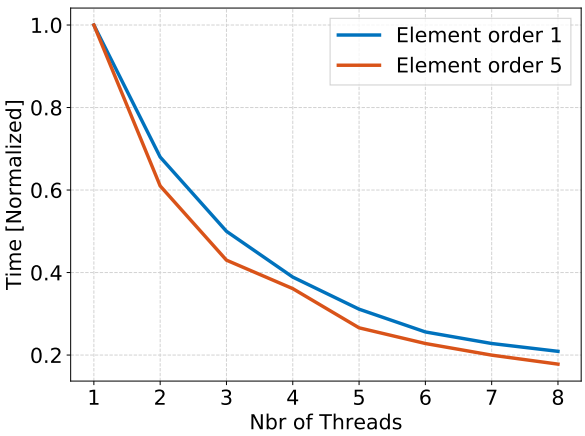


Figure 18: OpenMP scaling as function of the number of threads. The scaling is displayed for first and fifth order elements.

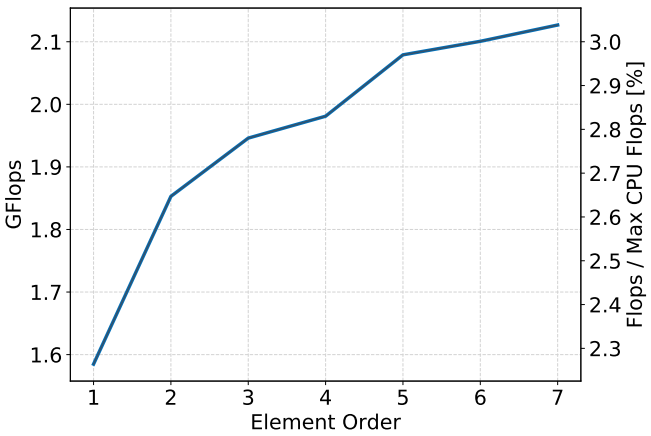


Figure 19: Floating point operations per second (Flops). The only operations accounted for are $+, -, /, \times$.

First, by looking at the openMP scaling we see that the doubling the number of threads reduce the computation time of 35%. This is a satisfactory result, although less than optimal as the discontinuous finite elements are particularly well suited for multithreading. Secondly, our implementation achieve an average of 2.6% of the theoretical maximum number of floating-point operations. This result highlights the weakness of the implementation which must be considered as a first approach to discontinuous finite element method. To give an order of magnitude, our code is about 10 to 20 times slower than production stage solver.

6 Results

6.1 2D cases

An initial perturbation of the pressure field in a circular domain of radius $r = 300$ [m] is computed in this first example. The considered environment is air at 25°C which has a density $\rho_0 = 1.1839$ [kg/m^3] and admits a speed of sound $c_0 = 346$ [m/s]. The initial perturbation is a Gaussian centered at $(x_0 = 0, y_0 = 50)$ [m] with a standard deviation $\sigma = 12$ [Pa] and an amplitude of $A = 1$ [Pa]:

$$p'_0(x, y) = A \exp\left(\frac{(x - x_0)^2 + (y - y_0)^2}{2\sigma^2}\right).$$

The domain is composed of T6 elements and an absorbing condition is attributed to the boundaries of the domain while a reflecting condition is attributed to the holes. The absorbing condition require a zero mean velocity field (Figures 20, 21, 22 and 23).

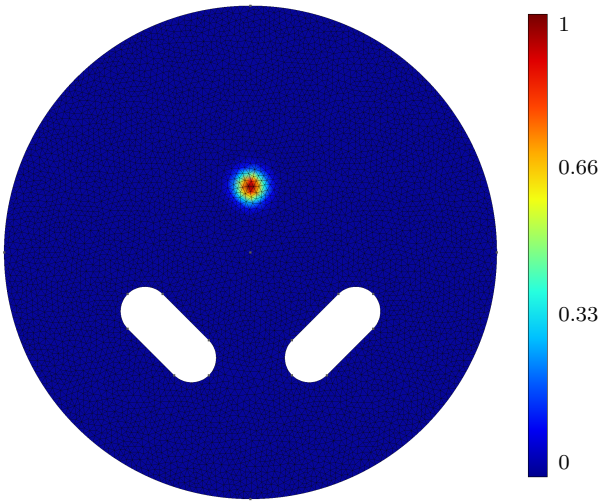


Figure 20: p' field in [Pa] at $t = 0$ [s].

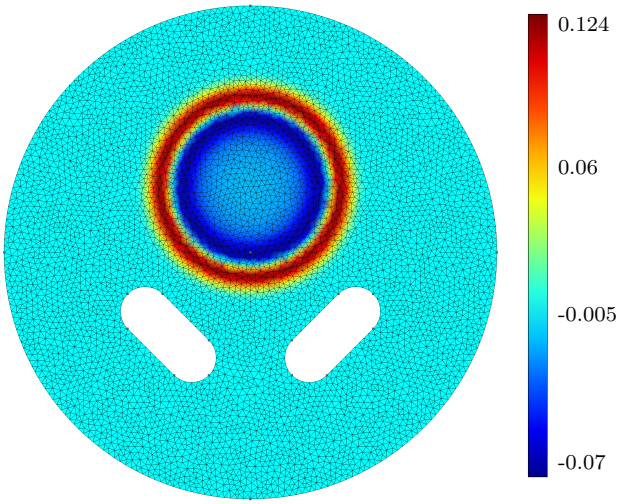


Figure 21: p' field in [Pa] at $t = 0.2$ [s].

The next simulation (Figures 24, 25, 26 and 27) displays the behavior of a centered dipole source oscillating at a frequency 1 [Hz] and amplitude 1 [Pa] for a mean velocity field $v_{0x} = -100$ [m/s] and $v_{0y} = 150$ [m/s]. A reflecting condition is assigned to the surface of the holes and the border of the domain. The mesh is composed of T6 elements.

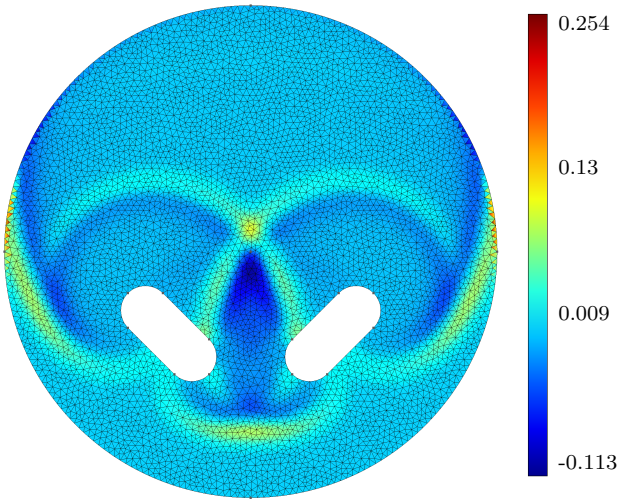


Figure 22: p' field in [Pa] at $t = 0.85$ [s].

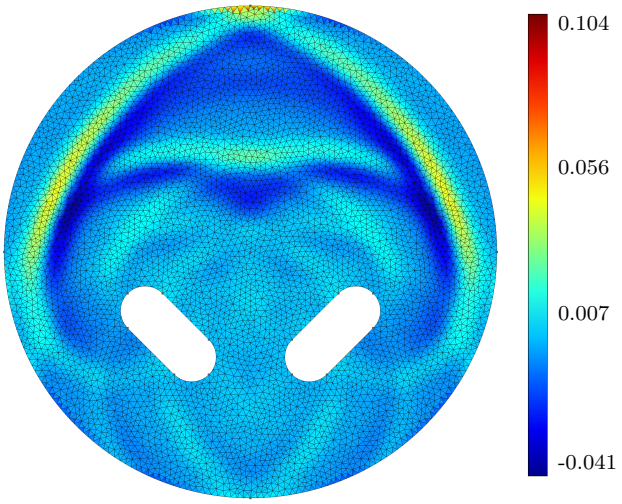


Figure 23: p' field in [Pa] at $t = 1.5$ [s].

