Free surface flow simulation: correcting and benchmarking the ALE method in *Code_Saturne*

A thesis submitted to The University of Manchester for the degree of
Master of Philosophy in the Faculty of Engineering and Physical Sciences

2010

Olivier COZZI

School of Mechanical, Aerospace and Civil Engineering

Table of Contents

Word count: 16 744 words

Table of Figures

Abstract

The present thesis was written by Olivier Cozzi at the University of Manchester in pursuance of the degree of Master of Philosophy in 2010. It presents "Free surface flow simulation: correcting and benchmarking the ALE method in *Code_Saturne*", that is to say the implementation of free surface flows within *Code_Saturne*, an in-house code developed by EDF (Électricité de France) to solve CFD problems, using the Arbitrary Lagrangian Eulerian (ALE) method already embedded in this code.

For a code like *Code_Saturne*, which aims at being easily implemented in a wide range of applications, the handling of free surface flows is critical because it extends the range of possible applications (tank filling, marine turbine interactions with waves and currents, water supply and reject points ...). Up to now, the ALE module within *Code_Saturne* was only used for fluid coupling with a solid structure; thus we had to adapt it to free-surface flows by adding a convergence loop to perform the free surface movement incrementally within each time step. Afterwards, the geometry was updated at the outer iterations level by imposing the displacement of each cell-vertex within the global domain: the cell-vertex motion is then computed for the free-surface cell-vertices in the first place and for the internal cell-vertices secondly.

The new free-surface module was then implemented to three different test cases:

- a standing wave in a tank (unsteady test case with a periodic analytic solution),
- a solitary wave in a tank (unsteady test case with an analytic solution),
- a submerged hydrofoil (steady test case with experimental measurements).

The results are encouraging and the feasibility is clearly demonstrated. Some limitations still exist – mainly caused by the inaccurate interpolation performed by *Code_Saturne* between the free-surface cell-vertex displacement and the free-surface cell-face centre velocities – but these could be eliminated during the next stages of the project.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

Copyright

Acknowledgements

I would like to thank my supervisors, Professor Peter Stansby, Professor Dominique Laurence and Dr Yacine Addad, for their valuable help and support throughout this project. I would like to thank Juan Uribe, Flavien Billard and Stefano Rolfo for sharing their knowledge about *Code_Saturne* and Linux.

Many thanks also to EDF in general for the funding of this MPhil, and particularly Frédéric Archambaud, Marc Sakiz, David Monfort and François Jusserand from the MFEE department at EDF R&D, and also Michel Benoit from the LNHE department at EDF R&D.

I will conclude thanking all my colleagues, both at the School of Mechanical, Aerospace and Civil Engineering in Manchester and at the MFEE department in Chatou for everything they taught me and for the amazing atmosphere they created; and finally a special acknowledgement to my sister, my helpful proofreader.

Chapter 1

Introduction

1.1. Background

The massive increase in the capacity and affordability of computers, as well as a greater awareness of the potential usefulness of numerical simulation (design optimization, physics simulation ...) have led to Computational Fluid Dynamics (CFD) being used to cope with increasingly complex and varied fluid-flow problems. In this work the development and application of moving-mesh and free surface capabilities within the general-purpose finite volume industrial code of EDF (Electricité de France) *Code_Saturne* are described and illustrated with flow calculations for different cases.

Flows with moving boundaries are indeed common among engineering problems; the movement of moving boundaries may be externally imposed (e.g., piston motion in an engine cylinder) or it may be caused by fluid forcing (for example, flow-induced vibration of the nuclear fuel rods inside a nuclear core). Moving free surfaces are frequent in hydraulic engineering, especially when it consists in an air-water interface such as in waves and tidal flows. The free surface shape, most of the times, is not easily determined, even when the free surface is stationary – such as a flow over a hydrofoil under a free surface.

The vast range of physical problems involving free surface flows has led to the implementation of a variety of CFD approaches, each with its specific applications.

Among all these approaches, the shallow water equations are applicable for long-length waves, that is to say when the horizontal length scale is much greater than the vertical length scale and when the vertical velocities are small; that is why the shallow water equations are frequently used in tidal flows and to simulate non-breaking wave propagation. These equations, also called Saint-Venant equations in their uni-dimensional form, are derived from depth-integrating the Navier-Stokes equations and considering that vertical pressure gradients are nearly hydrostatic. This shallow water approach is used in the finite element code

Telemac-2D, a free and open source CFD code, for applications in free-surface maritime or river hydraulics, such as the study presented in [1] by Chini.

The potential flow methods – which consist in describing the velocity field as the gradient of a scalar function – are applicable for flows with low vorticity effects and are used for wave dynamics and water-entry problems; in the potential flow approach, unsteady flow can be described accurately [2].

These methods are powerful tools in their own areas but their embedded approximations make them unsuitable for general-purpose flow solvers.

Among the general purpose grid-based flow solvers with free surface capabilities, one can list:

- *OpenFOAM* (Open Field Operation and Manipulation), a free and open source CFD software, is based on a finite volume approach to solve systems of partial differential equations ascribed on any 3D unstructured mesh of polyhedral cells. The fluid flow solvers are developed within a robust, implicit, pressure-velocity, iterative solution framework and space parallel computation is available. For free surface flows, both surface tracking and surface capturing methods are available.
- *STAR-CCM+*, a finite-volume-based program package for the modelling of fluid flow problems, is developed by the computer software company CD-adapco. It solves the Navier-Stokes equations with a segregated, algebraic multigrid solver using the Rhie & Chow interpolation for pressure-velocity coupling. Furthermore the SIMPLE algorithm is applied to control the overall solution. For free surface flows, *STAR-CCM+* uses the Volume of Fluid (VOF) approach. The numerical model can be applied to any structured and unstructured grid with arbitrarily shaped control volumes.
- *CFX* is developed by the engineering simulation software company ANSYS. In *CFX*, a conservative finite-element-based control volume method (i.e. a finite volume approach with parts of the finite element method) is implemented. The Navier-Stokes equations are discretized in a collocated way and solved by an algebraic multigrid solver. To avoid pressure-velocity decoupling, the Rhie & Chow interpolation is used; all conservation equations are solved in one linear equation system, with all equations being fully coupled. The reconstruction of the free surface is based on the VOF approach where the volume fraction is computed using an upwind-biased discretization.

- *FLOW-3D* is a general purpose CFD code based on a finite volume/finite difference approach. For free surface flows, an adapted version of the Volume of Fluid method known as TruVOF is used. *FLOW-3D* and its TruVOF technique consider the three ingredients of the original VOF method: a scheme to locate the free surface, an algorithm to track the free surface as a sharp interface moving through a computational grid, and a means of applying boundary conditions at the free surface.

Other codes like *FLUENT* (ANSYS) or *STAR-CD* (CD-adapco) must also be mentioned.

As presented further in the section 3.1.1, *Code_Saturne* is a general purpose, free and open source industrial CFD code developed by EDF. It is based on a collocated finite volume approach and accepts unstructured and non-conform meshes. The velocity and the pressure are both considered with a cell-centered co-located approach. The velocity-pressure coupling is obtained by a predictor-corrector scheme based on the SIMPLEC method.

1.2. Objectives

The aim of the ReDAPT (Reliable Data Acquisition Platform for Tidal) project is to simulate the flow around marine turbines with the presence of free-surface effects. Within the framework of this project, *Code_Saturne* was chosen for the development of a 3D numerical model of an idealised geometry of a horizontal-axis tidal turbine. Indeed, some capabilities within *Code_Saturne*, such as the fully validated Arbitrary Lagrangian Eulerian method and the fluid-structure coupling, make it capable of handling numerical simulations of a submerged and rotating marine turbine. This is why a free-surface capability must be implemented and validated within *Code_Saturne*; this is the objective of the present work.

1.3. Outline of the report

After an introduction to free surface flows and free surface modeling in Computational Fluid Dynamics in Chapter 2, the Arbitrary Lagrangian Eulerian method already embedded in *Code_Saturne* (Chapter 3) is used to develop a free surface module (Chapter 4). Then in Chapter 6, limits of existing module are explained and an outline of improvements for a new version is presented. A number of applications are described in Chapter 5. These are: (5.1) oscillation of small-amplitude waves in a tank and (5.2) solitary wave in a channel, both with inviscid fluid; and (5.3) flow over a hydrofoil under a free surface, for which J.H. Duncan realized many experiments.

Chapter 2

Introduction to free surface flows and computational fluid dynamics

2.1. Free surface flows

2.1.1. Definition of a free surface

In physics a free surface is the surface of a fluid that is subject to constant perpendicular normal stress and zero parallel shear stress, such as the boundary between two homogeneous fluids, for example liquid water and the air in the Earth's atmosphere.

A liquid in a gravitational field will form a free surface if unconfined from above. Under mechanical equilibrium this free surface must be perpendicular to the forces acting on the liquid; if not there would be a force along the surface, and the liquid would flow in that direction. Thus, on the surface of the Earth, all free surfaces of liquids are horizontal unless disturbed (except near solids dipping into them, where surface tension distorts the surface locally).

In a free liquid at rest, that is to say one subject to internal attractive forces only and not affected by outside forces such as a gravitational field, its free surface will assume the shape with the least surface area for its volume: a perfect sphere. This can be seen under weightless conditions, such as the spatial flight of Tintin and Captain Haddock during their journey to the moon (Figure 2.1.1-1).

FIGURE 2.1.1-1: CAPTAIN HADDOCK AND A SPHERICAL DROP OF HIS BELOVED WHISKY

("THE ADVENTURES OF TINTIN: EXPLORERS ON THE MOON" BY HERGÉ, CASTERMAN - 1954)

2.1.2. Computational fluid dynamics and free surface modeling

A state-of-the-art of the numerical methods used for the computation of incompressible flows involving a nonlinear free surface is presented by Tsai and Yue in [3]: in this article, potential as well as rotational and viscous free surface flows are considered.

In [4], Floryan and Rasmussen classify the available algorithms for the analysis of viscous flows with moving interfaces in three groups: Lagrangian, Eulerian and mixed, i.e. Eulerian-Lagrangian methods. The Lagrangian group mainly consists of strictly Lagrangian and particle methods. The Eulerian group is composed by fixed grid and adaptive grid methods. The third group of the mixed methods relies on both Lagrangian and Eulerian concepts.

In [5], Ferziger and Peric present the three main CFD approaches to solve free surface problems: fixed-mesh methods, moving-mesh methods and mesh-free methods.

In fixed-mesh methods (interface-capturing methods), the computation is performed on a fixed grid, which extends beyond the free surface, and the shape of the free surface is determined by computing the fluid-containing fraction of each near-interface cell. In the context of free-surface flows, well-known techniques include the volume-of-fluid (VOF) method where a transport equation is solved for the fraction of the cell occupied by the liquid phase (also known as void fraction), and the marker-and-cell (MAC) method where the free surface is tracked by following the motion of particles on the interface. Hirt and Nichols present the VOF method in [6], whereas the MAC method is described by Harlow and Welch in [7]. The

fixed-mesh methods are more robust than the moving mesh ones presented below. They can indeed handle breaking waves, but the drawback is that these methods are not effective for resolving a sharp interface and, for free surfaces which are rapidly varying in time or space, they require an extremely fine mesh that needs to be created beforehand (one needs to know where the free surface will be located); these methods can be very time consuming.

Moving-mesh methods (interface-tracking methods) consist in adapting dynamically the mesh in such a way that it is always surface-conforming (mesh cells always contain fluid): boundary faces are then impermeable solids or free surfaces. We will adopt these methods in the present work because they are particularly adapted to the finite volume approach, thanks to their natural relationship with the fundamental integral forms of the governing conservation equations (equations (1) to (4)). This approach was applied by Mayer in [8] and by Thé, Raithby and Stubley in [9]. However, the handling of breaking waves with a moving mesh is quite complex: it requires an algorithm capable of removing or adding cells around the multiple connected regions where the wave breaks.

Mesh-free methods can also solve free surface flows. The popular smoothed-particle hydrodynamics (SPH) method is the earliest mesh-free particle method to be developed: it is a mesh-free Lagrangian method particularly adapted to model fluid motion, with a lot of benefits over traditional grid-based techniques (for example mass conservation, pressure computation, free surface geometry) but also drawbacks – such as the need for a large number of particles to obtain similar results, which is time consuming. An interesting application of the SPH method to complex turbulent free surface flows is presented by Violeau and Issa in [10], using the *SPHysics* code developed at the University of Manchester.

2.2. Equations of the free surface

Considering a control volume Ω with a moving boundary $\partial\Omega$, the three usual conservation laws in fluid mechanics which have to be considered are:

- Mass Conservation Law:

$$\frac{d}{dt}\int_{\Omega} \rho d\Omega + \int_{\partial\Omega} \rho(\boldsymbol{u}-\boldsymbol{w}).\boldsymbol{dS} = 0 \qquad (1)$$

- Momentum Conservation Law:

$$\frac{d}{dt}\int_{\Omega} \rho \boldsymbol{u} d\Omega + \int_{\partial\Omega} (\rho(\boldsymbol{u}-\boldsymbol{w}) \otimes \boldsymbol{u} - \boldsymbol{T}).\boldsymbol{dS} = \int_{\Omega} \boldsymbol{s_u} d\Omega \quad (2)$$

- Scalar Conservation Law (concentration, thermal energy, ...):

$$\frac{d}{dt}\int_{\Omega} \rho \Phi d\Omega + \int_{\partial\Omega} (\rho(\boldsymbol{u}-\boldsymbol{w})\Phi - \boldsymbol{q}).\boldsymbol{dS} = \int_{\Omega} s_{\Phi} d\Omega \quad (3)$$

Where $\boldsymbol{u}$ is the fluid velocity, $\boldsymbol{w}$ is the velocity of the boundary $\partial\Omega$ of the control volume Ω (e.g. the free surface velocity), $\boldsymbol{dS}$ is the surface vector, ρ represents the fluid density, Φ stands for any scalar quantity, s_{Φ} and $\boldsymbol{s_u}$ are the volumetric sources of scalar quantity and momentum, and $\boldsymbol{T}$ and $\boldsymbol{q}$ are respectively the stress tensor and the flux vector.

Because of the moving boundary, the surface velocity $\boldsymbol{w}$ needs to meet the Space Conservation Law (SCL):

$$\frac{d}{dt}\int_{\Omega} d\Omega - \int_{\partial\Omega} \boldsymbol{w}.\boldsymbol{dS} = 0 \quad (4)$$

This equation describes the conservation of space when the domain changes its shape and position with time. Considering the SCL, the equations (1) to (3) simplify to:

- Mass Conservation Law:

$$\int_{\Omega} \frac{d\rho}{dt} d\Omega + \int_{\partial\Omega} \rho \boldsymbol{u}.\boldsymbol{dS} = 0 \quad (5)$$

- Momentum Conservation Law:

$$\int_{\Omega} \rho \frac{d\boldsymbol{u}}{dt} d\Omega + \int_{\partial\Omega} (\rho(\boldsymbol{u}-\boldsymbol{w}) \otimes \boldsymbol{u} - \boldsymbol{T}).\boldsymbol{dS} - \boldsymbol{u}\int_{\partial\Omega} \rho(\boldsymbol{u}-\boldsymbol{w}).\boldsymbol{dS} = \int_{\Omega} \boldsymbol{s_u} d\Omega \quad (6)$$

- Scalar Conservation Law (concentration, thermal energy, ...):

$$\int_{\Omega} \rho \frac{d\Phi}{dt} d\Omega + \int_{\partial\Omega} (\rho(\boldsymbol{u}-\boldsymbol{w})\Phi - \boldsymbol{q}).\boldsymbol{dS} - \Phi\int_{\partial\Omega} \rho(\boldsymbol{u}-\boldsymbol{w}).\boldsymbol{dS} = \int_{\Omega} s_{\Phi} d\Omega \quad (7)$$

The specificity of free surface flows is based on two additional equations, the kinematic and the dynamic boundary conditions for the free surface:

- The dynamic boundary condition is quite simple when shear stress, normal stress and effect of the surface tension can be neglected on the free surface; it then consists in:

$$p = p_{atm} \tag{8}$$

where p and p_{atm} are respectively the free surface pressure and atmospheric pressure.