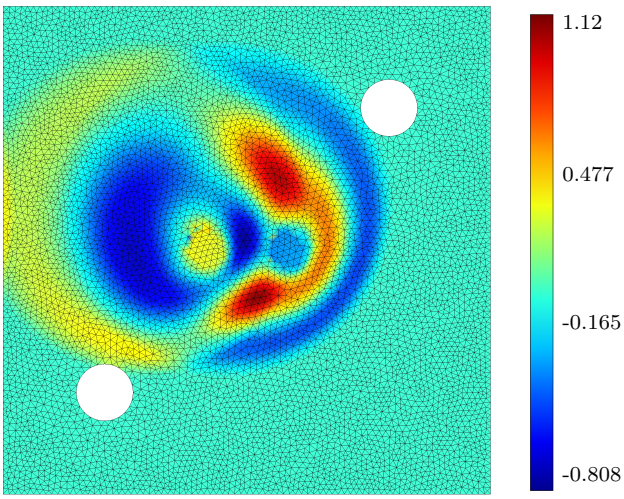


Figure 24: p' field in [Pa] at $t = 1$ [s].

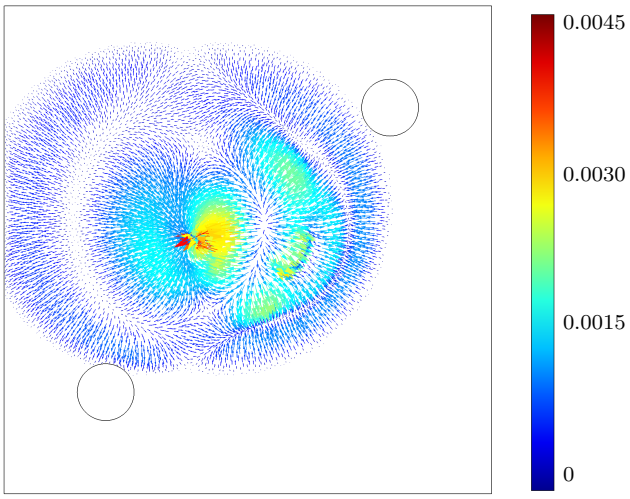


Figure 25: $\vec{v}'$ field in [m/s] at $t = 1$ [s].

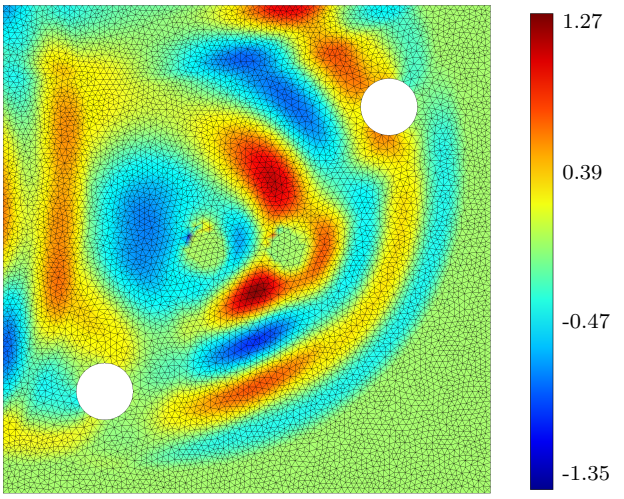


Figure 26: p' field in [Pa] at $t = 2$ [s].

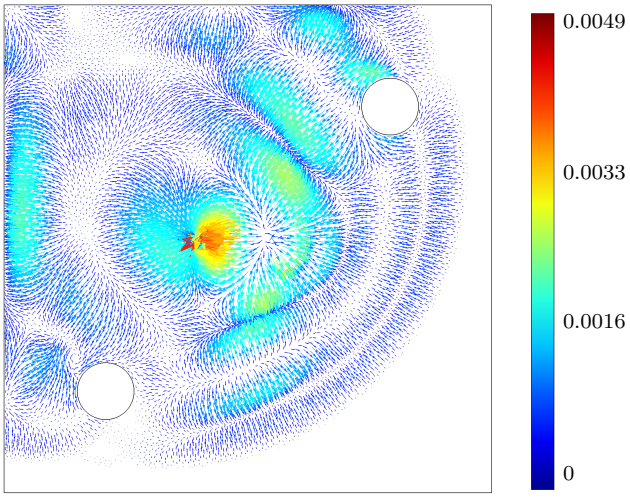


Figure 27: $\vec{v}'$ field in [m/s] at $t = 2$ [s].

6.1.1 Room acoustics

Acoustic simulations in a 2-dimensional floor with multiple rooms and obstacles. The dimensions of the floor are 10×6 [m]. The simulation takes place in ambient conditions: 20°C, air density $\rho_0 = 1.225$ [kg/m^3] and speed of sound $c_0 = 343$ [m/s]. The mesh is made of $11,662$ elements (order 3) corresponding to $53,467$ nodes and $17,866$ faces. The walls and obstacles are designed to ensure 5% reflection which typically corresponds to rigid surface such as painted walls or wood.

The acoustic of a such rooms can be assessed in multiple ways depending on the objective. One of the most common technique is the computation of the room impulse response. Any complex signal can then be convoluted with the impulse response to reproduce the room acoustic. A second approach is the direct simulation of the wave source, this approach is obviously much more costly but fits greatly to our small 2-dimensional mesh. It also provides a real time visualization of the propagation as illustrated in the figures 28 to 31 for a frequency source of 1500 [Hz].

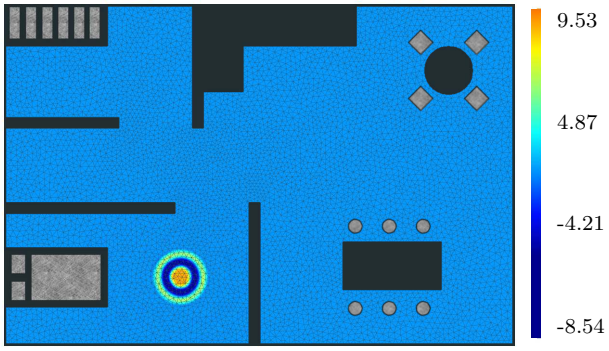


Figure 28: p' field [Pa] at $t = 0.001$ [s].

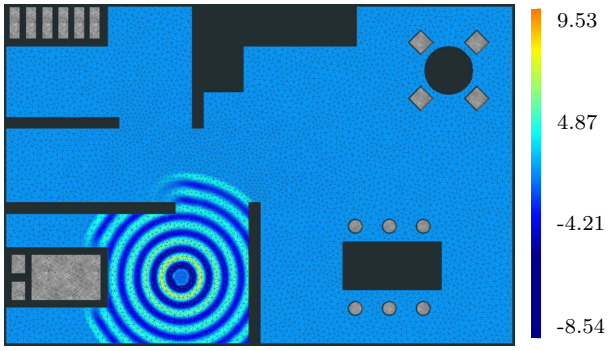


Figure 29: p' field [Pa] at $t = 0.005$ [s].

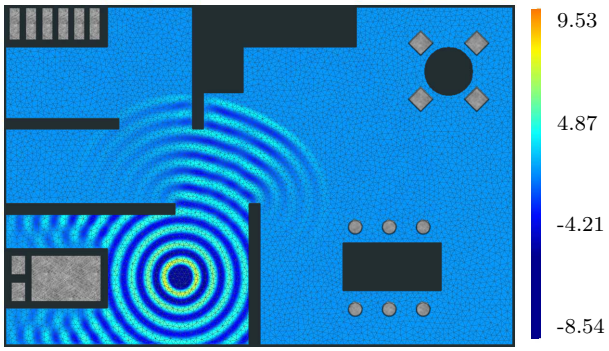


Figure 30: p' field [Pa] at $t = 0.01$ [s].