- The kinematic boundary condition can be thought of as a "zero mass flux through the free surface":

$$\rho(\boldsymbol{u} - \boldsymbol{w}).\boldsymbol{S} = 0 \tag{9}$$

where ρ is the fluid density and $\boldsymbol{S}$ is the surface vector as it is shown in Figure 2.1.2-1:

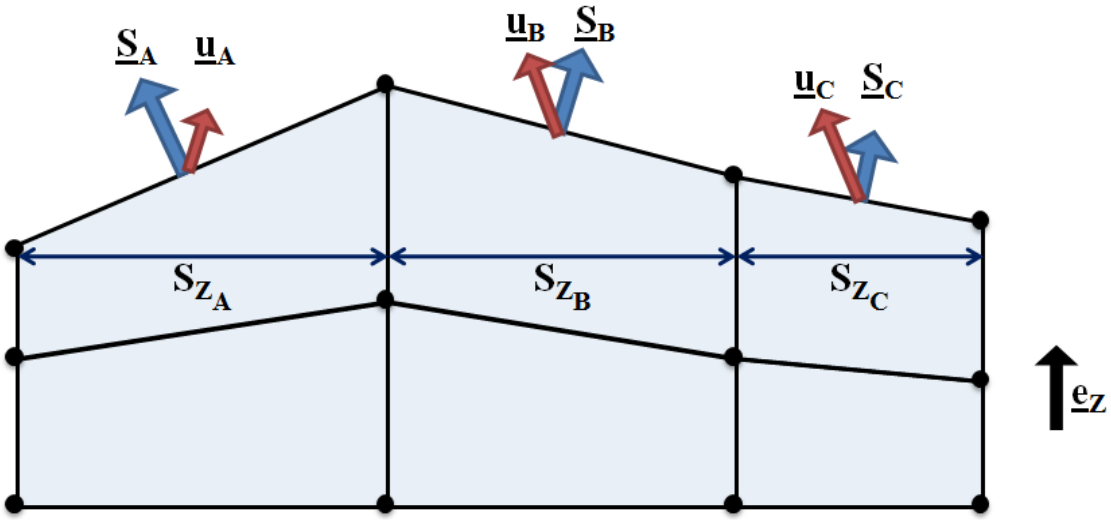

FIGURE 2.1.2-1: 2D REPRESENTATION OF THE MESH GEOMETRY UNDER THE FREE SURFACE

Considering that the fluid density is constant (incompressible flows) and the mesh velocity is only in the vertical direction ($\boldsymbol{w} = w.\boldsymbol{e_z}$ where the unit vector $\boldsymbol{e_z}$ depends on the gravity vector $\boldsymbol{g}$ such as $\boldsymbol{e_z} = -\boldsymbol{g}/\|\boldsymbol{g}\|$), the kinematic boundary condition can be rewritten as:

$$w = \frac{\boldsymbol{u}.\boldsymbol{S}}{\boldsymbol{e_z}.\boldsymbol{S}} = \frac{\dot{m}_{fs}}{\rho S_z} \tag{10}$$

where S_z is the vertical component of the surface vector $\boldsymbol{S}$, and $\dot{m}_{fs}$ is the mass flux through the free surface (or rather the mass flux supposed to go through the free surface when the free surface is considered as fixed).

In the present work, a moving-mesh method will be adopted to tackle free surface flows with *Code_Saturne*. This requires two capabilities: the handling of a moving mesh and the control of the free surface motion. The first of these two capabilities is managed by the Arbitrary Lagrangian Eulerian module (ALE) already embedded in *Code_Saturne* and presented in the next chapter. The second is the result of this work and is presented in Chapter 4.

Chapter 3

Code_Saturne and its ALE module

3.1. *Code_Saturne*

3.1.1. Presentation of *Code_Saturne*

All the developments presented in this report have been done using *Code_Saturne*, a code developed by EDF (Electricité de France) to solve CFD problems. A complete description of this code can be found in [11] or in the document "Theory and Programmer's Guide" of *Code_Saturne*. Below is a relatively short presentation of *Code_Saturne*.

Code_Saturne is initially designed to solve the Navier-Stokes equations for three-dimensional single phase flows using a finite volume discretization scheme. It handles unstructured and non-uniform meshes for steady or transient, laminar or turbulent, incompressible or slightly compressible flows. *Code_Saturne* also computes the transport of passive scalar and features many other modules to handle a wide range of particular physics, such as combustion problems. *Code_Saturne* is divided into two separate programs:

- the kernel solves the problem equations – e.g. the Navier-Stokes, turbulence, passive scalars equations,
- and the shell processes the mesh to make it readable by the kernel and also creates the outputs necessary for post-processing software.

Code_Saturne uses the finite volume approach in which the equations are written in a conservative form and then integrated over control volumes in order to be solved. This finite volume method is described by Versteeg and Malalasekera in [12] and by Zwart in his thesis [13]. The velocity and the pressure are both considered with a cell-centered co-located approach. The Gauss theorem is used to transform integrals of the divergence of any vector field into surface integral of flux over faces; this way the conservative form of the momentum equation reads:

$$\frac{\partial \boldsymbol{u}}{\partial t} + div(\boldsymbol{u} \otimes \boldsymbol{u}) = div\left(-\frac{P}{\rho} Id + \nu\left(grad(\boldsymbol{u}) + {}^{t}grad(\boldsymbol{u})\right)\right)$$

where ν is the molecular viscosity (the turbulent viscosity is ignored here).

The velocity-pressure coupling is obtained by a predictor-corrector scheme: at each time step n, the momentum equation is first solved taking the pressure as explicit, that is the velocity prediction step and leads to the predicted velocity $\mathbf{u}^{n+1/2}$. This predicted velocity is then modified by the corrector step in order to be divergence free (incompressible flow). The time discretization (fractional step scheme) can be associated with the SIMPLEC method. The SIMPLEC algorithm is also presented by Versteeg and Malalasekera in their book [12] presenting the volume finite method.

A - The predictor step

In this step, *Code_Saturne* solves the following equation using an Euler implicit scheme:

$$\rho \frac{\boldsymbol{u}^{n+1/2} - \boldsymbol{u}^{n}}{\Delta t} + div\left(\boldsymbol{u}^{n+1/2} \otimes (\rho \boldsymbol{u})^{n}\right) = div\left(-P^{n} Id + \mu\left(grad\left(\boldsymbol{u}^{n+1/2}\right) + {}^{t}grad(\boldsymbol{u}^{n})\right)\right)$$

The mass flux $(\rho \boldsymbol{u})^{n}$ in the left-hand side of this equation is taken as explicit in order to uncouple the 3 components of the velocity.

As usually done in the finite volume method, the domain Ω is partitioned in control volumes Ω_i. Let S_{ij} be defined as $S_{ij} = \partial\Omega_i \cap \partial\Omega_j$ the common face to Ω_i and Ω_j, and S_{bik} the k-th boundary face of $\partial\Omega_i \cap \partial\Omega_j$, represented in Figure 3.1.1-1 below.

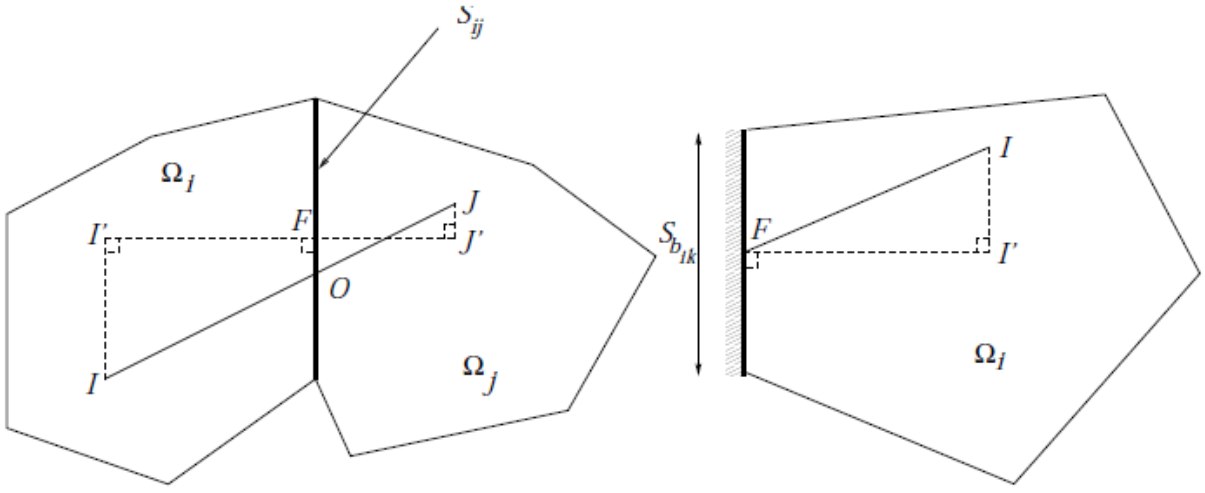

FIGURE 3.1.1-1: REPRESENTATION OF A CELL AND A BOUNDARY FACE

Thanks to the Gauss theorem, the volume integral $\int_{\Omega_i} div\left(\boldsymbol{u}^{n+1/2} \otimes (\rho \boldsymbol{u})^n\right) d\Omega$ of the convective term is transformed into $\int_{\partial\Omega_i}\left(\boldsymbol{u}^{n+1/2} \otimes (\rho \boldsymbol{u})^n\right).\boldsymbol{dS}$. The discretization over the faces of Ω_i reads:

$$\sum_{j\in Neigh(i)} \left[\left(\rho \boldsymbol{u}_{ij}^n\right).\boldsymbol{S}_{ij}\right]\boldsymbol{u}_{f,ij}^{n+1/2} + \sum_{k\in\gamma_b(i)} \left[\left(\rho \boldsymbol{u}_{bik}^n\right).\boldsymbol{S}_{bik}\right]\boldsymbol{u}_{f,bik}^{n+1/2}$$

The terms $\left(\rho \boldsymbol{u}_{ij}^n\right).\boldsymbol{S}_{ij}$ and $\left(\rho \boldsymbol{u}_{bik}^n\right).\boldsymbol{S}_{bik}$ stand for the mass fluxes, expressed on inner faces and boundary faces. The set $Neigh(i)$ and $\gamma_b(i)$ represent respectively the neighbouring cells of the cell i and its adjacent boundary faces. The unknown values of $\boldsymbol{u}_{f,ij}^{n+1/2}$ and $\boldsymbol{u}_{f,bik}^{n+1/2}$ have to be related to the values of the variables expressed at the nearby cell centres. *Code_Saturne* uses three schemes to compute these unknowns:

- The UPWIND scheme (first order):

$\boldsymbol{u}_{f,ij}^{n+1/2} = \boldsymbol{u}_I^{n+1/2}$ if $\left(\rho \boldsymbol{u}_{ij}^n\right).\boldsymbol{S}_{ij} \geq 0$

$\boldsymbol{u}_{f,ij}^{n+1/2} = \boldsymbol{u}_J^{n+1/2}$ if $\left(\rho \boldsymbol{u}_{ij}^n\right).\boldsymbol{S}_{ij} < 0$

- The Second Order Linear Upwind (SOLU) scheme:

$\boldsymbol{u}_{f,ij}^{n+1/2} = \boldsymbol{u}_I^{n+1/2} + \boldsymbol{IF}.grad\left(\boldsymbol{u}^{n+1/2}\right)_I$ if $\left(\rho \boldsymbol{u}_{ij}^n\right).\boldsymbol{S}_{ij} \geq 0$

$\boldsymbol{u}_{f,ij}^{n+1/2} = \boldsymbol{u}_J^{n+1/2} + \boldsymbol{JF}.grad\left(\boldsymbol{u}^{n+1/2}\right)_J$ if $\left(\rho \boldsymbol{u}_{ij}^n\right).\boldsymbol{S}_{ij} < 0$

- The second order scheme:

$$\boldsymbol{u}_{f,ij}^{n+1/2} = \alpha_{ij}\boldsymbol{u}_I^{n+1/2} + \left(1-\alpha_{ij}\right)\boldsymbol{u}_J^{n+1/2} + \frac{1}{2}\left(grad\left(\boldsymbol{u}^{n+1/2}\right)_I + grad\left(\boldsymbol{u}^{n+1/2}\right)_J\right).\boldsymbol{OF},$$

using $\alpha_{ij} = \frac{FJ'}{I'J'}$ (see Figure 3.1.1-1)

In second order schemes, the computation of the gradients needs a gradient reconstruction technique when the mesh has non-orthogonalities ($I \neq I'$). It is an iterative process that takes into account first order terms in space.

At the boundaries, the value of the predicted velocity is always given by: $\boldsymbol{u}_{f,bik}^{n+1/2} = \boldsymbol{u}_I^{n+1/2}$ if $\left(\rho \boldsymbol{u}_{bik}^n\right).\boldsymbol{S}_{bik} \geq 0$ and $\boldsymbol{u}_{f,bik}^{n+1/2} = \boldsymbol{u}_J^{n+1/2}$ if $\left(\rho \boldsymbol{u}_{bik}^n\right).\boldsymbol{S}_{bik} < 0$.

Thanks to the Gauss theorem, the volume integral $\int_{\Omega_i} div\left(\boldsymbol{u}^{n+1/2} \otimes (\rho \boldsymbol{u})^n\right) d\Omega$ of the convective term is transformed into $\int_{\partial\Omega_i} \left(\boldsymbol{u}^{n+1/2} \otimes (\rho \boldsymbol{u})^n\right).\boldsymbol{dS}$. The discretization over the faces of Ω_i reads:

As for the diffusion term, in the finite volume method, the volume integral $\int_{\Omega_i} div\left(\mu\, grad\left(\boldsymbol{u}^{n+1/2}\right)\right) d\Omega$ is transformed into $\int_{\partial\Omega_i} \mu\, grad\left(\boldsymbol{u}^{n+1/2}\right).\boldsymbol{dS}$ and is expressed, after discretization:

$$\sum_{j \in Neigh(i)} \mu \frac{\boldsymbol{u}_{J'}^{n+1/2} - \boldsymbol{u}_{I'}^{n+1/2}}{\boldsymbol{I'J'}} S_{ij} + \sum_{k \in \gamma_b(i)} \mu \frac{\boldsymbol{u}_{bik}^{n+1/2} - \boldsymbol{u}_{I'}^{n+1/2}}{\boldsymbol{I'F'}} S_{bik}$$

This manner of discretizing the diffusion term does not cause any problems on orthogonal meshes (when $I = I'$). Otherwise, as done for the convection term, reconstruction techniques are necessary. The value of the velocity at the boundaries $\boldsymbol{u}_{f,bik}^{n+1/2}$ depends on the velocity in the adjacent cell I and the boundary conditions given by the user.

B - The correction step

In the correction step, i.e. the second step of the SIMPLEC scheme, a Poisson equation is solved to compute the pressure and then a divergence free corrected velocity field $\boldsymbol{u}^{n+1}$ is obtained. The pressure P^n is updated by adding an increment δP^{n+1} ($\delta P^{n+1} = P^{n+1} - P^n$). The following problem has to be solved (the convection term is neglected in the SIMPLEC algorithm):

$$\frac{(\rho \boldsymbol{u})^{n+1} - (\rho \boldsymbol{u})^{n+1/2}}{\Delta t} = -grad(\delta P^{n+1})$$

$$div(\rho \boldsymbol{u})^{n+1} = 0$$

In order to solve this system of equations, the divergence operator is applied to the first equation above, which produces a Laplacian of the pressure increment:

$$div\left(\Delta t grad(\delta P^{n+1})\right) = div(\rho \boldsymbol{u})^{n+1/2}$$

With the usual discretization of the Laplace operator, odd and even nodes are uncoupled, leading to the well known chessboard-like pressure field: a pressure field, whose values are −1 on odd nodes and +1 on even nodes on a hexahedral regular mesh is solution of the homogeneous Poisson equation, and can appear for any solution and alter it. In order to avoid this problem, the Rhie & Chow filter [14] is used in *Code_Saturne*.

At this point, the mass flows and the velocity field are updated according to:

$$(\rho \boldsymbol{u})^{n+1} = (\rho \boldsymbol{u})^{n+1/2} - \Delta t grad(\delta P^{n+1})$$

Afterwards the turbulence and scalar equations are solved. When all the values of the new time step n+1 are known, the algorithm can carry on with the next time step.

3.1.2. Calling tree

In the *Code_Saturne* documentation, and particularly in the "Theory and Programmer's Guide", a calling tree is presented (see *Appendix 1: Successive stages within a time step*) but this calling tree is not very clear for our subject: the routines related to the Arbitrary Lagrangian Eulerian method (ALE) are not mentioned and the links between routines are not explained. Let us consider this other calling tree (Figure 3.1.2-1):

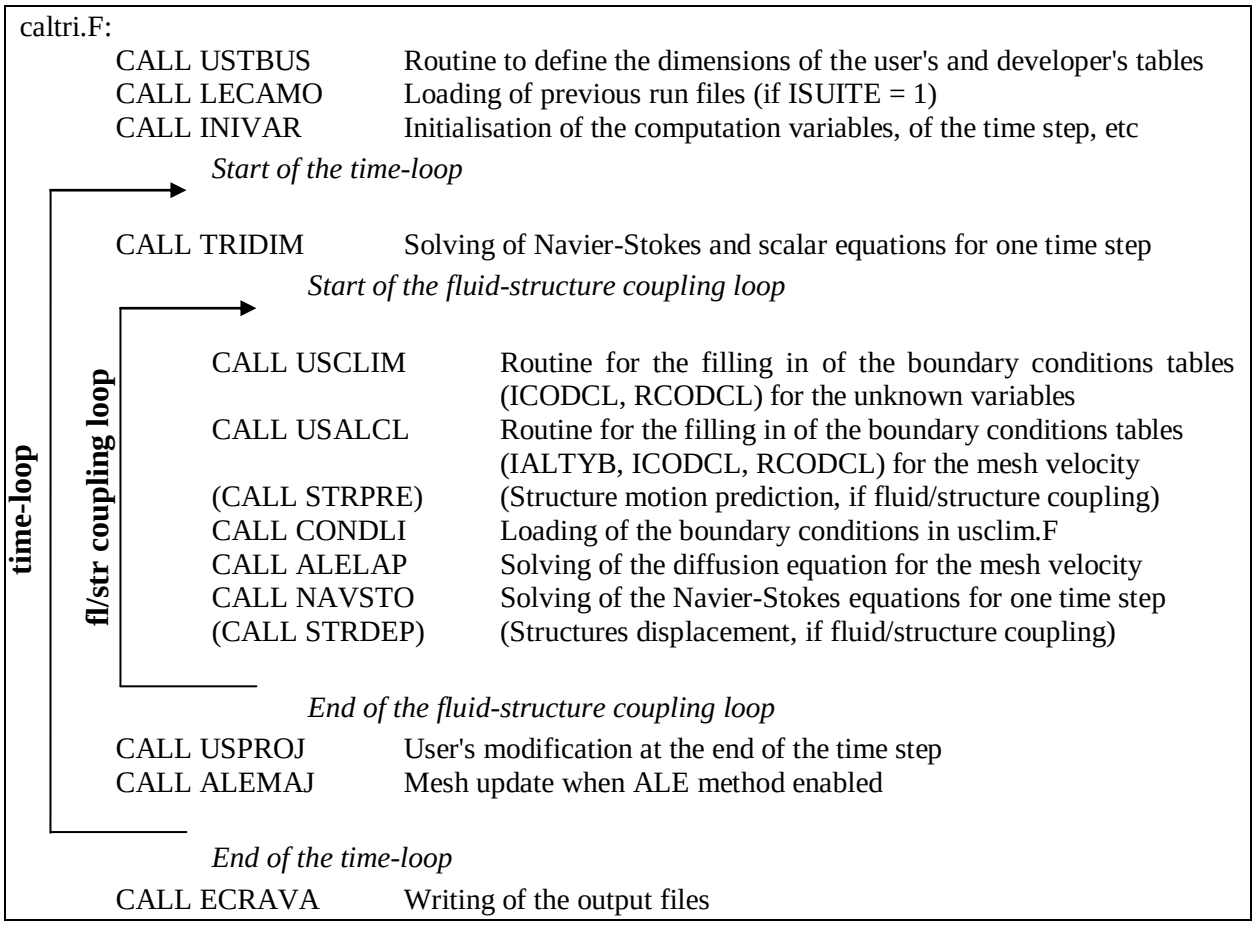

FIGURE 3.1.2-1: SUCCESSIVE STAGES WITHIN A TIME STEP (ALE ENABLED).

In the above calling tree, a fluid-structure coupling loop appears. This loop is necessary when the coupling between fluid forces and mobile structures is enabled; this allows the code to

iterate on the structures displacement until convergence. The free surface module is built in a similar way as explained in Chapter 4.

3.2. ALE module

The Arbitrary Lagrangian Eulerian (ALE) method has been introduced by e.g. Chan in [15]. A first version for an ALE module within *Code_Saturne* was developed in 1999 in what was then called "Solveur Commun", the prototype of *Code_Saturne* developed by EDF; this is specifically presented in the EDF report [16]. The current version of the ALE module is still very similar and was only used, up to now, for fluid coupling with a solid structure. Hereafter is a short summary of the method.

3.2.1. Navier-Stokes equations for a moving domain

In *Code_Saturne*, the Navier-Stokes equations for an incompressible flow (momentum (6) and mass (5) conservation with the velocity-pressure formulation) are, for a moving domain Ω:

$$\rho \frac{\partial \boldsymbol{u}}{\partial t} + \nabla.(\rho(\boldsymbol{u}-\boldsymbol{w}) \otimes \boldsymbol{u} - \mu \nabla \boldsymbol{u}) - \boldsymbol{u}\nabla.\left(\rho(\boldsymbol{u}-\boldsymbol{w})\right) = -\nabla P + \nabla.\left(\mu\ {}^{t}\nabla \boldsymbol{u}\right) + \rho \boldsymbol{g} \tag{11}$$

$$\nabla.\boldsymbol{u} = 0 \tag{12}$$

The variables $\boldsymbol{u}$ and P are the fluid velocity and pressure, $\boldsymbol{w}$ is the mesh velocity and $\boldsymbol{g}$ is the gravity vector. All velocities are defined in a Galilean reference frame (i.e. not tied to the mesh motion).

The mesh velocity field $\boldsymbol{w}$ appears: this term represents the moving nature of the global domain Ω for the current time step.

3.2.2. Mesh velocity computation

The computation of the mesh velocity is based on the solving of a Laplace equation: thanks to the imposed deformation on the moving boundary, the mesh velocity at the cell centres is computed within the internal domain.

$$\nabla.(\lambda \nabla \boldsymbol{w}) = 0 \tag{13}$$

$$\boldsymbol{w}_{\Gamma(t)} \text{ and } \boldsymbol{w}_{\partial\Omega/\Gamma(t)} \tag{14}$$

where $\Gamma(t)$ is the moving boundary and $\partial\Omega/\Gamma(t)$ is the remaining boundary of the domain Ω.

The boundary conditions for the mesh velocity can be value specified (i.e. Dirichlet condition – e.g. a fixed mesh on a side), or gradient specified (i.e. Neumann condition – e.g. when the mesh is allowed to slide along a wall).

3.2.3. Algorithm

The ALE algorithm within *Code_Saturne* is summarised here:

(i) The boundary conditions $\boldsymbol{w}_{\Gamma(t)}$ for the moving boundary (routine **usalcl.F**) are:

- known when the deformation is imposed by a law,
- extrapolated from the previous time values when there is no information about the border at the current time step (which is the case for free surface flows).

The boundary displacements are then known (or projected) at the centre of the border faces (routine **altycl.F**).

The mesh velocity system, a Laplace equation, is solved (routine **alelap.F**) in the known geometry of time step t^n.

(ii) The Navier-Stokes equations are solved in two steps (predictor-corrector algorithm, routine **navsto.F**) with the addition of two supplementary terms related to the mesh velocity in the prediction step. The equations are solved in the known geometry of time step t^n.

(iii) The geometry is updated (routine **alemaj.F**):

- the mesh velocity for the cell centres has been calculated (because of the finite volume approach); a value for the vertices displacement is then deduced,
- the vertices position ξ is updated,

$$\xi^{n+1} = \xi^n + \boldsymbol{w}^{n+1}(\xi)\Delta t$$

- at this point, the new geometry of the domain Ω^{n+1} and the associated values $\boldsymbol{u}^{n+1}$ and P^{n+1} are known – considering that $\boldsymbol{u}^{n+1}(\Omega^{n+1}) = \overline{\boldsymbol{u}}^{n+1}(\Omega^n)$ where $\overline{\boldsymbol{u}}^{n+1}$ is the approximation of the fluid velocity calculated at time step t^{n+1} in the known geometry of time step t^n.

3.2.4. Limitations

A - Discrete Geometric Conservation Law

Given that the ALE method requires a moving grid, we have to consider an additional law: the Discrete Geometric Conservation Law (DGCL), which is the discretized form of the Space Conservation Law (SCL). That is explained by J.H. Ferziger and M. Peric in [5].

For a moving domain of volume Ω and closed surface $\partial\Omega$ filled with an incompressible fluid, the SCL can be thought as the continuity equation (1) in the limit of zero fluid velocity: $\boldsymbol{u} = 0$. Then the continuity equation simplifies to the SCL:

$$\frac{d}{dt}\int_{\Omega} d\Omega - \int_{\partial\Omega} \boldsymbol{w}.\boldsymbol{n} dS = 0 \qquad (15)$$

where $\boldsymbol{w}$ is the mesh velocity and $\boldsymbol{n}$ the normal vector.

If we consider a 2D displacement, we can draw the following diagram (Figure 3.2.4-1):

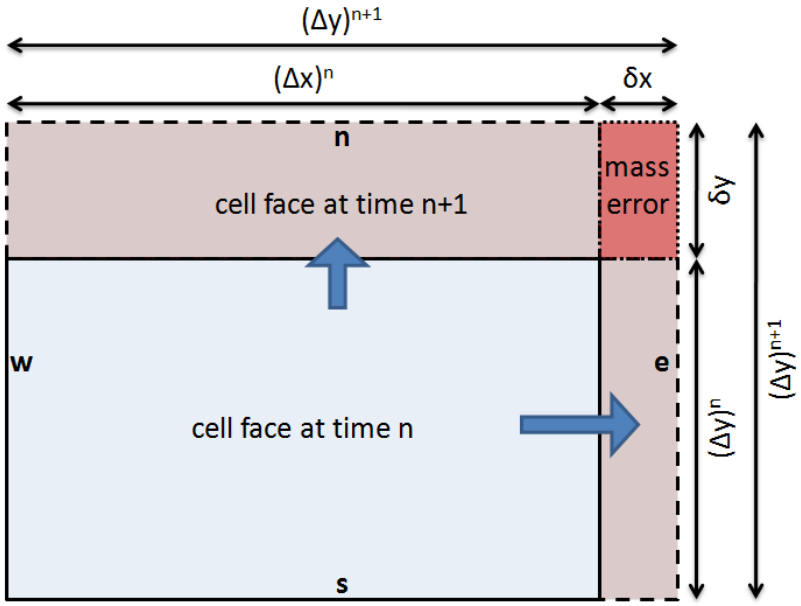

FIGURE 3.2.4-1: INCREASING 2D RECTANGULAR CONTROL VOLUME

Using the implicit Euler time scheme, a discretized form of the SCL is the DGCL:

$$\frac{(\Delta\Omega)^{n+1} - (\Delta\Omega)^{n}}{\Delta t} = \sum_{f} (\boldsymbol{w}.\boldsymbol{n})_f S_f \qquad (16)$$

where $f = e, w, n, s$ are the four sides of the 2D rectangular control volume.

According to the figure above:

$$(\Delta\Omega)^{n+1} = (\Delta x \Delta y)^{n+1}$$

$$(\Delta\Omega)^{n} = [(\Delta x)^{n+1} - \delta x][(\Delta y)^{n+1} - \delta y]$$

$$\sum_f (\boldsymbol{w}.\boldsymbol{n})_f S_f = (w_n - w_s)(\Delta x)^{n+1} + (w_e - w_w)(\Delta y)^{n+1} = \frac{\delta y}{\Delta t}(\Delta x)^{n+1} + \frac{\delta x}{\Delta t}(\Delta y)^{n+1}$$

There is a problem though in equation (16) given that an artificial mass source term appears:

$$\delta\dot{m} = \rho \frac{\delta x \delta y}{\Delta t} \tag{17}$$

This error $\delta\dot{m}$ disappears if the domain moves in only one direction (i.e. $\delta x = 0$ or $\delta y = 0$), or if the grid velocities are equal at opposite sides of the control volume (i.e. $w_n = w_s$ or $w_e - w_w$, then $\delta y = 0$ or $\delta x = 0$ respectively).

For most of the simulations we will test, the domain only moves vertically, that is to say in only one direction; this way, the SCL is always satisfied. However, it is interesting to check if *Code_Saturne* does meet the DGCL in the general case of a mesh moving in the three directions. With that aim, Charbel Farhat et al. propose an application with a uniform flow in [17].

This application consists in considering the case of a one-dimensional uniform flow $\boldsymbol{u} = \boldsymbol{u}^*$ at a Mach number $M_\infty = 0.2$ inside a rigid tube of length $L = 20\,m$ and a $1\,m\,x\,1\,m$ square cross section. The four lateral sides of the tube have symmetry-boundary conditions so that they do not have an influence on the uniform velocity field.

The computational domain is discretized by 200 equally spaced nodes in the direction x of the flow (lengthwise of the tube), and 10 equally spaced nodes in each of the y and z directions (square cross section of the tube). A first mesh is constructed by connecting these nodes with simple tetrahedral volumes. A random three-dimensional displacement $\Delta\boldsymbol{x}^0$ is then computed for every node and we use this random displacement $\Delta\boldsymbol{x}^0$ to perturb the initial position of the nodes according to

$$\Delta\boldsymbol{x}(t) = \Delta\boldsymbol{x}^0 \, sin(500\pi t)$$

without however creating any crossover.

The vibrating mesh is used to compute the time history of the flow and the relative error:

$$RellErr(t^n) = \frac{\|\boldsymbol{u}(t^n) - \boldsymbol{u}^*\|_2}{\|\boldsymbol{u}^*\|_2}$$

This relative error (a ratio of sums of absolute values so that random errors do not compensate each other) is compared for different values of the computational time step Δt.

If the scheme violates its DGCL, it will lead to a nonlinear instability with spurious oscillations appearing around the exact solution of the velocity field. The magnitude of these oscillations increases with the computational time step.

In *Code_Saturne*, the value $u^* = 1\ m.s^{-1}$ and the mesh described above were taken. For one computation, the following random deformation of the mesh appeared (Figure 3.2.4-2) and the following results were obtained for three different time step values (Figure 3.2.4-3):

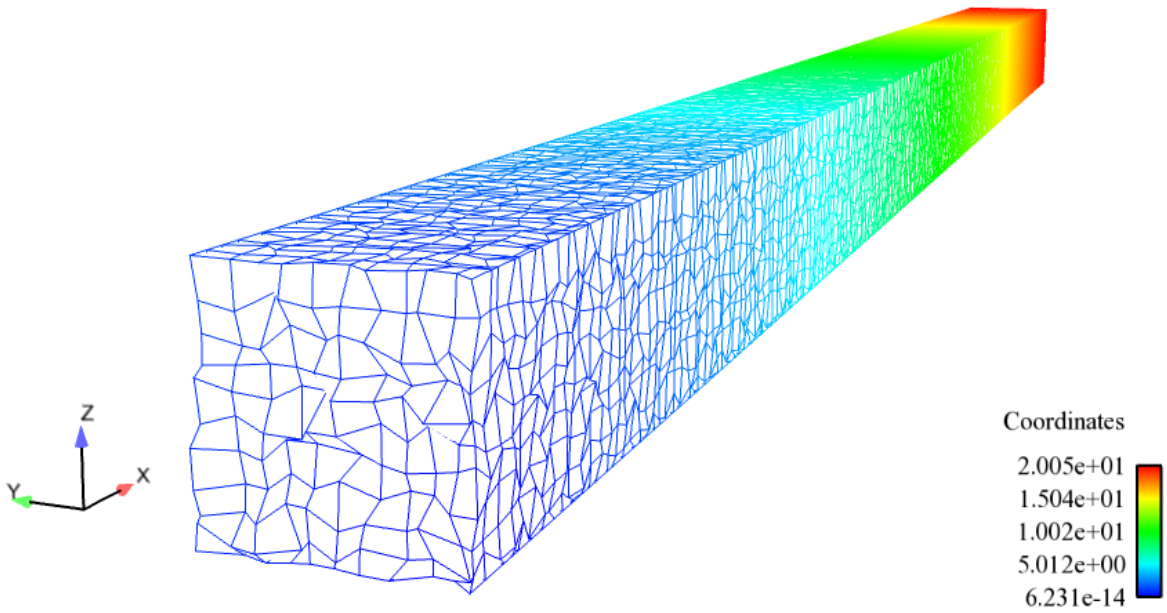

FIGURE 3.2.4-2: RANDOM DEFORMATION OF THE MESH

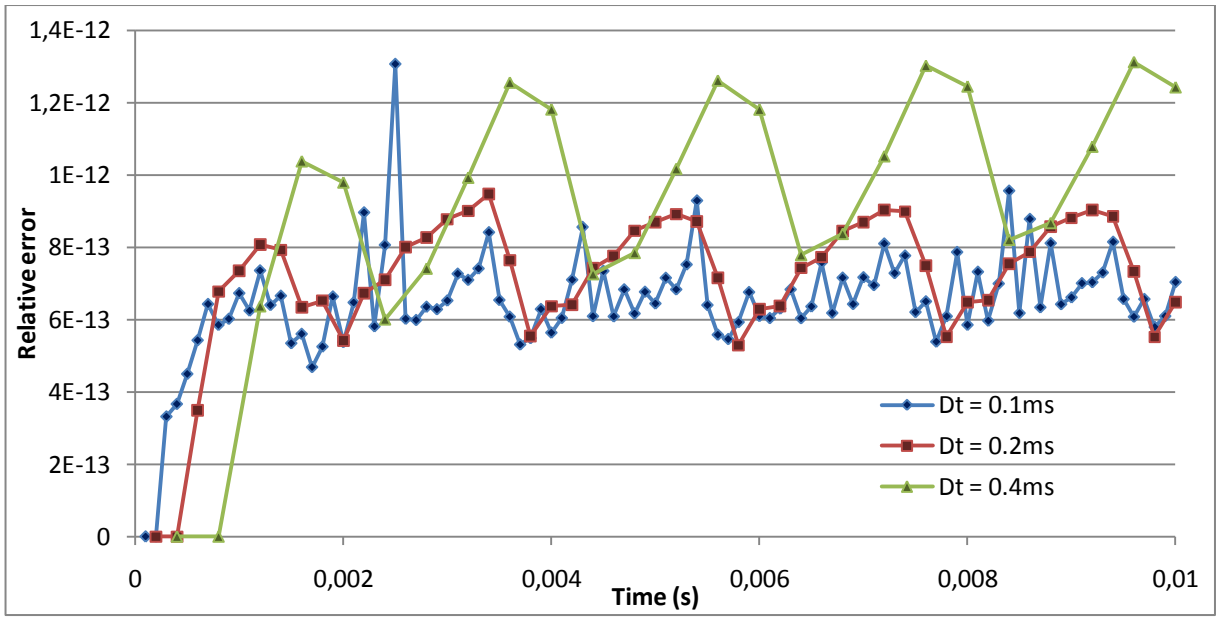

FIGURE 3.2.4-3: RELATIVE ERROR AS A FUNCTION OF TIME FOR THREE DIFFERENT TIME STEPS

Figure 3.2.4-3 shows that, for the three different time steps, there is no instability when the mesh is moved randomly in the three directions, the uniform flow is exactly predicted (indeed E-12 is negligible compared to the relative precision for the solution of the linear systems whose default value is E-8). In that case, it seems that *Code_Saturne* naturally satisfies its DGCL.