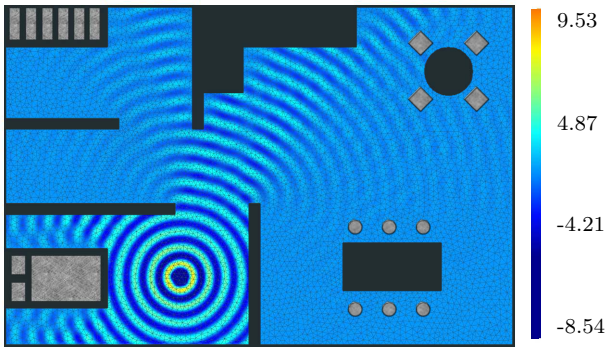


Figure 31: p' field [Pa] at $t = 0.025$ [s].

To provide a complete and realistic simulation, we decided to use a real music source as emitter. We choose the piano prelude in C major (BWV 846) from Johann Sebastian Bach. We limited our simulation to the first 30 seconds. The propagation of such music piece can however become extremely time consuming as it requires to propagate waves with frequency ranging from 20 to $10,000$ [Hz]. We therefore decided to restrict the emitter frequency range to 3000 [Hz] by applying a low pass filter.

This restriction has minimal effect on the sound quality. Finally, the time step is chosen to match the source rate: i.e $\Delta t = 1/44100 = 4.53 \cdot 10^{-5}$ [s].

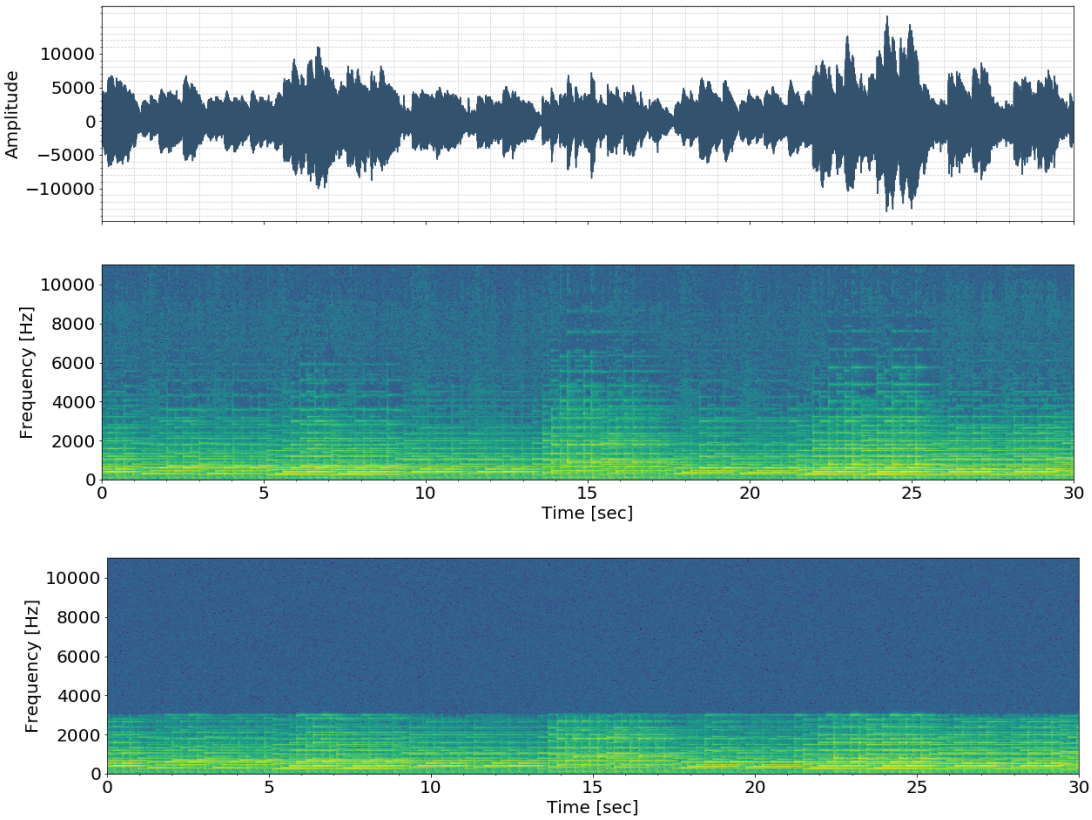
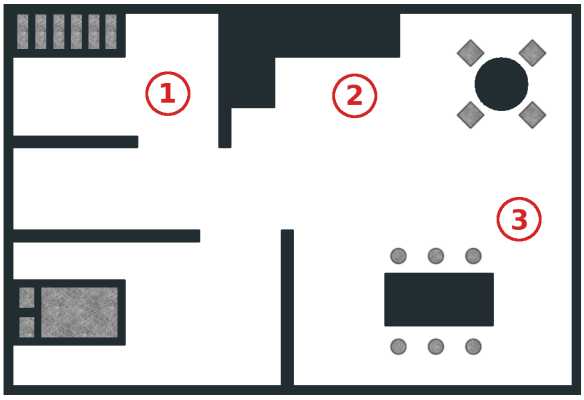


Figure 32: First 30 seconds of the prelude in C major (BWV 846) from Johann Sebastian Bach. The second histogram shows the effect of the low pass filter on the emitted frequency range. **Listen:** `https://github.com/pvanberg/DGFEM-Acoustic/blob/master/assets/bach_emitter.mp3`

The sound is then recorded at different positions to demonstrate the sound distortion and the effects of the room geometry. The received sound still present two issues. First, it is highly attenuated with the distance and secondly it appears noisy. We therefore simulated the receiver electronics by an amplifier and a low pass filter to remove excessive high frequency noise.



The recorded locations are:

1. Upper left room: $x = 3$, $y = 5$ [m]. **Listen:** bach_receiver_1.mp3
2. Top center: $x = 5.8$, $y = 5$ [m]. **Listen:** bach_receiver_2.mp3
3. Center right: $x = 9$, $y = 2.7$ [m]. **Listen:** bach_receiver_1.mp3

6.2 3D cases

6.2.1 Introduction

Our DGFEM code has also been extended to 3-dimensional cases and supports high order elements. In particular, it deals efficiently with complex and curved geometries by taking advantage of the Gmsh api. As illustration, the element normals evaluated at the integration points are a great visualization spotlight.

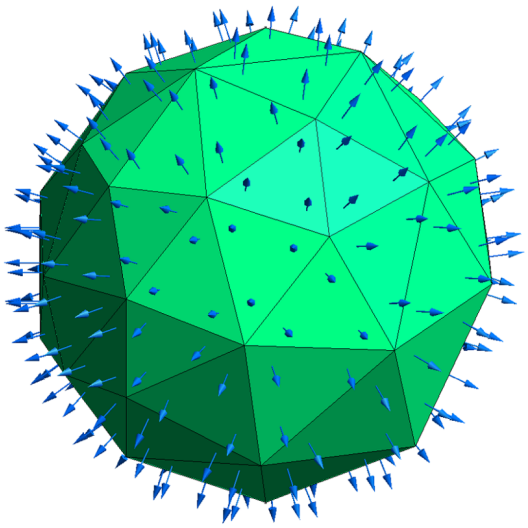


Figure 33: First order element normals evaluated at the integration points over a sphere of unit radius.

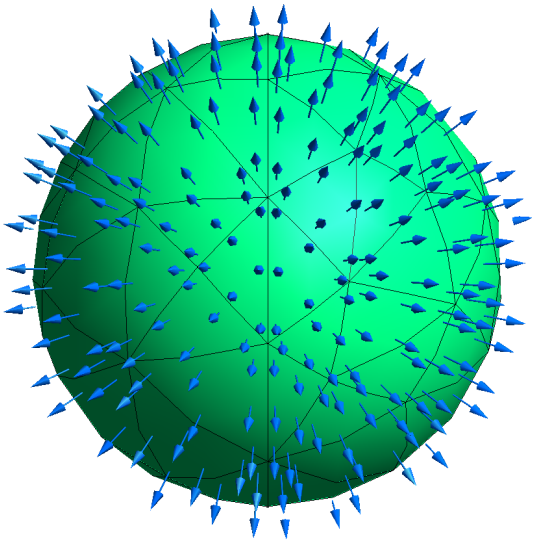


Figure 34: Second order element normals evaluated at the integration points over a sphere of unit radius.

Our 3D implementation is fully integrated with the lower dimension cases and do not require specific actions. It however important to stay attentive to the fact that our code does not yet support Hexahedron meshes and thus is restricted to Tetrahedron meshes. Consequently, hybrid meshes are also not supported. It is the main difference with the 2D-cases where both element types are supported.

6.2.2 Gaussian propagation

The first simulation is dedicated to the propagation of a given initial condition. We imposed a Gaussian at the mesh center $(0, 0, 0)$, with unit intensity and unit RMS width. This simple test case, is used to highlight the propagation of acoustic waves in a 3D medium and asses the algorithm accuracy.

The propagation is simulated in a cubic region of $20 \times 20 \times 20$ [m], filled with air at 20°C, with a density of $\rho_0 = 1.225$ [kg/m^3] and a corresponding speed of sound $c_0 = 343$ [m/s]. The space is meshed with $4,587$ nodes and $20,983$ elements corresponding to $43,562$ faces. The simulation is run with a step size of 10^{-4} [s] over $1,000$ iterations for a total computational time of 193sec using 4 threads (intel xeon e3-1230v5).

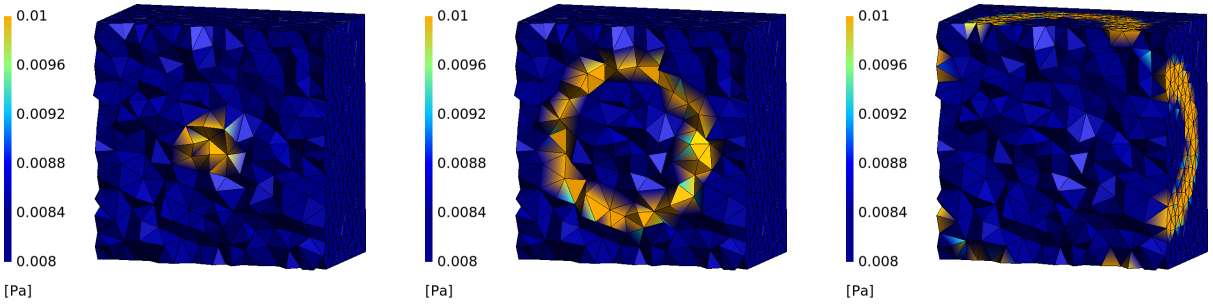


Figure 35: Pressure wave propagation in 3D homogeneous medium using first order element. The colorbar indicates the absolute pressure. The snapshots are taken at respectively 0.001, 0.015 and 0.030 [s].

The simulation is then reproduced with order 3 elements to compare with the first order simulation. Local refinement is also used to express a higher degree of approximation of the shape functions.

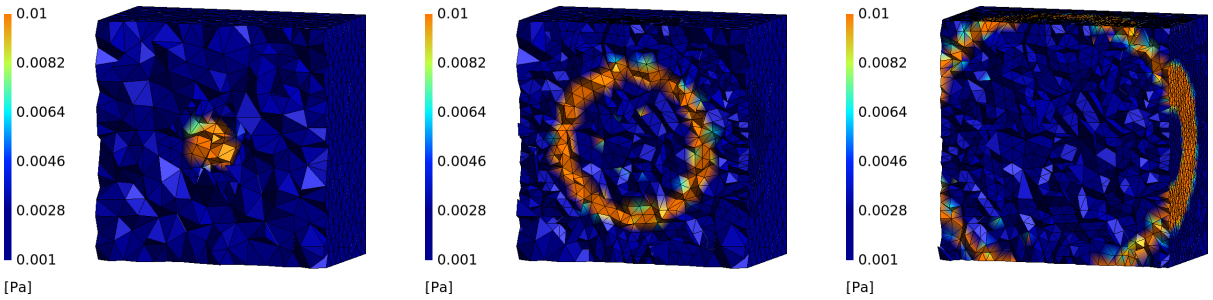


Figure 36: Pressure wave propagation in 3D homogeneous medium using third order element. The colorbar indicates the absolute pressure. The snapshots are taken at respectively 0.001, 0.015 and 0.030 [s].

We also present the temporal evolution of the isosurfaces over the domain. It is particularly suitable to give a real 3-dimensional representation of the pressure wave.

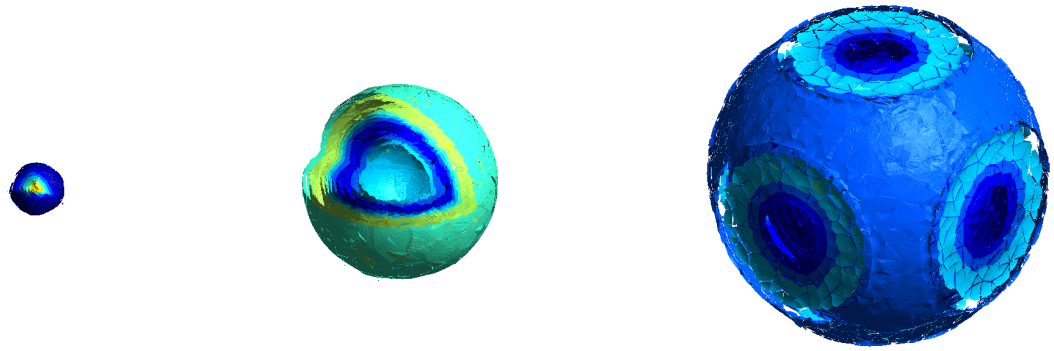


Figure 37: Isosurfaces of the pressure wave in 3D homogeneous medium. The colorbar indicates the absolute pressure. The snapshots are taken at respectively 0.001, 0.015 and 0.030 [s].

6.2.3 Auditorium

After evaluating our implementation in simple cases, we decided to move to a real world application. We decided to model a complete auditorium and to perform acoustic simulations in it. The auditorium model is purely handcrafted and is not the exact reconstruction of an existing structure.

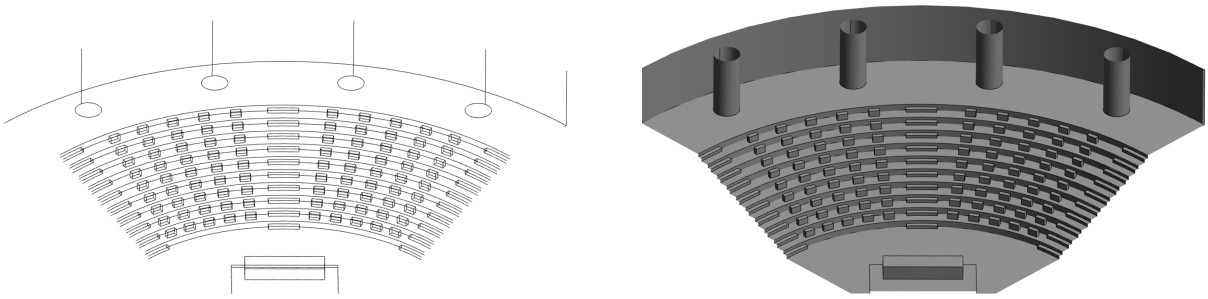


Figure 38: Auditorium front view.