B - Diffusion equation for the mesh velocity solver

From the boundary conditions for the mesh velocity contained in the usalcl.F, the alelap.F routine solves a diffusion equation in order to know this mesh velocity on the whole domain. In reality different problems can occur: mesh crossover due to an excessive deformation, inaccuracy in the interpolation of the velocities, etc.

When a mesh crossover occurs, negative volumes appear and that stops the computation immediately. The occurrence of a crossover is often caused when the Courant number ($Co = \frac{u.\Delta t}{\Delta x}$ where u is the local fluid velocity, Δt is the time step and Δx is the average mesh-node spacing) is higher than one, but sometimes, the problem is more complicated: time step after time step, the global mesh gets increasingly distorted - even when the deformation is just in one direction - and that can lead to a mesh crossover. In that case, the longer a computation, the more likely this problem will emerge, but it is unfortunately impossible to predict beforehand.

The ALE module has another important limitation: the interpolation between face centre velocity and mesh nodes displacement is not accurate enough for a free surface application (the volume conservation requires the free surface to be moved precisely).

To test the accuracy of this interpolation, a simple case was run with a hexahedron as computational domain and the following boundary conditions for the mesh velocity:

- upper side: constant and uniform mesh velocity,
- lower side: homogeneous Dirichlet condition (fixed mesh),
- four remaining sides: slip condition.

This way the hexahedron is distorted and its volume has to increase as a linear function of time. For this test case, the initial mesh of the DGCL case is used and a constant and uniform vertical mesh velocity $w^* = 1m.s^{-1}$ is set at the face centres of the upper side. The physical time for the computation is 1s.

In the figures hereafter, the mesh velocity values at the face centres (Figure 3.2.4-4) and the nodes displacement (Figure 3.2.4-5) of the 3D hexahedron can be seen:

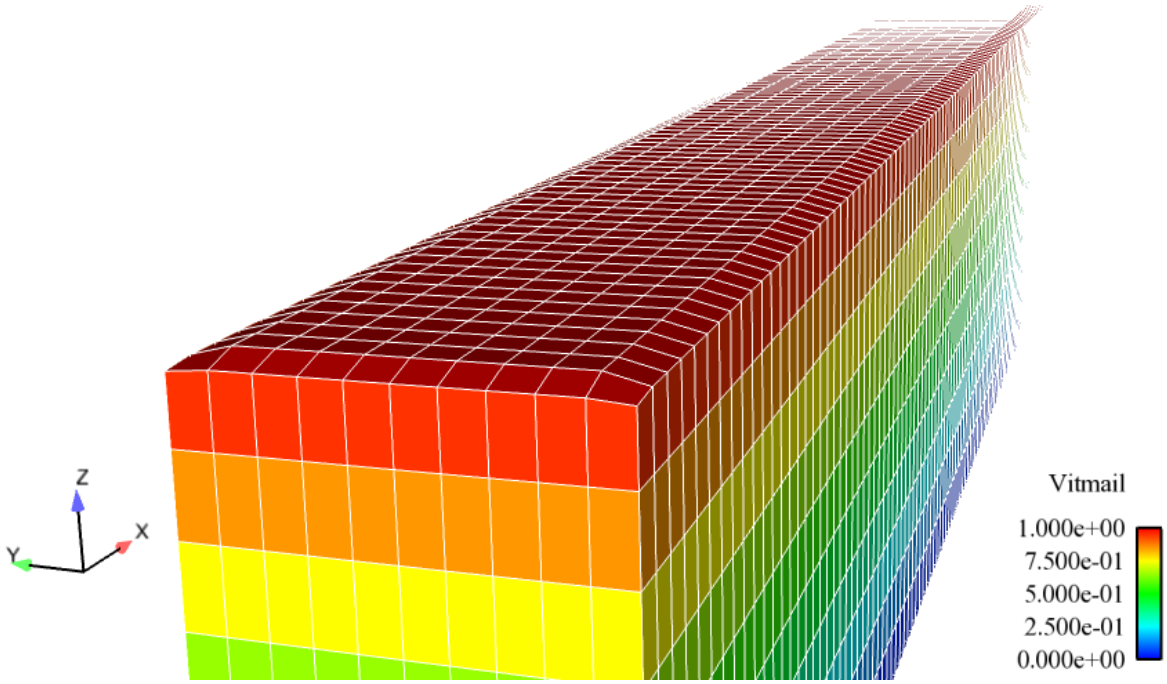

FIGURE 3.2.4-4: MESH VELOCITY VALUES FOR THE 3D GEOMETRY

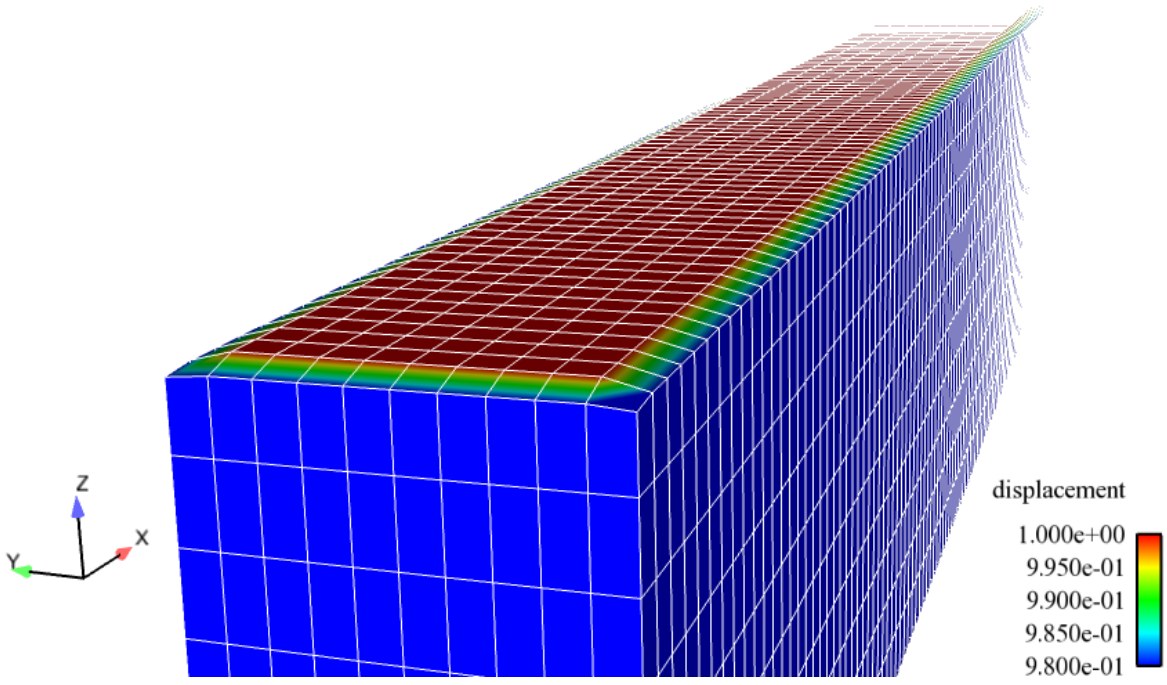

FIGURE 3.2.4-5: NODES DISPLACEMENT FOR THE 3D GEOMETRY

In this 3D case, the mesh velocities values are good but an edge effect can be noticed: the nodes displacement is not uniform on the moving side.

To understand the edge effect, it is interesting to run the same case but with a 2D computational domain (there is one single mesh in the third direction y). In the following

figures, the mesh velocity values at the face centres (Figure 3.2.4-6) and the nodes displacement (Figure 3.2.4-7) for the 2D hexahedron are displayed:

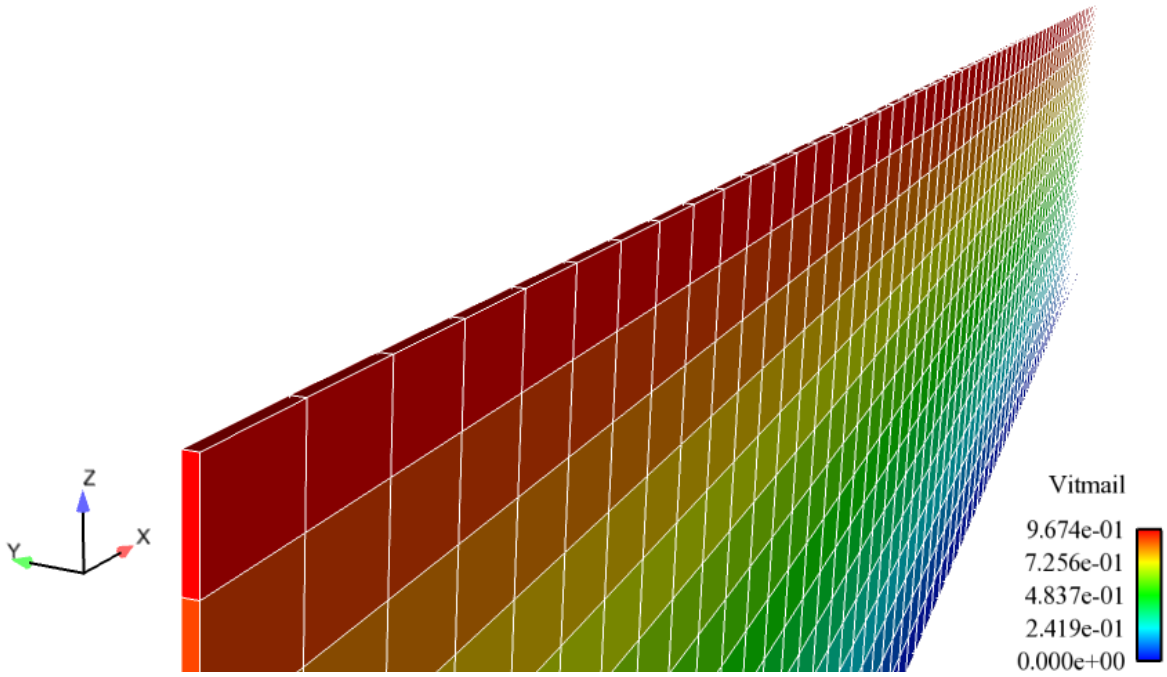

FIGURE 3.2.4-6: MESH VELOCITY VALUES FOR THE 2D GEOMETRY

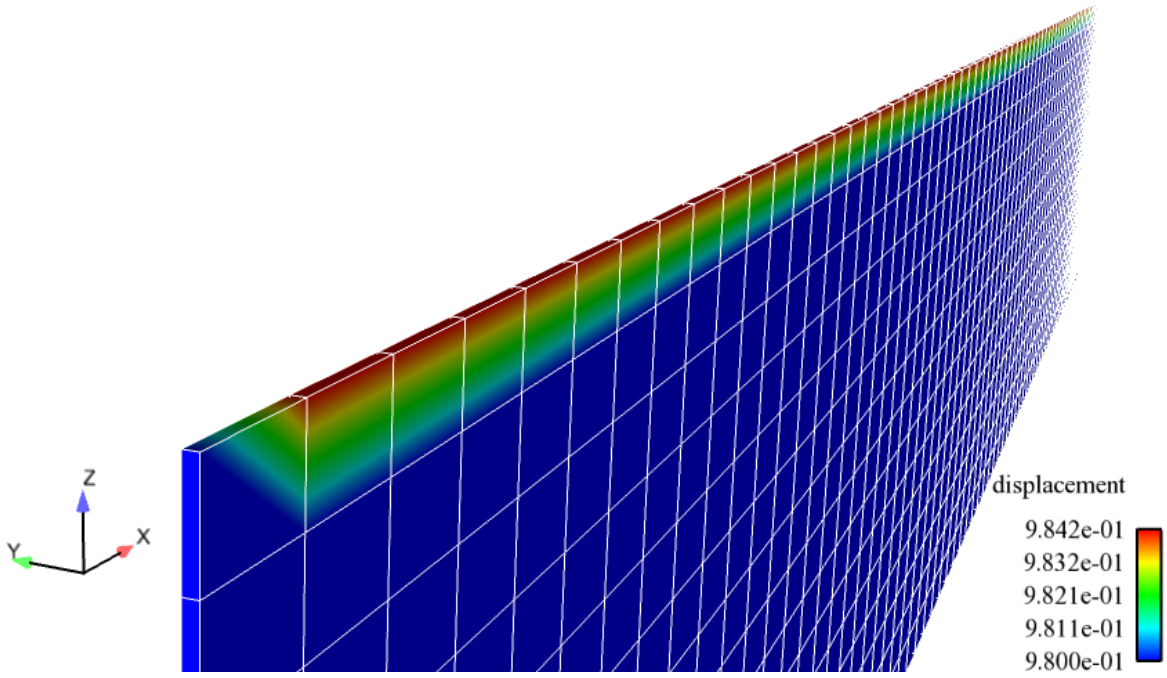

FIGURE 3.2.4-7: NODES DISPLACEMENT FOR THE 2D GEOMETRY

As we can see here, because of the edge effect, all the mesh velocities are strictly less than w^* and the nodes displacement is not uniform on the moving side.

As a consequence, when imposing the mesh velocity at the face centres of the moving side (constant RCODCL(IFAC,.,1)), the global volume does not increase as expected; whereas

when calculating and imposing the displacement for each of the free surface nodes (DEPALE(INOD,.), the value of the global volume is correct. And that is particularly true for 2D test cases.

Therefore, to avoid mesh crossovers and interpolation errors, it will be necessary to control precisely the nodes displacement. This will be discussed further in Chapter 4.

In the following paragraph, the ALE module already embedded in *Code_Saturne* will be adapted to the free surface flows case.

Chapter 4

Free Surface module

4.1. Method

The handling of free surface flows using the ALE method has already been done by Souli and Zolesio in a finite element formulation [18]; a similar approach with a finite volume formulation was applied by Demirdzic and Peric in [19]. Given the existing ALE module within *Code_Saturne*, a free surface module is going to be implemented. With that aim, we can start considering the special boundary conditions for the free surface.

For the dynamic boundary condition (8) ($p = p_{atm}$ on the free surface), we only need to set a Dirichlet condition for the pressure.

The kinematic boundary condition (9) is more complicated: we need to move each face of the free surface in order to maintain a zero net mass flux through this face. In section 2.2, it was written that for incompressible flows with a one-dimensional mesh velocity (we consider that the mesh only moves in the vertical direction $\boldsymbol{w} = w.\boldsymbol{e_z}$), the kinematic boundary condition can be reduced to:

$$w = \frac{\boldsymbol{u}.\boldsymbol{S}}{\boldsymbol{e_z}.\boldsymbol{S}} = \frac{\dot{m}_{fs}}{\rho S_z} \tag{18}$$

For each face of the free surface, three elements are needed to compute the face velocity w:

- the mass flow values through the free surface $\dot{m}_{fs}$,
- the vertical component of the surface vector S_z,
- and the density ρ.

The last one is easy to know: ρ is a constant for an incompressible flow.

For the mass flow and the surface vector, they vary with time and space position: their values change, time step after time step, and depend on the position on the free surface.

First, to prevent the mass flow values through the free surface from being affected by the free surface boundary conditions, a free outlet has to be set, with a Neumann condition for the velocities.

4.1.1. Convergence loop

In David Apsley's paper [20], two strategies are presented to enforce the kinematic boundary condition numerically. For simplicity, both strategies assume that the displacement of the cell vertices or control points is only vertical.

One of these strategies consists in moving the free-surface (by moving either cell vertices or face-centre control points) according to:

$$S_z \frac{\Delta h}{\Delta t} = (\boldsymbol{u}.\boldsymbol{S})_{fs} \tag{19}$$

where Δh is the average height increment over the free-surface cell face for the whole time step Δt. This equation (19) is equivalent to the kinematic boundary condition (18) because we can express the free-surface face velocity as $w = \Delta h/\Delta t$.

Given that the free-surface displacement Δh depends on the fluid velocity $\boldsymbol{u}$ (whose value varies during the solving of the Navier-Stokes equations), the free-surface face velocity w is then performed incrementally within each time step.