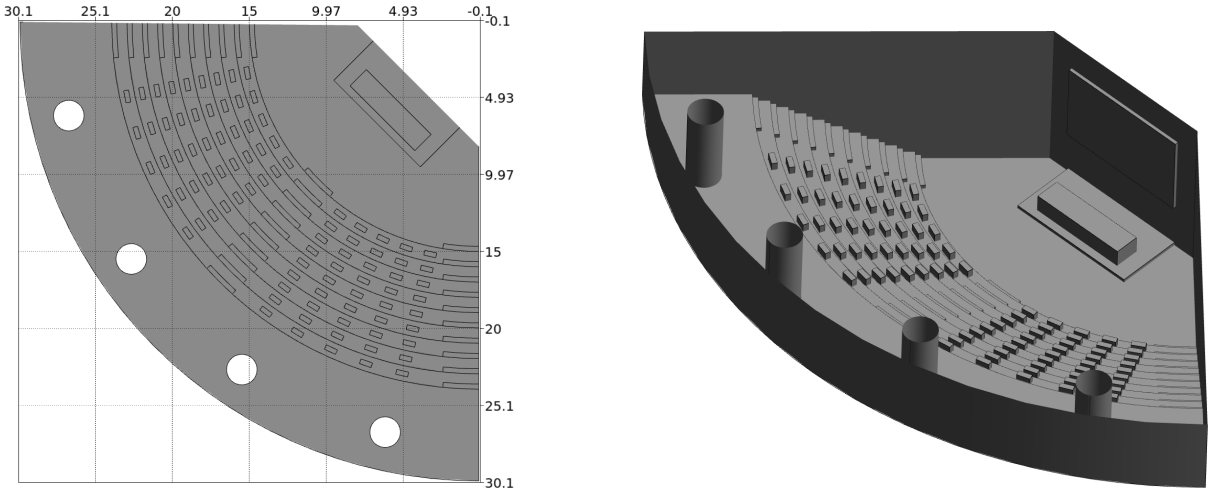


Figure 39: Auditorium geometry. The outer radius length is 30 [m], the inner radius length is 6 [m] and the total height is 10 [m]. Each floor is 0.5 [m] high and 1 [m] deep. For the furniture, we have a main desk of $1.5 \times 5 \times 0.9$ [m] and the individual desks/chairs are modeled by boxes of size $0.8 \times 0.3 \times 0.8$ [m].

For the simulations of acoustic waves, the inner space of the geometry is meshed. We opted for a tetrahedron mesh going up to the order 3. Despite our code being able to go upon third order, computational resources and time limitations prevented us to explore higher order meshes. For the same reasons, we limited our simulations to 1 million elements. Some parts of the code are also probably too inefficient to consider going further.

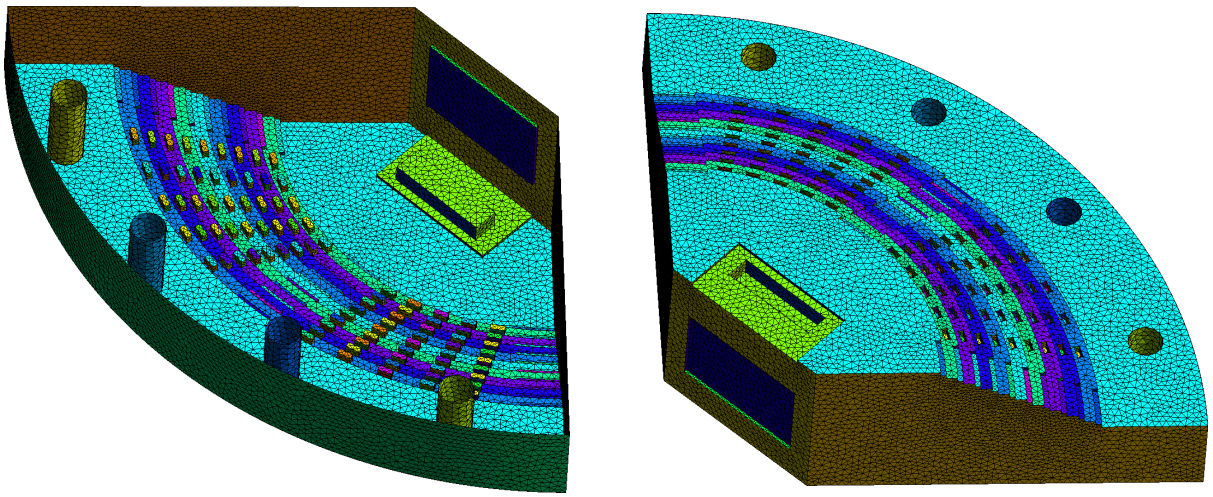


Figure 40: Auditorium 2D boundary mesh, first order elements. **Left:** Top inside view. **Right:** Bottom external view.

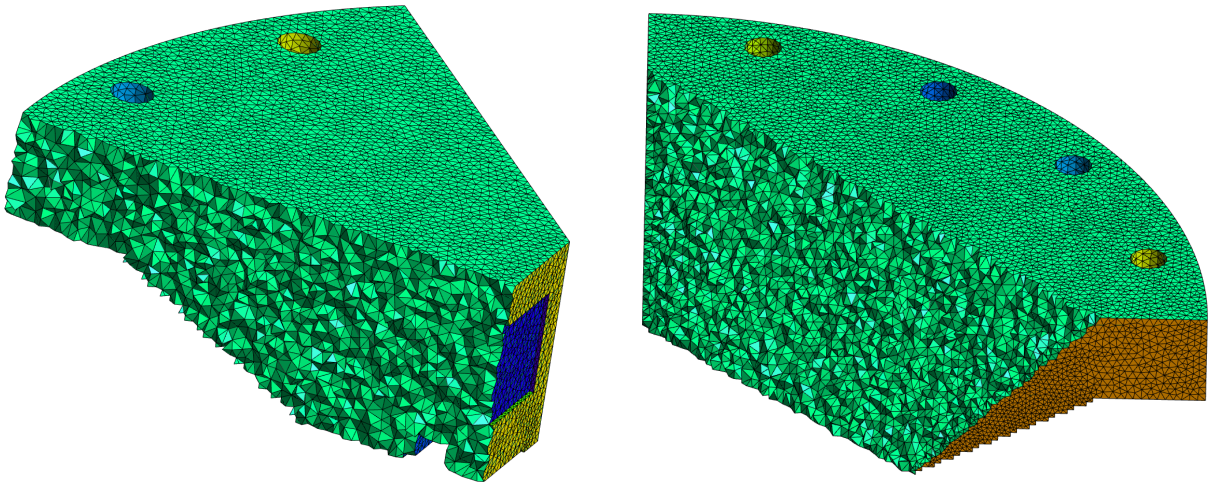
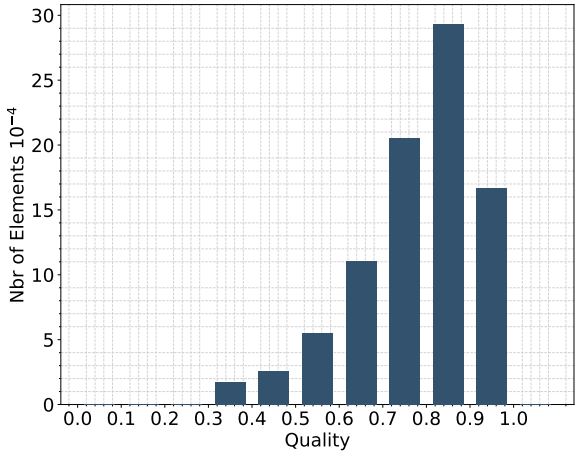


Figure 41: Auditorium bulk 3D mesh, first order elements. **Left:** Transverse view. **Right:** Inclined Longitudinal view.



The mesh is composed of:

- $202,857$ nodes
- $1,066,765$ elements
- $2,254,653$ unique faces

It has been generated with Gmsh and Delaunay algorithm in less than 5.4 [s].

Figure 42: 3D Mesh element quality.

One of most efficient way to assess the acoustic behaviour of a room is to compute its impulse response. Knowing the impulse response is indeed equivalent to know the acoustic response at each and every frequency.

$$\delta(\vec{x}) = \frac{1}{(2\pi)^3} \int_{-\infty}^{+\infty} e^{i\vec{k}\vec{x}} d\vec{k} \tag{31}$$

Following this principle, we decided to generate an impulse at the supposed position of the speaker. Furthermore, to avoid the unstable propagation of a discontinuous function such as a delta function, we used a Gaussian regularization.

$$\delta(\vec{x}) = \lim_{\sigma \to 0} \frac{1}{(2\pi\sigma^2)^{3/2}} e^{-\frac{||\vec{x}||^2}{2\sigma^2}} \tag{32}$$

The limit is numerically interpreted as the limit $\sigma \to 0.187$, where 0.187 [m] is the characteristic length of the mesh elements.

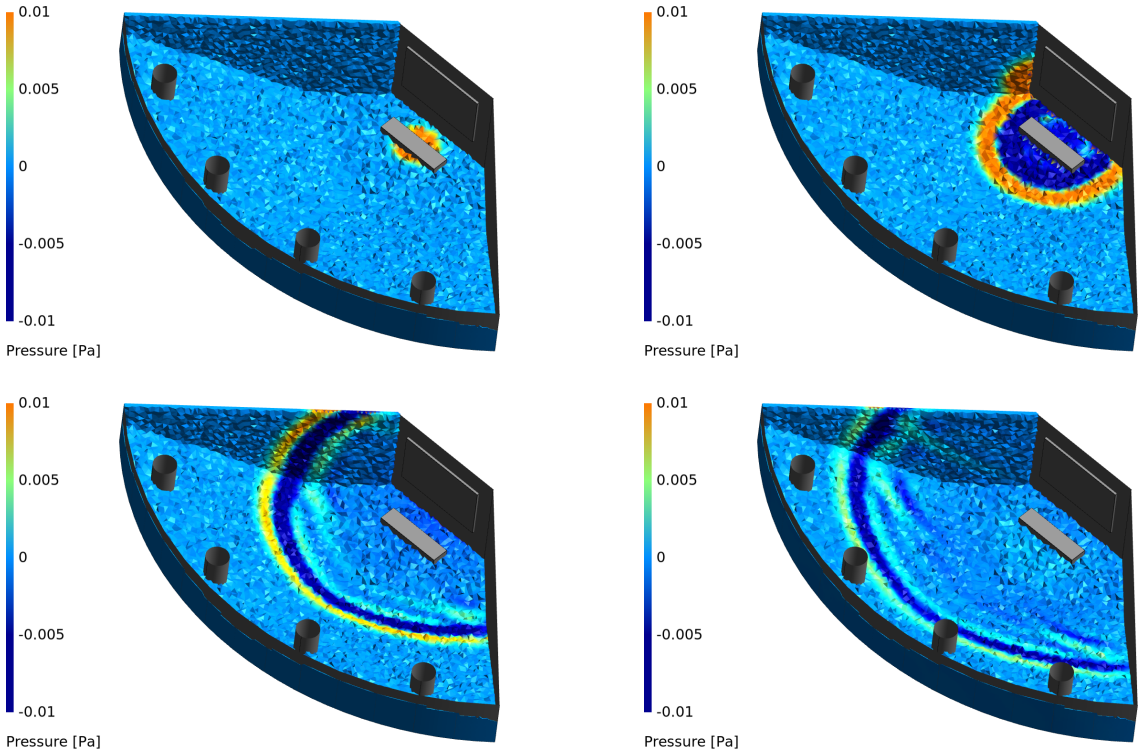


Figure 43: Propagation of a pressure pulse initially centered at $(6.5, 6.5, 1.8)$ with an amplitude of 0.05 [Pa]. The figures are respectively recorded after 0.00002, 0.0014, 0.037, 0.056 [s]. The color axis is saturated to ensure coherence between the figures and improve visualization of the propagation.

Let us thus consider a pulse centered at $(6.5, 6.5, 1.8)$ next to the desk and of amplitude 0.05 [Pa] corresponding to 60 [dB], the level of normal conversation. The walls, the stairs and the furniture are modelled as hard, reflective, nonporous surfaces with an absorbing coefficient of 5%. This behaviour is typical of interior building materials such as glass, wood or concrete. At the opposite, a perfect absorber is an open window since 100% of

the sound is transmitted. The simulation was launched with a time step of 10^{-5} [s] and up to 0.2 [s]. The simulation was parallelized using openMP over 8 threads (intel xeon e3-1230v5). The total execution time is 16 [h] 35 [min] for 1 million elements and $10,000$ steps. Note that about 20% of the time was consumed for the preprocessing. In particular the computation of the element connectivity showed a poor scaling and may represent a real challenge for larger mesh. For completeness and the elegance of the results, here is the corresponding isosurfaces.

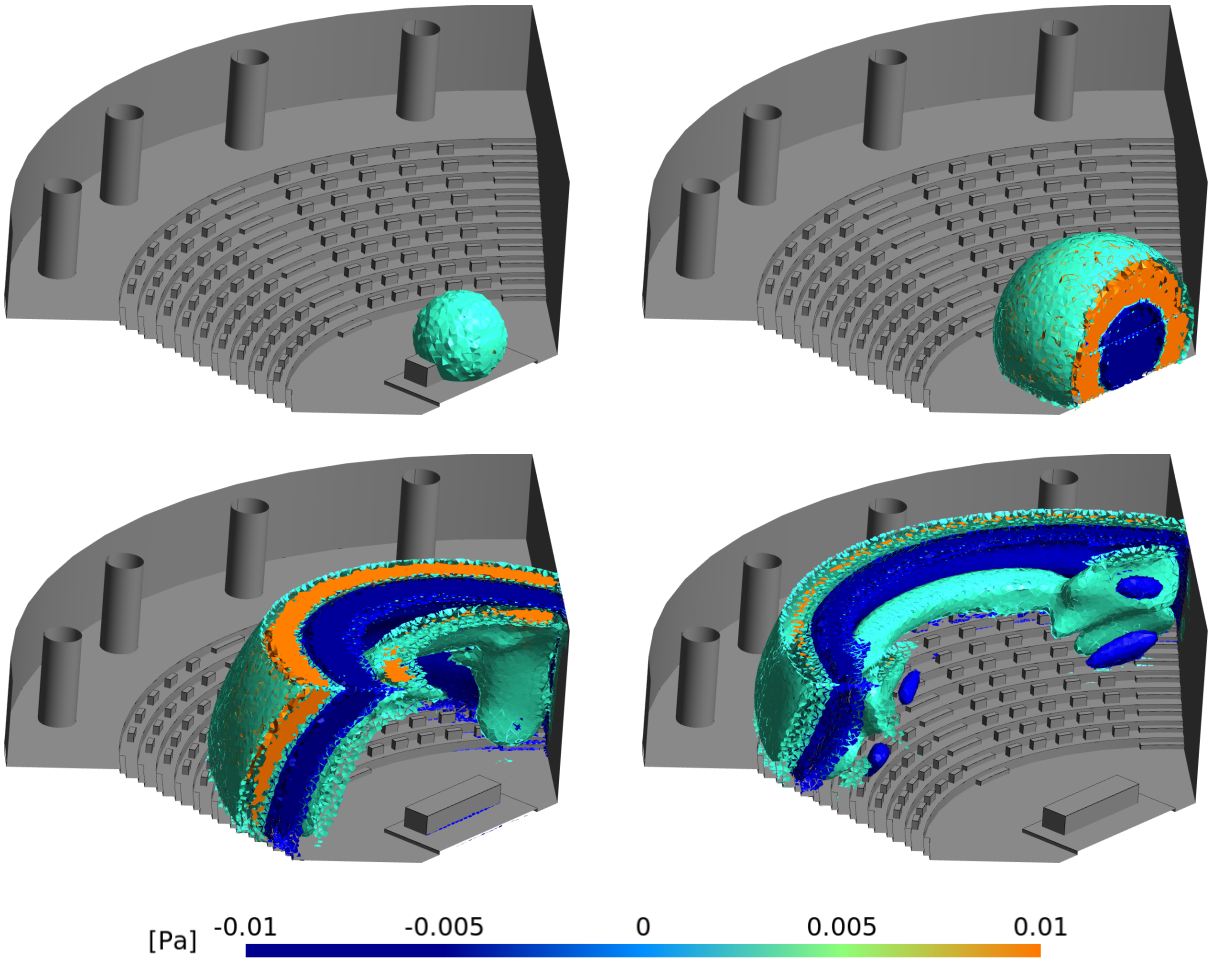


Figure 44: Propagation of a pressure pulse initially centered at $(6.5, 6.5, 1.8)$ with an amplitude of 0.05 [Pa]. The figures are respectively recorded after 0.00002, 0.0016, 0.035, 0.056 [s]. The color axis is saturated to ensure coherence between the figures and improve visualization of the propagation.

Another alternative to the simulation of the impulse response of the auditorium is the direct propagation of a wave source. We therefore reproduced the simulation using this time a continuous sine wave source of frequency 1100 [Hz] ($\approx$ C6 note). The location and the intensity are identical.