So, following this strategy, a convergence loop on the mass flow values through the free-surface faces has been implemented. This loop is clearly visible in the representation of the free surface algorithm hereafter (Figure 4.1.1-1):

Data: Values of time step t^n: m_{fs}^n, w^n, P^n, $\underline{u}^n$, S_z^n

begin

Loop initialization: $m_{fs_0}^{n+1} = m_{fs}^n$, $w_0^{n+1} = w^n$, $P_0^{n+1} = P^n$, $\underline{u}_0^{n+1} = u^n$

call usalcl: computation of the free surface velocity $w_1^{n+1} = \dfrac{m_{fs_0}^{n+1}}{\rho S_z^n}$

call fssave: save of the initial boundary conditions for P and $\underline{u}$

call navsto: prediction of $m_{fs_1}^{n+1}$, P_1^{n+1} and $\underline{u}_1^{n+1}$

- **call preduv**: computation of the predicted velocity $\underline{u}_1^*$ solving
  $\rho^n \dfrac{\underline{u}_1^* - \underline{u}^n}{\Delta t^n} + \mathrm{div}(\rho^{n-1}(\underline{u}^n - \underline{w}^n) \otimes \underline{u}_1^* - \mu\, \underline{\underline{\mathbf{grad}}}\, \underline{u}_1^*) =$
  $-\underline{\mathbf{grad}}\, P^n + \mathrm{div}(\mu\, {}^t\underline{\underline{\mathbf{grad}}}\, \underline{u}^n) + \rho^n \underline{g} + \underline{u}_1^* \mathrm{div}(\rho^{n-1}(\underline{u}^n - \underline{w}^n))$
- **call inimas**: initialization of the mass flow values $\mathrm{div}(\rho^n \underline{u}_1^*)$
- **call resolp**: computation of the pressure correction δP_1 solving
  $\mathrm{div}(-\Delta t^n \underline{\mathbf{grad}}\, \delta P_1) = -\mathrm{div}(\rho^n \underline{u}_1^*)$ and update of the pressure $P_1^{n+1} = P^n + \delta P_1$
- **call itrmas**: update of the mass flow values through the free surface faces
  $m_{fs_1}^{n+1} = \mathrm{div}(\rho^n \underline{u}_1^{n+1}) = \mathrm{div}(\rho^n \underline{u}_1^*) - \mathrm{div}(\Delta t^n \underline{\mathbf{grad}}\, \delta P_1)$
- **call inimas**: update of the mass flow values $\mathrm{div}(\rho^n(\underline{u}_1^{n+1} - \underline{w}_1^{n+1}))$
- Update of the velocity field $\underline{u}_1^{n+1} = \underline{u}_1^* - \dfrac{\Delta t^n}{\rho^n} \underline{\mathbf{grad}}\, \delta P_1$

$m = 1$

while $\left| \dfrac{m_{fs_m}^{n+1} - m_{fs_{m-1}}^{n+1}}{m_{fs_m}^n} \right|_\infty > \epsilon_{fs}$ ***and*** $m \leq NbIt_{fs}$ **do**

call fsload: reloading of P^n, $\underline{u}^n$ and their initial boundary conditions

call usalcl: computation of $w_{m+1}^{n+1} = \dfrac{m_{fs_m}^{n+1}}{\rho S_z^n}$

call navsto: prediction of $m_{fs_{m+1}}^{n+1}$, P_{m+1}^{n+1} and $\underline{u}_{m+1}^{n+1}$

- **call preduv**: computation of the predicted velocity $\underline{u}_{m+1}^*$ solving
  $\rho^n \dfrac{\underline{u}_{m+1}^* - \underline{u}^n}{\Delta t^n} + \mathrm{div}(\rho^n(\underline{u}_m^{n+1} - \underline{w}_m^{n+1}) \otimes \underline{u}_{m+1}^* - \mu\, \underline{\underline{\mathbf{grad}}}\, \underline{u}_{m+1}^*) =$
  $-\underline{\mathbf{grad}}\, P^n + \mathrm{div}(\mu\, {}^t\underline{\underline{\mathbf{grad}}}\, \underline{u}^n) + \rho^n \underline{g} + \underline{u}_{m+1}^* \mathrm{div}(\rho^n(\underline{u}_m^{n+1} - \underline{w}_m^{n+1}))$
- **call inimas**: initialization of the mass flow values $\mathrm{div}(\rho^n \underline{u}_{m+1}^*)$
- **call resolp**: computation of the pressure correction δP_{m+1} solving
  $\mathrm{div}(-\Delta t^n \underline{\mathbf{grad}}\, \delta P_{m+1}) = -\mathrm{div}(\rho^n \underline{u}_{m+1}^*)$
  and update of the pressure $P_{m+1}^{n+1} = P^n + \delta P_{m+1}$
- **call itrmas**: update of the mass flow values through the free surface faces
  $m_{fs_{m+1}}^{n+1} = \mathrm{div}(\rho^n \underline{u}_{m+1}^{n+1}) = \mathrm{div}(\rho^n \underline{u}_{m+1}^*) - \mathrm{div}(\Delta t^n \underline{\mathbf{grad}}\, \delta P_{m+1})$
- **call inimas**: update of the mass flow values $\mathrm{div}(\rho^n(\underline{u}_{m+1}^{n+1} - \underline{w}_{m+1}^{n+1}))$
- Update of the velocity field $\underline{u}_{m+1}^{n+1} = \underline{u}_{m+1}^* - \dfrac{\Delta t^n}{\rho^n} \underline{\mathbf{grad}}\, \delta P_{m+1}$

$m = m + 1$

At this point, $m_{fs}^{n+1} = m_{fs_m}^{n+1}$, $w^{n+1} = w_m^{n+1}$, $P^{n+1} = P_m^{n+1}$ and $\underline{u}^{n+1} = \underline{u}_m^{n+1}$

call alemaj: computation of S_z^{n+1} during the mesh update

Result: Values of time step t^{n+1}: m_{fs}^{n+1}, w^{n+1}, P^{n+1}, $\underline{u}^{n+1}$, S_z^{n+1}

FIGURE 4.1.1-1: PRESENTATION OF THE FREE SURFACE ALGORITHM

Within tridim.F, there is already an optional loop for the fluid/structure coupling (strpre.F and strdep.F); a similar structure is used for the free surface convergence loop.

For this purpose, we need:

- Two new routines:
    - fssave.F to save the boundary conditions values for the velocities and the pressure at the beginning of the time step,
    - fsload.F to load these initial boundary conditions values for the velocities and the pressure between two iterations in the convergence loop. To avoid endless iterations, a convergence test has to be added here.
- Three new parameters:
    - ACTIFS to enable the convergence loop on the mass flow values through the free surface faces,
    - NBITFS to limit the maximum number of iterations within the loop,
    - EPALFS to test the convergence criterion and end the loop prematurely.
- One variable: ITERFS to know the number of the current iteration within the convergence loop, and possibly stop the loop.
- Several modifications in other routines:
    - tridim.F to create the structure of the free surface loop,
    - navsto.F to save the mass flow values through the free surface faces. These are the values of the mass flows which will be used to compute the free surface mesh velocities; the convergence test will also compare these values between two consecutive iterations.

Some of these modifications can be seen in the Figure 4.1.1-1.

4.1.2. Free-surface cell-vertices displacement

We saw in the section 3.2.4.B that it is better to impose the cell-vertices displacement on the free surface. Indeed the interpolation done by *Code_Saturne* between the free-surface cell-face centre velocities and the free-surface cell-vertices displacement is not accurate enough; therefore, it seems wiser, for now, to compute directly the displacement of these cell-vertices, without using *Code_Saturne* solver. For this purpose, there are two main possibilities:

- computing straight away an explicit free-surface cell-vertices displacement thanks to the free-surface cell-face centre velocities *w*; this scheme does not meet the local volume conservation, but does meet the global one.
- or, to compute an implicit free-surface cell-vertices displacement by solving a system of equations between free-surface cell-vertices displacement and free-surface cell-face centre velocities; this scheme does verify the local volume conservation and thus the global one.

Figure 4.1.2-1 is a diagram showing the interpolation between free-surface cell-face centre velocities and free-surface cell-vertices displacement:

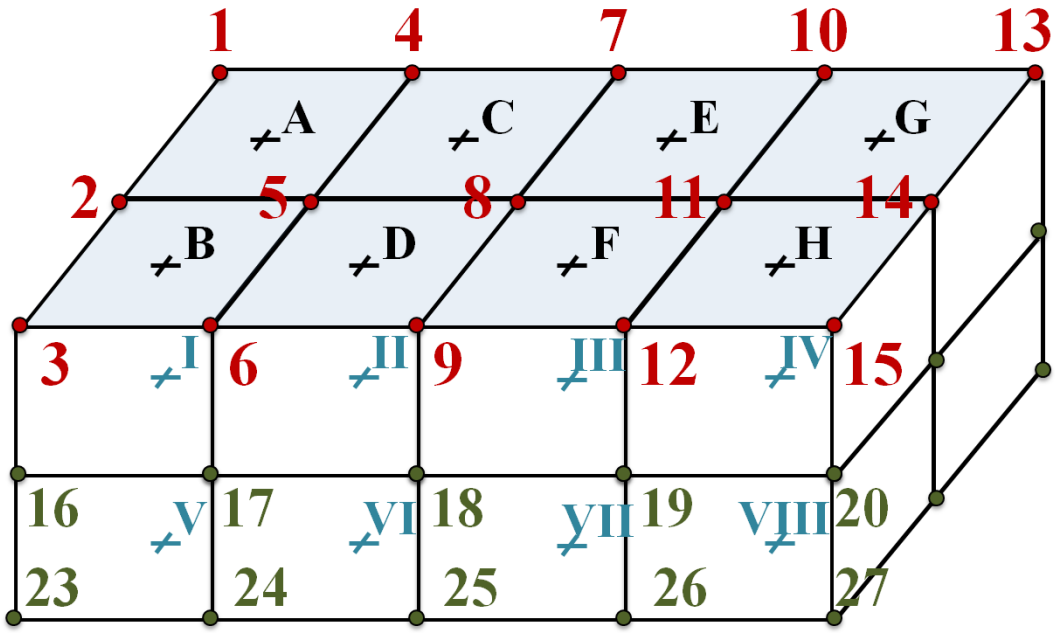

FIGURE 4.1.2-1: CELL-CENTRES, CELL-FACE CENTRES AND CELL-VERTICES LOCATION

In this figure, red and green points represent cell-vertices (i.e. the "corners" of cells, defining the mesh and its displacement; red point for a free-surface cell-vertex, green point for an internal cell-vertex), black crosses locate free-surface cell-face centres and blue crosses point to cell-centres (where all discrete values of variables are computed).

In the following table, index $i = 1,2,3, \ldots$ represents a free-surface cell-vertex, index $I = A, B, C, \ldots$ represents a free-surface cell-face centre; therefore V_1 is the velocity of the free-surface cell-vertex 1 and V_A is the velocity of the free-surface cell-face centre A.

Explicit scheme	Implicit scheme
On a corner: $V_i = V_I$ e.g. $V_1 = V_A$ On a side: $V_i = \frac{V_I+V_{II}}{2}$ e.g. $V_2 = \frac{V_A+V_B}{2}$ Otherwise: $V_i = \frac{V_I+V_{II}+V_{III}+V_{IV}}{4}$ e.g. $V_5 = \frac{V_A+V_B+.V_C+V_D}{4}$	$V_I = \frac{V_i+V_{i+1}+V_{i+2}+V_{i+3}}{4}$ e.g. $V_A = \frac{V_1+V_2+V_4+V_5}{4}$ That gives us (n-1)*(N-1) equations (one equation for each free-surface face centre); but there are n*N free-surface vertices. ➔ We need to add n+N-1 conditions in order to solve the system (e.g. $V_1 = V_A$).

4.1.3. Internal cell-vertices displacement

For the internal cell-vertices, as seen in the section 3.2.4.B, in order to avoid mesh crossovers, it is better to impose the displacement of each cell-vertex. To do that, each internal cell-vertex i below a free-surface cell-vertex i_{max} can be displaced according to the initial ratio between their two heights: $V_i = V_{i_{max}} \frac{Z_i}{Z_{i_{max}}}$ (e.g. $V_{16} = V_3 \frac{Z_{16}}{Z_3}$). This way, crossovers should be avoided.

4.2. Features

In this section, the main features of the free surface module are presented:

- Embedded module in the *Code_Saturne* 1.3.3 version.
- Activation variable for the free surface module ACTIFS: when enabled, *Code_Saturne* is able to compute free surface flows, otherwise it's just the "classic" *Code_Saturne* (there is no side effects due to the new module implementation).
- Control of the free surface convergence loop thanks to the convergence accuracy EPALFS and the max iteration number NBITFS parameters.
- Time scheme selection (second-order Crank-Nicolson method or first-order implicit Euler method) is available.
- Parallel computation available (at least partially).
- 3D computation available with the explicit prediction of the free-surface cell-vertices displacement.

4.3. Implementation of the new module

To implement the free surface module, new routines were created and some routines of *Code_Saturne* were modified. The main modifications (*) within the routines are listed in *Appendix 2: Implementation of the new module*.

* All the modifications in non-user routines can be found, by searching for "C MOD OLIVIER" in the body of the file.

Chapter 5

Application of the new module to different test cases

5.1. Standing wave

The first test problem considered in the present study is a standing wave in a water tank; this is a common test case for the development of free surface codes because, for low-amplitude waves, an analytic solution for the wave shape exists. Indeed, this case was previously run in 1999 with the "Solveur Commun", a prototype of *Code_Saturne* developed by EDF (see the EDF report [21]).

5.1.1. Presentation

The test case deals with the standing wave motion in a tank of length L. The shape is two-dimensional and initially set to:

$$y(x,t) = h + A\cos\left(\frac{2\pi x}{\lambda}\right) + \frac{\pi A^2}{\lambda}\left[1 - \frac{1}{4\,ch^2\left(\frac{2\pi h}{\lambda}\right)} + \frac{3}{4\,sh^2\left(\frac{2\pi h}{\lambda}\right)}\right]\cos\left(\frac{4\pi x}{\lambda}\right) \quad (20)$$

where:

- h = 15 m is the tank depth,
- L = 106.146 m represents the tank length,
- A = 2 m is the wave amplitude,
- λ = L = 106.146 m is the wave length.

Figure 5.1.1-1 shows the initial shape set for the free surface in this test case:

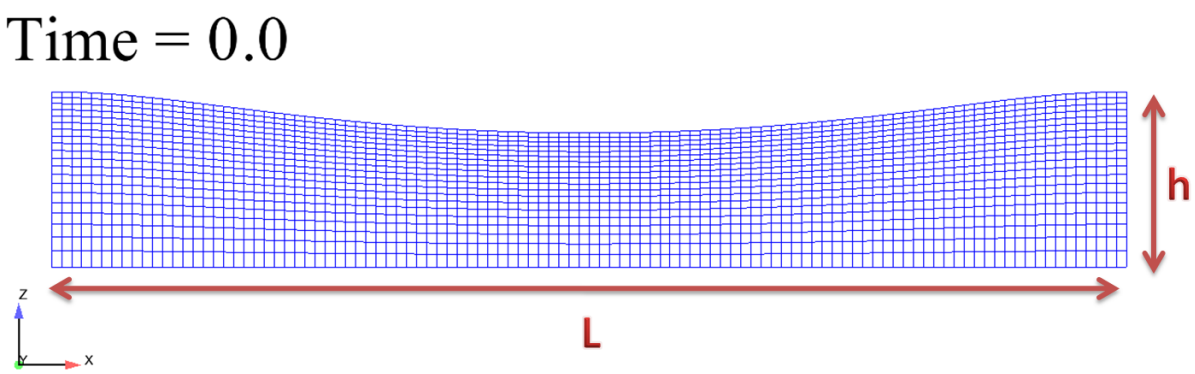

FIGURE 5.1.1-1: INITIAL SHAPE FOR THE STANDING WAVE TEST CASE

Thus a standing wave is created, and considering the inviscid theory, the Chabert d'Hières' formula presented in [22] gives a second order approximation in A of the wave shape:

$$y(x,t) = h + A\cos\left(\frac{2\pi x}{\lambda}\right)\cos\left(\frac{2\pi t}{T}\right) + \frac{\pi A^2}{\lambda}\left[\cos^2\left(\frac{2\pi t}{T}\right) - \frac{1}{4\,ch^2\left(\frac{2\pi h}{\lambda}\right)} + \frac{3\cos\left(\frac{4\pi t}{T}\right)}{4\,sh^2\left(\frac{2\pi h}{\lambda}\right)}\right]\cos\left(\frac{4\pi x}{\lambda}\right) \quad (21)$$

where $T = \frac{\lambda}{c}$ is the time period and c is the wave celerity, given by the Airy's formula:

$$c^2 = g\frac{\lambda}{2\pi}th\left(\frac{2\pi h}{\lambda}\right) \quad (22)$$

In this case, the numerical application gives $T = 9.783$ s.

5.1.2. Physical characteristics

The test case is run with the following physical characteristics:

- fluid density: $\rho = 10^3$ kg. m^{-3} (water value),
- fluid viscosity: $\mu = 10^{-12}$ kg. m^{-1}. s^{-1} (small enough to consider an inviscid fluid),
- gravity: $g = 9.81$ m. s^{-2},
- Reynolds number: $R_e \to \infty$ for an inviscid fluid.

5.1.3. Mesh characteristics

The mesh is structured and composed by hexahedra; the mesh spacing is constant in the horizontal direction, variable in the vertical direction and has only one cell in the third

direction (0.1 m). Three meshes were used: 106*1*20, 212*1*40 and 424*1*80 for the x, y and z directions respectively.

5.1.4. Boundary conditions

The boundary conditions for the fluid velocity are:

- free surface: homogeneous Neumann condition,
- tank bottom: homogeneous Dirichlet condition,
- walls: slip condition.

The boundary conditions for the mesh velocity are:

- free surface: Dirichlet condition according to the formula (9) page 18,
- tank bottom: homogeneous Dirichlet condition,
- walls: slip condition.

The boundary conditions for the pressure are:

- free surface: homogeneous Dirichlet condition according to the formula (8) page 18,
- tank bottom: Neumann condition,
- walls: Neumann condition.