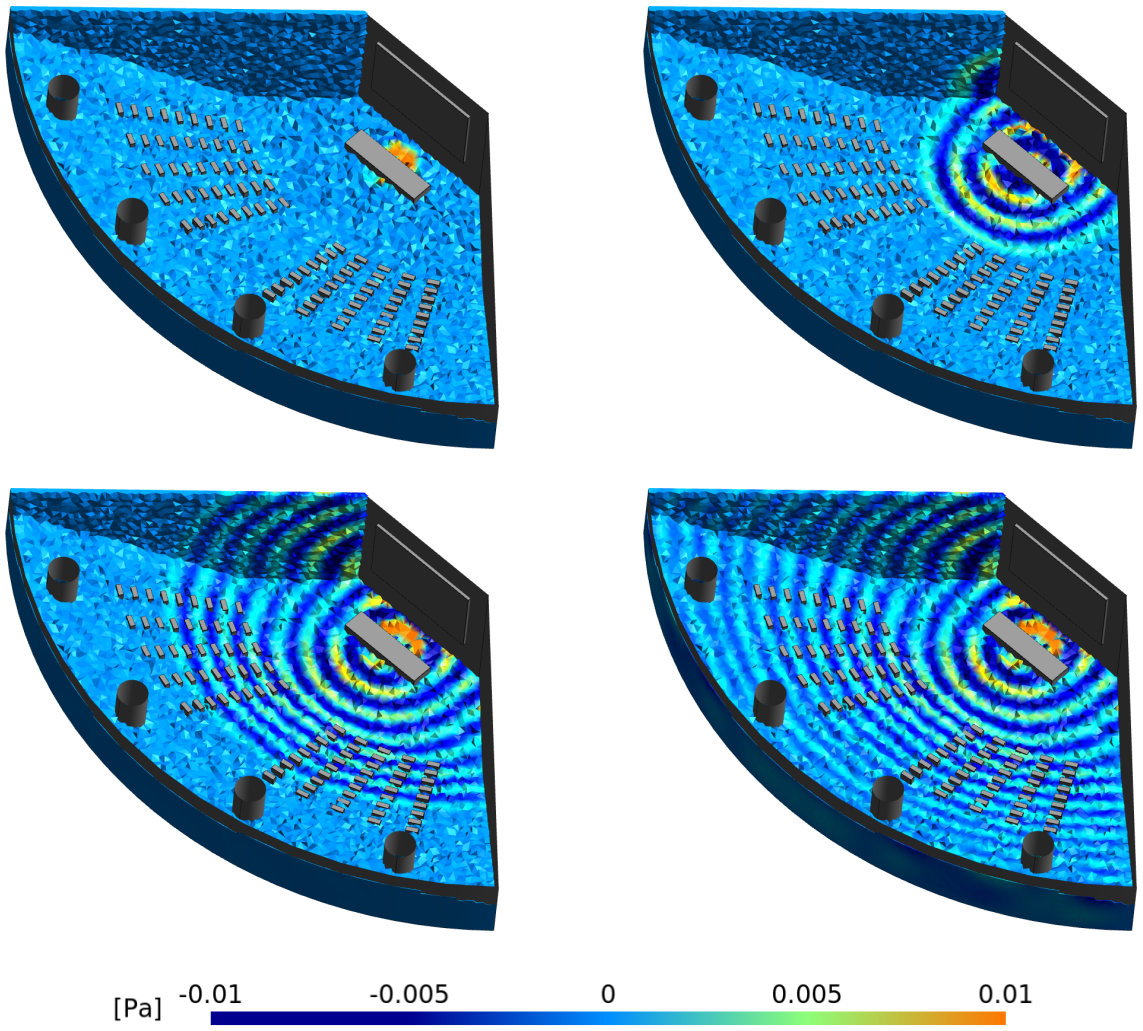


Figure 45: Propagation of a sine wave of frequency 1100 [Hz] centered at $(6.5, 6.5, 1.8)$ with an amplitude of 0.05 [Pa]. The figures are respectively recorded after 0.0002, 0.02, 0.04, 0.06 [s]. The color axis is saturated to ensure coherence between the figures and improve visualization of the propagation.

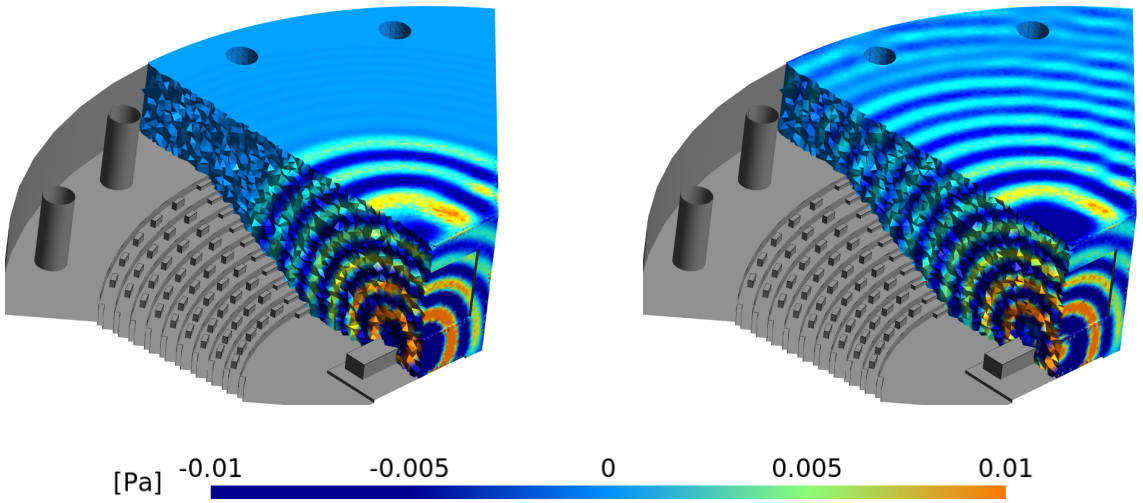


Figure 46: Propagation of a sine wave of frequency 1100 [Hz] centered at $(6.5, 6.5, 1.8)$ with an amplitude of 0.05 [Pa]. The figures are respectively recorded after 0.036, 0.06 [s].

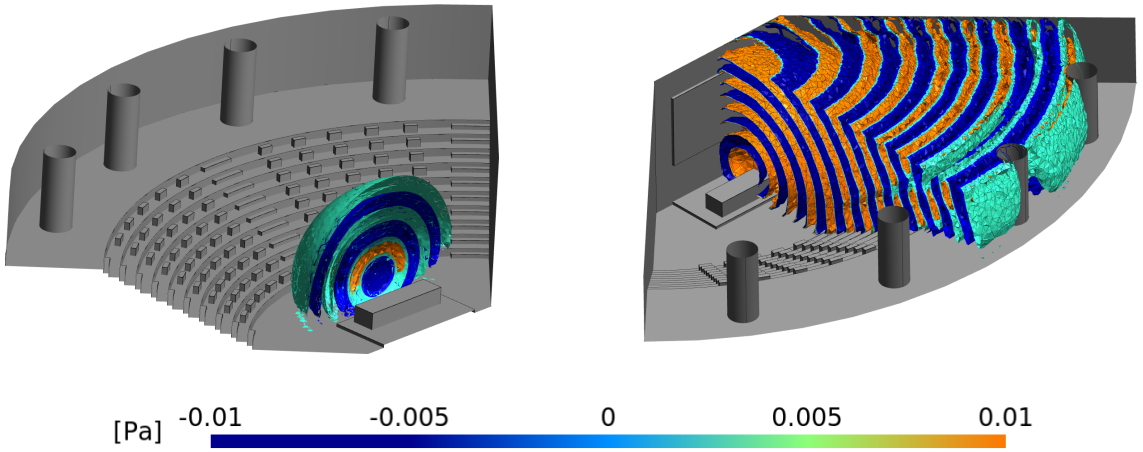
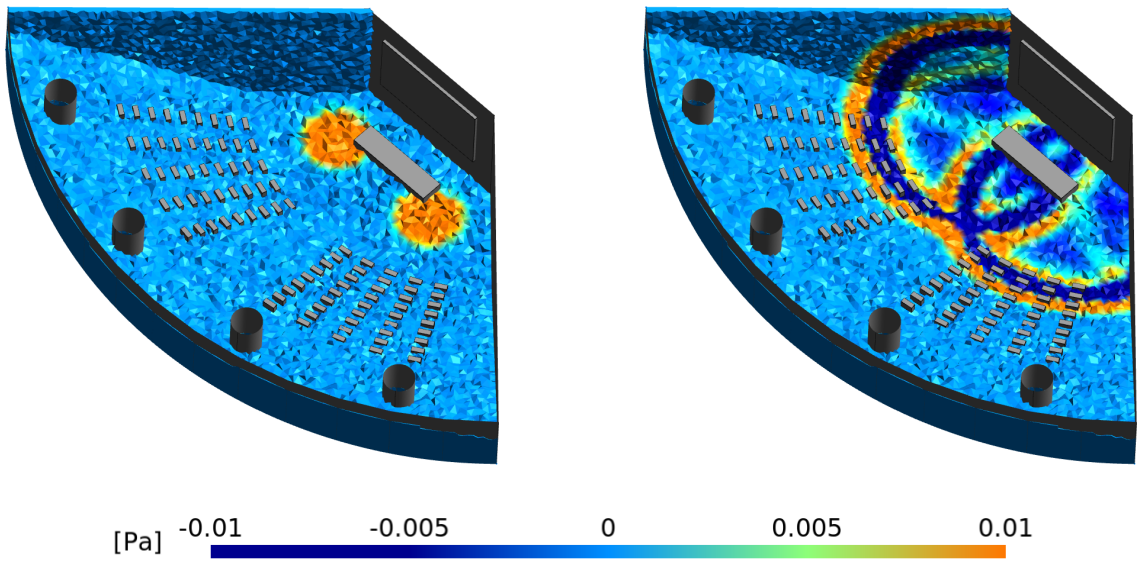


Figure 47: Isosurfaces of a sine wave of frequency 1100 [Hz] centered at $(6.5, 6.5, 1.8)$ with an amplitude of 0.05 [Pa]. The figures are respectively recorded after 0.002, 0.05 [s].

First, the two previous simulations show us that the geometry of our auditorium produces a regular and smooth propagation of the wavefront. The front and back seats receive high fidelity sound while the seats on the side are much more subject to reflections and echoes. This behaviour is due to the poor absorption (5%) of the walls. The pulse isosurfaces highlight this phenomena by the presence of residual pressure waves close to the walls. The same pattern is observed with the continuous source where the wavefront next to the wall becomes irregular. To ensure the clarity of the speech regardless of the seating location, many auditoriums therefore introduce absorbing panels.

Finally to fully explore our geometry and push our simulation to the end, a last experiment is run with second order elements. This time, not only a single pulse is emitted, but two: the first one is located at $(7.5, 4, 2)$ and the second is symmetrically positioned at $(4, 7.5, 2)$. It can formally be interpreted as some kind of stereo emitting. The simulation is run up to 0.5 [s] to focus on the reflections and the echoes in the auditorium.



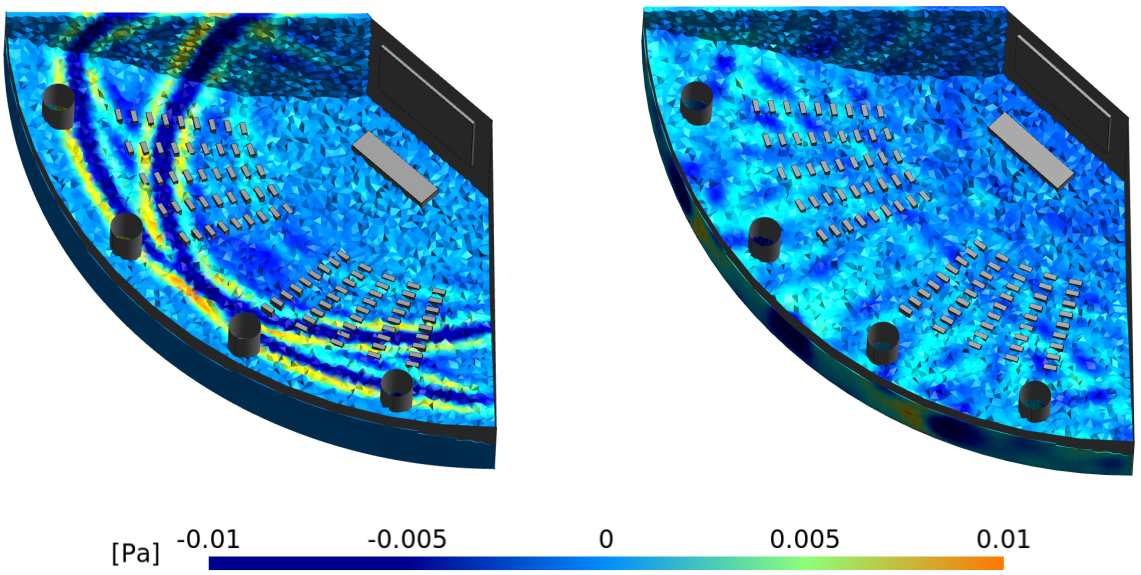


Figure 48: Propagation of two Gaussians symmetrically positioned with an initial amplitude of 0.1 [Pa]. The figures are respectively recorded after 0.005 ,0.015, 0.06, 0.1 [s]. The color axis is saturated to ensure coherence between the figures and improve visualization of the propagation.

The last picture is obviously harder to interpret, but still explicitly shows the effect of the non perfectly absorbing walls, mainly due to the back and the columns. The complex reflections are the physical interpretation of the reverberation characteristics of the auditorium. Depending on the seating, the effects are more or less important. In particular, the first rows are the optimal position both in term of intensity and sound quality.

7 Conclusion

This project has allowed us to better understand the basic principles of Discontinuous Galerkin methods as well as their limitations such as the complexity of the implementation of boundary conditions and numerical fluxes. We explored high order elements on unstructured meshes for 2- and 3-dimensional problems. We thus simulated the propagation of acoustic waves based on the conservative form of the linearized Euler equations.

Overall, a special effort has been made to produce a code as generic as possible while maintaining reasonable performances. In particular, the three-dimensional cases clearly showed us the need of highly optimized and parallelized codes.

8 Information

The source code and the different meshes presented are available at:
`https://github.com/pvanberg/DGFEM-Acoustic`

The final version of code is the labelled v1.2.0:
`https://github.com/pvanberg/DGFEM-Acoustic/releases/tag/v1.2.0`

Animation single pulse:
`https://github.com/pvanberg/DGFEM-Acoustic/blob/master/assets/auditorium_pulse_top.gif`

Animation continuous source:
`https://github.com/pvanberg/DGFEM-Acoustic/blob/master/assets/auditorium_source_side.gif`

Animation stereo:
`https://github.com/pvanberg/DGFEM-Acoustic/blob/master/assets/auditorium_stereo.gif`

9 References

[1] Jan S. Hesthaven and Tim Warburton. 2008 Nodal Discontinuous Galerkin Methods. *Texts in Applied Mathematics*. **54**, 1-39.

[2] James F. Kelly, Xiaofeng Zhao, Drew A. Murray, Simone Marras, and Robert J. McGough. 2017 Nonlinear Ultrasound Simulations Using a Time-explicit Discontinuous Galerkin Method. *Institute of Electrical and Electronics Engineers*. 1-2.

[3] Nicolas Chevaugeon and Jean-François Remacle. 2005 Efficient Discontinuous Galerkin Methods for Solving Acoustic Problems. *American Institute of Aeronautics and Astronautics*. 1-4.

[4] G. Mengaldo, D. De Grazia, J. Peiro, A. Farrington, F. Witherden, P. E. Vincent, S. J. Sherwin. 2014 A Guide to the Implementation of Boundary Conditions in Compact High-Order Methods for Compressible Aerodynamics. *American Institute of Aeronautics and Astronautics*. 6-14.