5.1.5. Main computations

All the computations here were run with *Code_Saturne* version 1.3.3 using only one core of a quad-core processor (Intel Xeon processor at 2.80 GHz with 4 GB of system RAM available); the laminar turbulence model and a second order time scheme were adopted. The most significant ones are listed hereafter and will be presented in the following paragraph of results.

Computation name	Time step (constant) Δt	Horizontal discretization	Vertical discretization	Physical time	Maximum Courant number
Dt=200ms - 106*20	200 ms	106	20	1000 s	$Co = 0.4$
Dt=100ms - 106*20	100 ms	106	20	1000 s	$Co = 0.2$
Dt=50ms - 106*20	50 ms	106	20	1000 s	$Co = 0.1$
Dt=20ms - 106*20	20 ms	106	20	1000 s	$Co = 0.04$
Dt=50ms - 212*40	50 ms	212	40	200 s	$Co = 0.2$

Dt=25ms - 424*80	25 ms	424	80	200 s	$Co = 0.2$
Theoretical height	These are the analytical results of the Chabert d'Hières' formula				

The maximum Courant number $Co_{max} = \frac{u_{max}.\Delta t}{\Delta x}$ is done by *Code_Saturne* using the computed values of the fluid velocities. When the maximum Courant number is larger than 1 (for example $\Delta t = 500$ ms with the 106*20 mesh), the computation tends to crash quickly.

5.1.6. Results

The Figure 5.1.6-1 and Figure 5.1.6-2 show the free surface shape, pressure and velocity fields at the physical times $t_1 = 875$ s and $t_2 = 950$ s respectively (for the computation: Dt=100ms - 106*20), after 90 time periods approximately. In both figures, the pressure and velocity fields seem to be physically right while the free surface shapes are in good agreement with the second order approximation in amplitude solution given by the formula (21) of Chabert d'Hières.

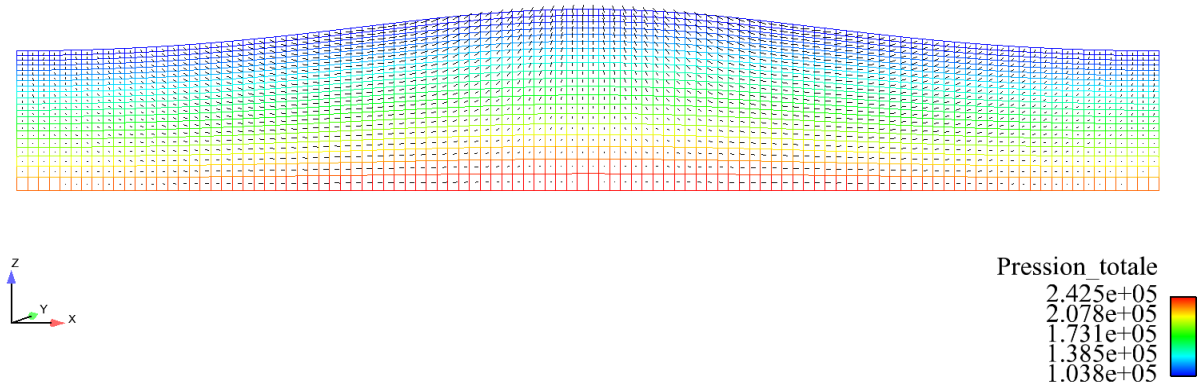

FIGURE 5.1.6-1: FREE SURFACE SHAPE, PRESSURE AND VELOCITY FIELDS AT $T_1 = 875$ S

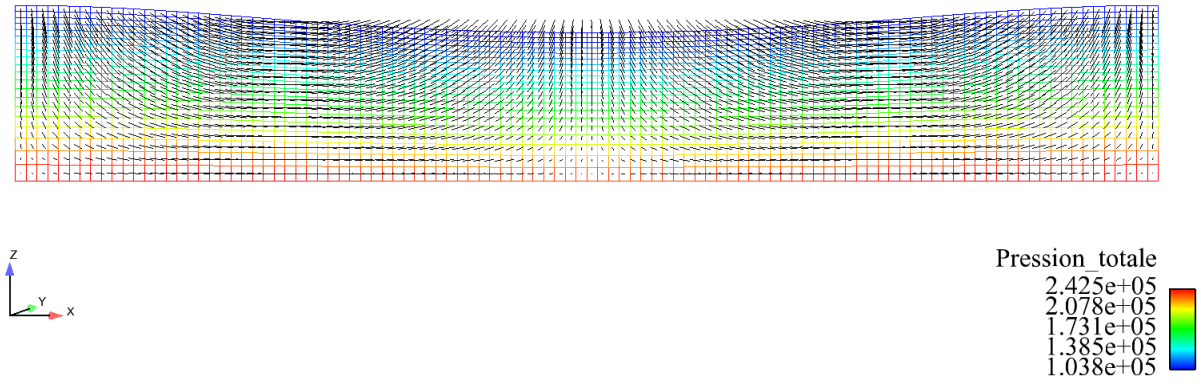

FIGURE 5.1.6-2: FREE SURFACE SHAPE, PRESSURE AND VELOCITY FIELDS AT $T_2 = 950$ S

The L2 error between the free surface height h and its theoretical value h_{th} (given by the Chabert d'Hières' formula) is computed on the whole free surface:

$$Error_{L2} = \frac{\sum_{fs\_vertices}(h - h_{th})^2}{\sum_{fs\_vertices} {h_{th}}^2}$$

where the two heights are compared at each free surface vertex.

The Figure 5.1.6-3 presents, for a fixed mesh and different time steps, this L2 error as a function of time. The results are quite good: the worst value is a L2 error of 0.6% after 1000 s of physical time, i.e. more than 100 time periods. That means:

- that the time period observed in the results is really close to the theoretical value of the Airy's formula (22),
- and that the free surface shape is very similar to the theoretical shape given by the Chabert d'Hières' formula (21).

This figure also shows that the L2 error tends to decrease when the time step gets smaller: after 1000 s of physical time, the L2 error ranges from 0.6% for a time step $\Delta t = 200$ ms to 0.04% for a time step $\Delta t = 20$ ms. When the time step decreases, the L2 error seems to reduce to a non-zero value which increases with elapsed time; maybe this is caused by the fact that the Chabert d'Hières' formula is a second order approximation in amplitude of the free surface shape and not an exact solution. Further analysis is then required to confirm convergence.

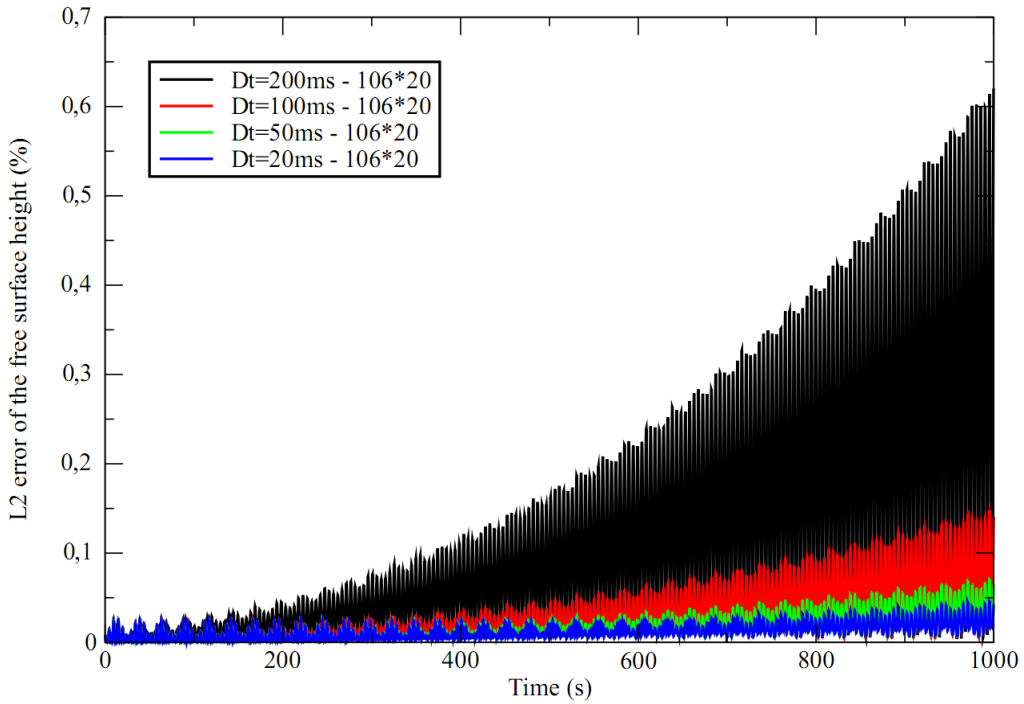

FIGURE 5.1.6-3: L2 ERROR OF THE FREE SURFACE SHAPE AS A FUNCTION OF TIME

Figure 5.1.6-4 is the log-log plot of Figure 5.1.6-3. The equation for a line of slope m on a log-log scale is $\log_{10}(F(t)) = m \log_{10}(t) + b$, thus $F(t) = t^m 10^b$ on a linear scale.

In the log-log scale, the L2 error tends to increase linearly with time: for the fixed mesh 106*20 and the different time steps, the slope is quite constant: $m \approx 2$. That means that the L2 error increases with time squared: $Error_{L2} = C\ t^2$.

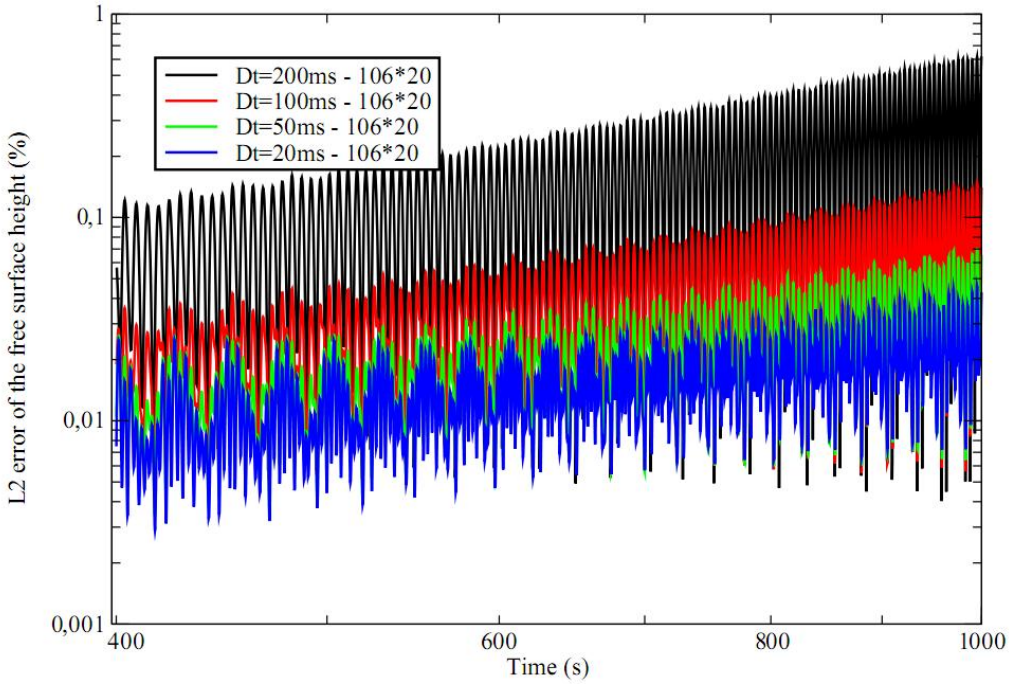

FIGURE 5.1.6-4: L2 ERROR OF THE FREE SURFACE SHAPE (LOG-LOG SCALE)

The relative error of global volume is given by:

$$Rel\_Err\_Vol = \frac{V(t) - V_0}{V_0}$$

where V_0 is the initial volume of water in the tank.

The Figure 5.1.6-5 presents, for a fixed mesh and different time steps, this relative error of global volume as a function of time. The worst value is a relative increase in volume of 5e-7 after 1000 s of physical time: this is a really small value, close to the numerical error of the code given by the relative precision for the solution of the linear system (the default value is 1e-8). We can therefore conclude that, in this test case, the free surface module is volume conservative (and then mass conservative given that an incompressible flow is considered).

This figure also shows that the variations of the relative error of global volume as a function of the time step value are complex: with a small time step $\Delta t = 20$ ms, the volume tends to decrease slightly whereas for a time step $\Delta t = 100$ ms, the volume increases.

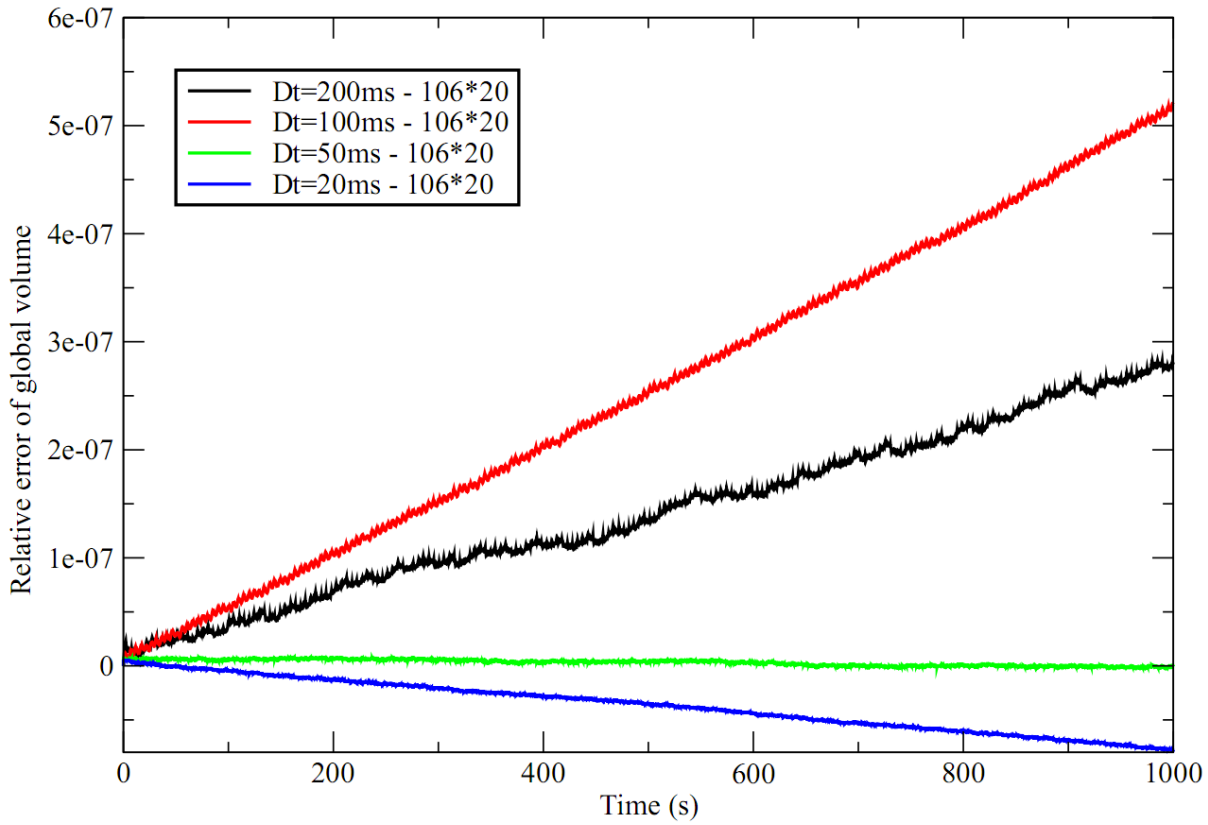

FIGURE 5.1.6-5: RELATIVE ERROR OF GLOBAL VOLUME AS A FUNCTION OF TIME

The relative error of global energy is given by:

$$Rel\_Err\_en = \frac{E(t) - E_0}{E_0}$$

where:

- E_0 is the initial energy of the water in the tank: $E_0 = E_{p_0}$,
- $E(t)$ is the energy of the water in the tank at the physical time t: $E(t) = E_p(t) + E_{kin}(t)$,

The Figure 5.1.6-6 presents, for a fixed mesh and different time steps, this relative error of global energy as a function of time. The worst value is a relative increase in energy of 0.3% after 1000 s of physical time. Because this value is quite important, we can conclude that the free surface module is not strictly conservative for the energy. This is logical considering that, in *Code_Saturne*, no equation for the energy conservation is solved (even the momentum conservation equation is solved in a way that is not strictly conservative).

This figure also shows that the variations of the relative error of global energy as a function of the time step value are complex: with a big time step $\Delta t = 200$ ms, the energy tends to increase whereas for a time step $\Delta t = 20$ ms, some energy is lost.

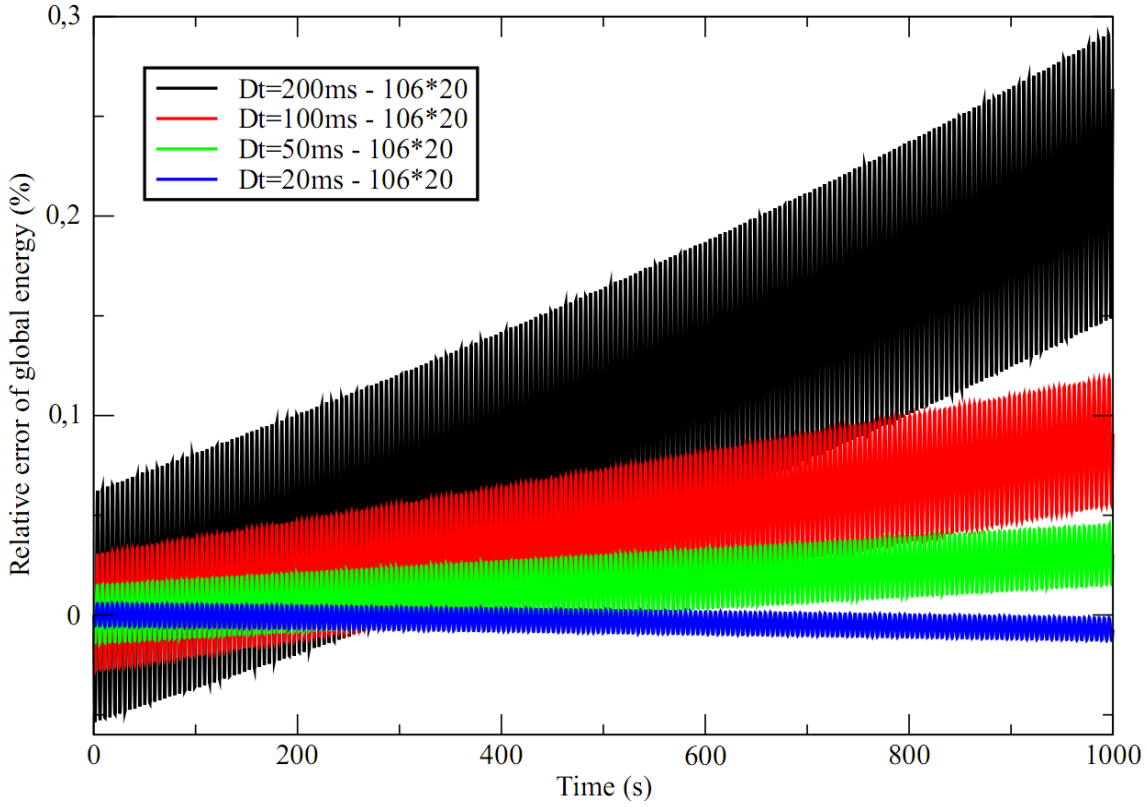

FIGURE 5.1.6-6: RELATIVE ERROR OF GLOBAL ENERGY AS A FUNCTION OF TIME

In the Figure 5.1.6-7, we can see that, at the physical time T = 200 s, the computed free surface shapes for the different simulations are really close to the theoretical height given by the Chabert d'Hières' formula (21).

At the physical time T = 875 s, that is to say 69 wave periods after T = 200 s, Figure 5.1.6-8 shows that the difference between the computed free surface shapes and the theoretical height tends to get smaller when the time step decreases. However, for the four different time steps, this difference increased from T = 200 s. Indeed, as presented in the following table, the wave periods of the simulations are slightly different from the theoretical value of the Airy's formula (22):

Mesh 106*20	Dt=200ms	Dt=100ms	Dt=50ms	Dt=20ms	Theory
Wave period	9.772 s	9.778 s	9.779 s	9.780 s	9.783 s

This explains why the difference between the computed and theoretical free surface shapes tends to increase with elapsed time.

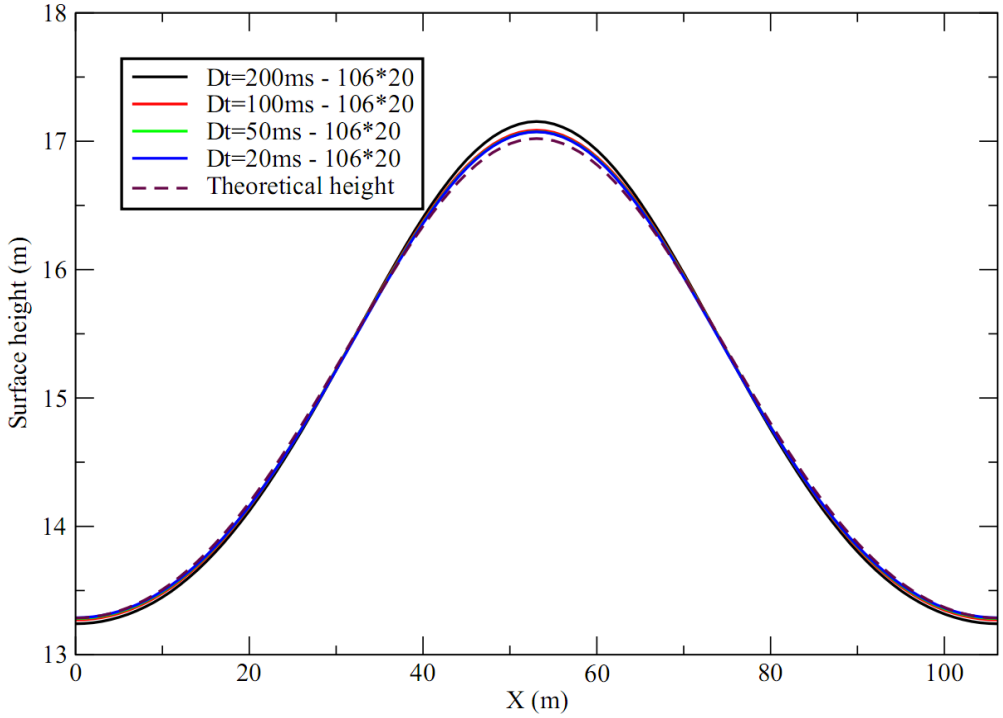

FIGURE 5.1.6-7: FREE SURFACE SHAPE AT THE TIME T = 200 S

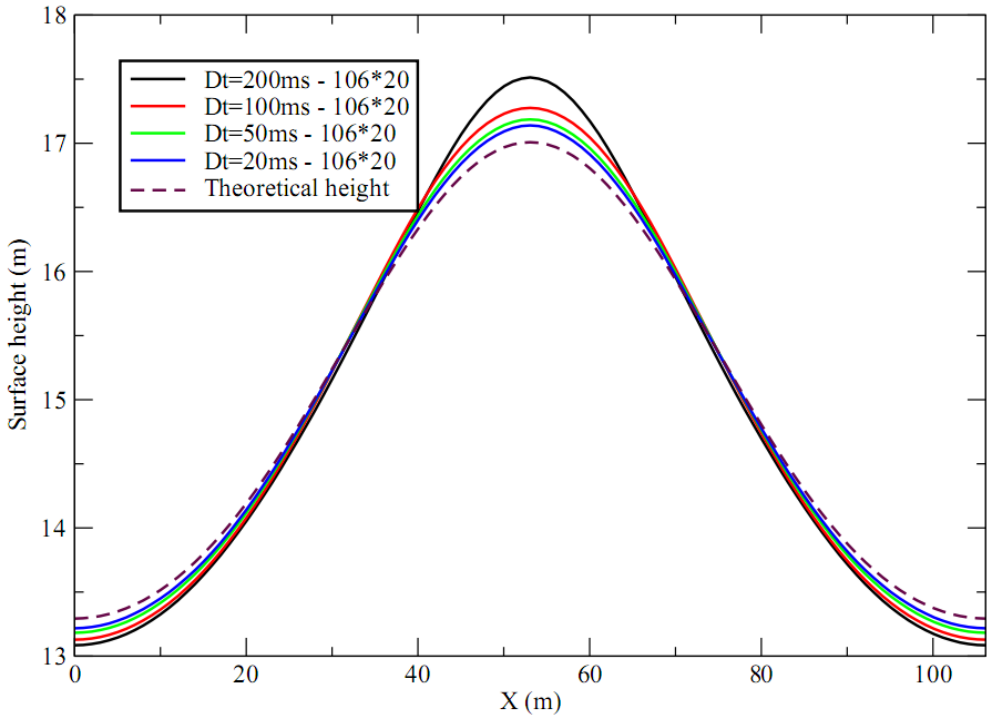

FIGURE 5.1.6-8: FREE SURFACE SHAPE AT THE TIME T = 875 S

The Figure 5.1.6-9 presents, for three different meshes and time step values (chosen in order to have a constant maximum Courant number $Co = 0.2$), the L2 error for the free surface height as a function of time. after 200 s of physical time, the L2 error ranges from 0.02% for the 424*80 mesh to 0.01% for the 106*20 mesh, i.e. the L2 error tends to increase when the mesh gets finer. This conclusion is interesting because, as a general rule, the opposite occurs. However this is true for classical steady-state CFD test case (such as the lid driven cavity), but here we are dealing with an initial value problem where the error will grow with physical time. When we compare different meshes for the same physical time, the smaller meshes and time steps will have involved more iterations and interpolations.

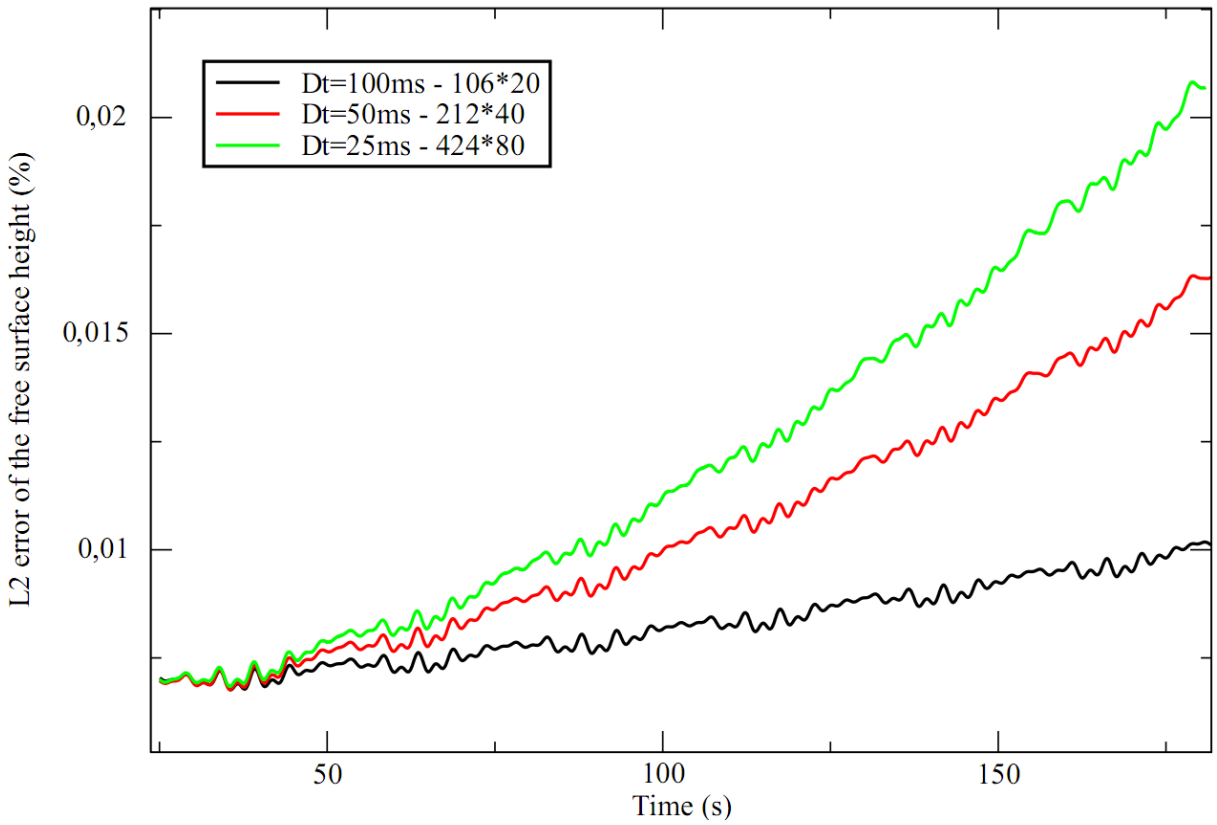

FIGURE 5.1.6-9: L2 ERROR OF THE FREE SURFACE HEIGHT AS A FUNCTION OF TIME

The Figure 5.1.6-10 presents, for the same three different meshes and time step values, the relative error of global volume as a function of time. The relative error of global volume ranges from something really close to zero after 200 s of physical time for the 424*80 mesh to 1e-7 for the 106*20 mesh, that is to say that the finer the mesh, the more the mass conservation is met.

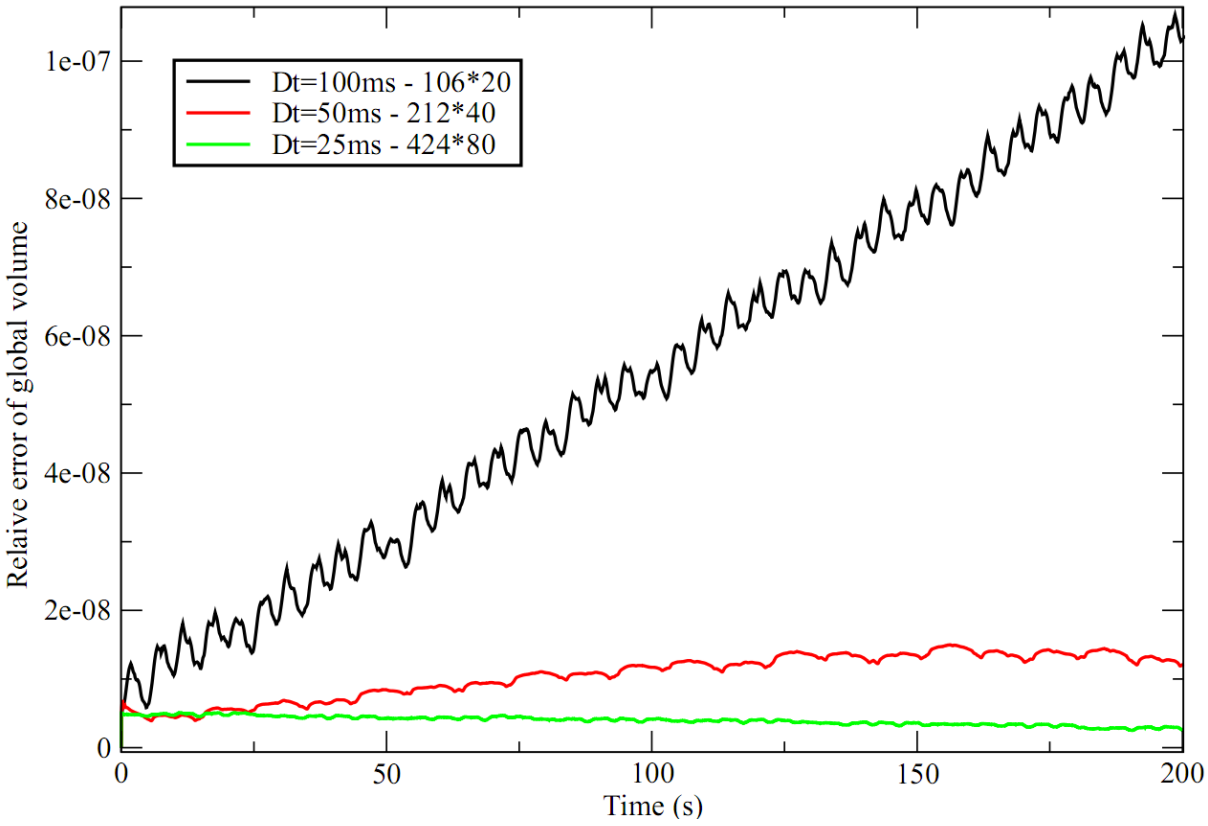

FIGURE 5.1.6-10: RELATIVE ERROR OF GLOBAL VOLUME AS A FUNCTION OF TIME

For the three different meshes and time step values, the Figure 5.1.6-11 shows the relative error of global energy as a function of time. The relative error of global energy ranges from 0.01% after 200 s of physical time for the 424*80 mesh to 0.04% for the 106*20 mesh, that is to say that the finer the mesh, the more the energy conservation is met.

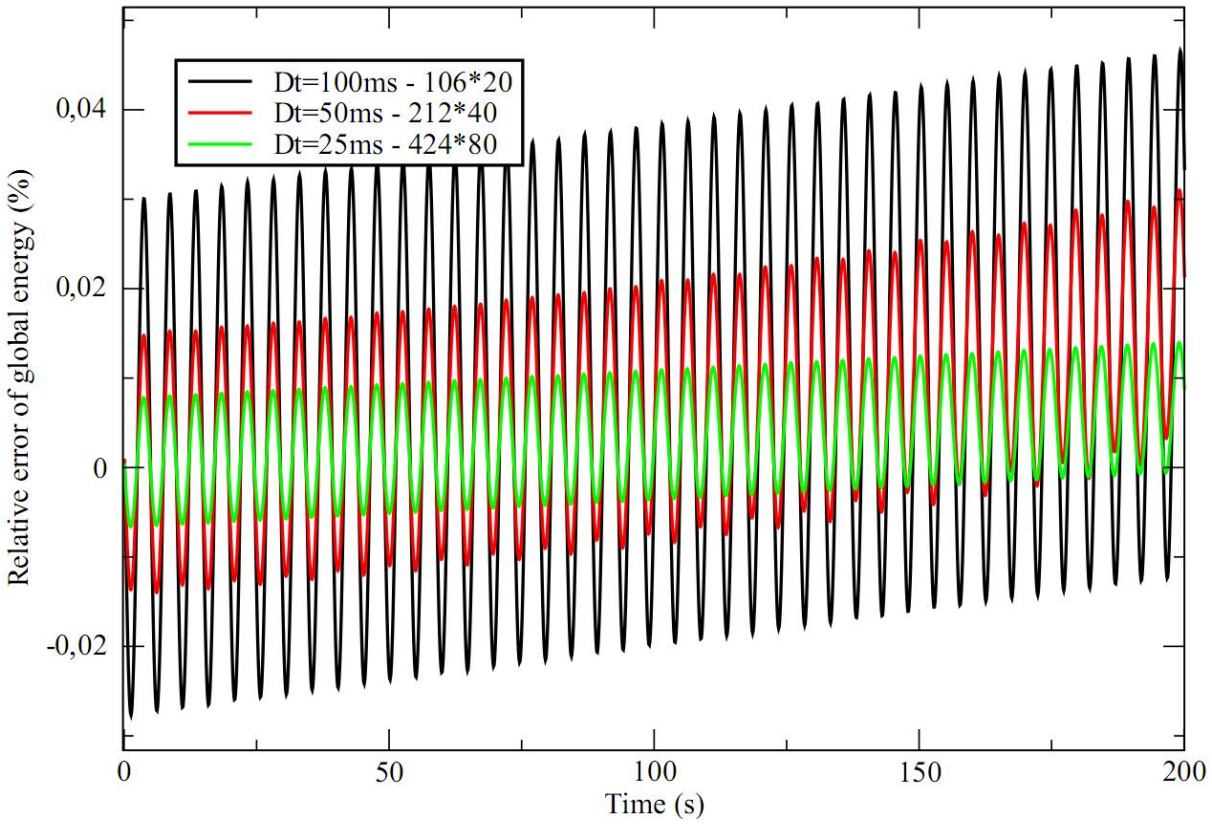

FIGURE 5.1.6-11: RELATIVE ERROR OF GLOBAL ENERGY AS A FUNCTION OF TIME

In the Figure 5.1.6-12, we can see that, at the physical time $T = 25$ s, the computed free surface shapes for the different simulations are quite close to the theoretical height given by the Chabert d'Hières' formula (21).

At the physical time $T = 200$ s, Figure 5.1.6-13 shows that the difference between the computed free surface shapes and the theoretical height tends to get smaller when the mesh is coarser. Indeed, as presented in the following table, the difference between the wave periods of the simulations and the theoretical value of the Airy's formula (22) increases when the mesh gets finer:

	Dt=100ms – 106*20	Dt=50ms – 212*40	Dt=25ms – 424*80	Theory
Wave period	9.776 s	9.769 s	9.766 s	9.783 s

These results show that the convergence in space of the method is not met. A possible explanation is that we are dealing here with an unsteady test case which is an initial value problem. The free surface shape is initially set according to the Chabert d'Hières' theory which is only a second order approximation in amplitude A of the wave shape; the initial error due to the second order approximation will grow with physical time and, for the same physical time, the smaller meshes and time steps need more iterations and interpolations, which can increase the final error.

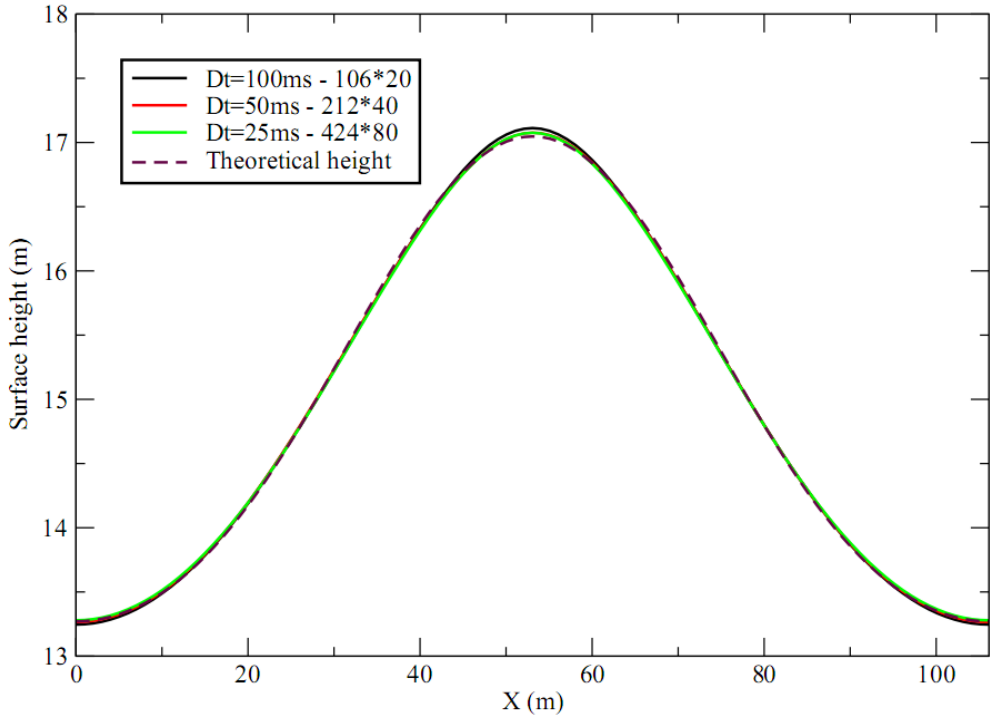

FIGURE 5.1.6-12: FREE SURFACE SHAPE AT THE TIME T = 25 S

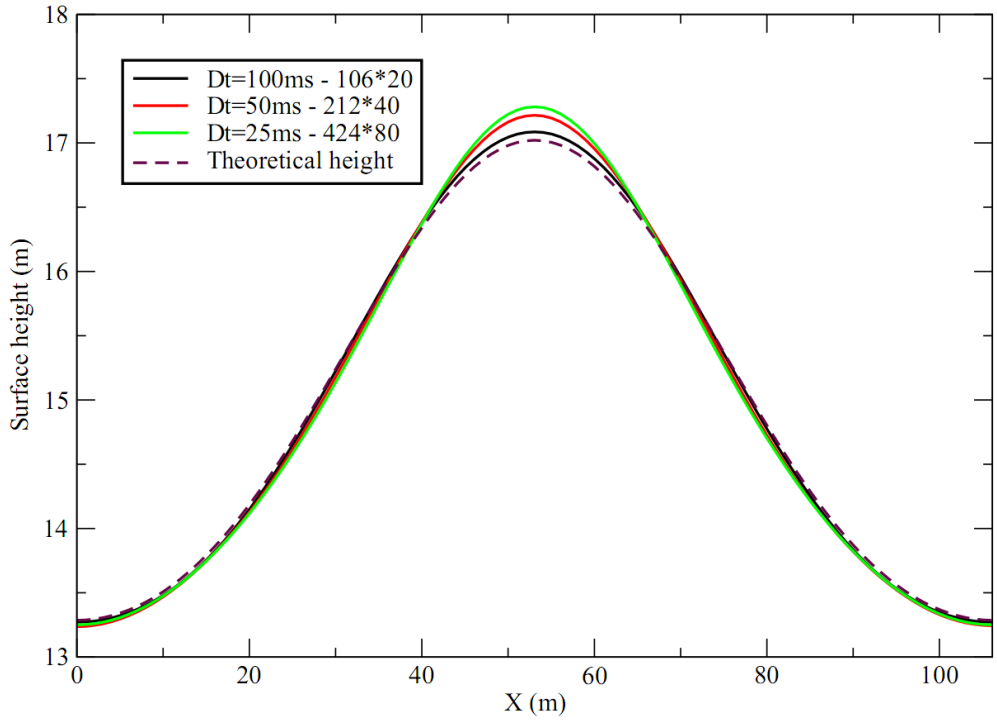

FIGURE 5.1.6-13: FREE SURFACE SHAPE AT THE TIME T = 200 S

To sum up, results are quite encouraging for this test case: the mass conservation is met and the time period of the standing wave oscillations is really close to the theoretical value given by the Airy's formula (22). The free surface shape is also in good agreement with the results given by the Chabert d'Hières' formula (21). Convergence in time shows encouraging results but convergence in space is unexpected and need a further analysis: for this nonlinear case, the simulations should be initially set and compared to a theory more accurate than the second order theory of Chabert d'Hières.

5.1.7. Computing resources used

All the computations presented before were run on a single core of a quad-core processor (Intel Xeon processor at 2.80 GHz with 4 GB of system RAM available) with a constant relative precision for the computation of the free surface mesh velocities (i.e. EPALFS criteria as explained in *Appendix 2: Implementation of the new module*); for each of them, the table hereafter shows the average number of sub-iterations required to converge within the free-surface loop and also the average computation time spent per time step.

Computation name (time step – mesh)	Average sub-iteration number	Average computation time	Maximum Courant number
Dt=200ms - 106*20	6.2	1.05 s	$Co = 0.4$
Dt=100ms - 106*20	5.6	811 ms	$Co = 0.2$
Dt=50ms - 106*20	4.8	616 ms	$Co = 0.1$
Dt=20ms - 106*20	4.0	452 ms	$Co = 0.04$
Dt=50ms - 212*40	6.0	11.4 s	$Co = 0.2$
Dt=25ms - 424*80	4.6	30.0 s	$Co = 0.2$

The mesh 106*20 being fixed, the average sub-iteration number and the average computation time tend to decrease when the time step gets smaller. For a constant maximum Courant number $Co = 0.2$, when the mesh gets finer, the average sub-iteration number tends to decrease whereas the average computation time tends to increase.

5.2. Solitary wave

The second test problem considered in the present study is another common test case for the development of free surface codes: a solitary wave in a water tank; this case is interesting because an analytic solution for the wave shape exists. As the standing wave, this case was already run in 1999 with Solveur Commun (see the EDF report [21]).

5.2.1. Presentation

The test case deals with the solitary wave motion in a tank of length $L = 500$ m and depth $h = 10$ m. The solitary wave amplitude is $A = 2$ m. The mesh used in this test case is composed by hexahedra and is structured; its mesh spacing is constant in both the horizontal direction with 500 cells (1 m), and the vertical direction with 20 cells (0.5 m) and has only one cell in the third direction (0.1 m). The original mesh is presented below in Figure 5.2.1-1:

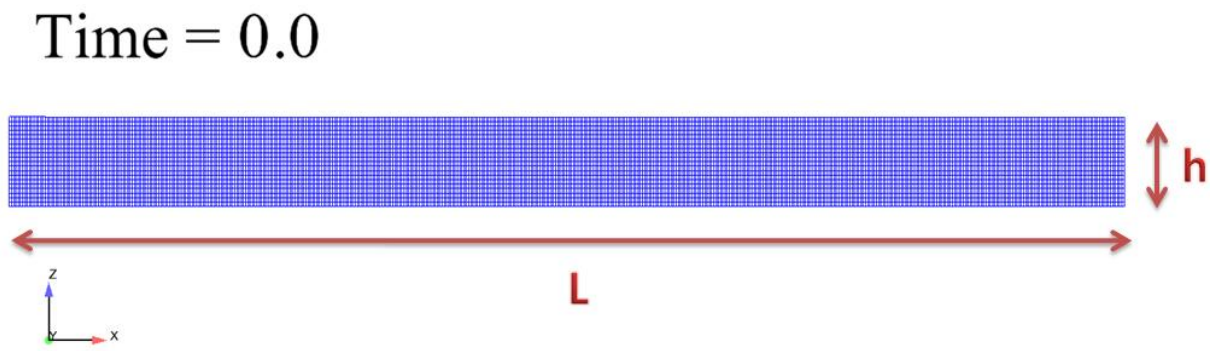

FIGURE 5.2.1-1: ORIGINAL MESH AND INITIAL SHAPE FOR THE SOLITARY WAVE TEST CASE

The shape is two-dimensional and the solitary wave is created by a velocity profile set on the inlet and coded in the routine usclim.F (the velocity profile was coded by Dr Yacine Addad, University of Manchester):

```
C Compute inlet velocity
      D = 10.D0 ! still water depth
      WAVEHT = 2.D0 ! wave height or amplitude
      GRAVITY = 9.81D0
      TWAVE = 10.0D0
C Wave quantities
      AK = SQRT(0.75*WAVEHT/D**3)
      C = SQRT(GRAVITY*(D+WAVEHT))
      DDX = C*TWAVE
C --- We set for the color "INLET" an inlet
      CALL GETFBR('INLET',NLELT,LSTELT)
C     ===========
      DO ILELT = 1, NLELT
```

```
      IFAC = LSTELT(ILELT)
      DO IPHAS = 1, NPHAS
C the solitary wave is created by a velocity profile set on the inlet
        ITYPFB(IFAC,IPHAS) = IENTRE
C the velocities are computed hereafter
C=======================
        ARG = AK *((CDGFBO(1,IFAC)+DDX)-C*TTCABS)
        ARG = MIN (MAX (ARG, -100.0D0),100.0D0)
        ETA = WAVEHT / COSH( ARG )**2
        UIN = C *ETA/ (D+ETA)
        DWDZ = 2.0D0*AK*D*UIN*TANH (ARG) / (D+ETA)
C
        ICODCL(IFAC,IU(IPHAS))   = 1
        RCODCL(IFAC,IU(IPHAS),1) = UIN
        RCODCL(IFAC,IU(IPHAS),2) = RINFIN
        RCODCL(IFAC,IU(IPHAS),3) = 0.D0
C
        ICODCL(IFAC,IV(IPHAS))   = 1
        RCODCL(IFAC,IV(IPHAS),1) = 0.0D0
        RCODCL(IFAC,IV(IPHAS),2) = RINFIN
        RCODCL(IFAC,IV(IPHAS),3) = 0.D0
C
        ICODCL(IFAC,IW(IPHAS))   = 1
        RCODCL(IFAC,IW(IPHAS),1) = DWDZ * CDGFBO(3,IFAC)
        RCODCL(IFAC,IW(IPHAS),2) = RINFIN
        RCODCL(IFAC,IW(IPHAS),3) = 0.D0
C
        ICODCL(IFAC,IPR(IPHAS))   = 3
        RCODCL(IFAC,IPR(IPHAS),1) = 0
        RCODCL(IFAC,IPR(IPHAS),2) = RINFIN
        RCODCL(IFAC,IPR(IPHAS),3) = 0.D0
      ENDDO
    ENDDO
```

Thus a solitary wave of $A = 2$ m amplitude is created. Considering the inviscid theory, the wave should move with a constant crest velocity and amplitude. The analytic solution for the wave shape is given by Archambeau et al. in [21]:

$$z(x,t) = h + 4A\frac{f(x,t)}{\left(1+f(x,t)\right)^2} \tag{23}$$

where the function $f(x,t)$ is defined hereafter:

$$\begin{cases} f(x,t) = exp\big(-2k(x-ct)\big) \\ k = \dfrac{1}{2h}\sqrt{\dfrac{3A}{h+A}} \\ c = \sqrt{g(h+A)} \end{cases} \quad (24)$$

In this case, the numerical application gives a crest velocity $c = 10.85\ \mathrm{m.s^{-1}}$.

As presented in *Appendix 3: Solitary test case – paddle movement*, an attempt to create the solitary wave by a paddle movement set on the inlet was made and coded in the routine usalcl.F (the paddle motion was coded by Rui Xu, University of Manchester, according to the wavemaker theory of Dean and Dalrymple [23]); first results were interesting and needed further development.

5.2.2. Physical characteristics

The test case is run with the following physical characteristics:

- fluid density: $\rho = 10^3\ \mathrm{kg.m^{-3}}$ (water value),
- fluid viscosity: $\mu = 10^{-12}\ \mathrm{kg.m^{-1}.s^{-1}}$ (small enough to consider an inviscid fluid),
- gravity: $g = 9.81\ \mathrm{m.s^{-2}}$,
- Reynolds number: $R_e \to \infty$ for an inviscid fluid.

5.2.3. Boundary conditions

The boundary conditions for the fluid velocity are:

- free surface and outlet: homogeneous Neumann condition,
- inlet: imposed velocity profile,
- tank bottom: homogeneous Dirichlet condition,
- symmetric walls: slip condition.

The boundary conditions for the mesh velocity are:

- free surface: Dirichlet condition according to the formula (9) page 18,
- tank bottom: homogeneous Dirichlet condition,
- symmetric walls, inlet and outlet: slip condition.

The boundary conditions for the pressure are:

- free surface: homogeneous Dirichlet condition according to the formula (8) page 18,

- tank bottom: Neumann condition,
- symmetric walls, inlet and outlet: Neumann condition.

5.2.4. Main computation

All the computations were run on two cores of an Intel Xeon quad-core processor (clock speed: 2.80 GHz, 4 GB of system RAM available) with *Code_Saturne* version 1.3.3 (the geometry allowed this given that all the internal nodes under a free surface node stayed in the same parallelised domain); the laminar turbulence model and a second order time scheme were adopted. The most significant computation was a 50s simulation run with a constant time step $\Delta t = 25$ ms; its results are presented hereafter.

5.2.5. Results

The Figure 5.2.5-1, Figure 5.2.5-2 and Figure 5.2.5-3 show the free surface shape, pressure and velocity fields at the physical times $t_1 = 8.75$ s, $t_2 = 25$ s and $t_3 = 50$ s respectively. In these figures, the pressure and velocity fields seem to be physically right while the free surface shapes are in good agreement with the inviscid theory: the wave crest appears to have a constant amplitude and velocity.

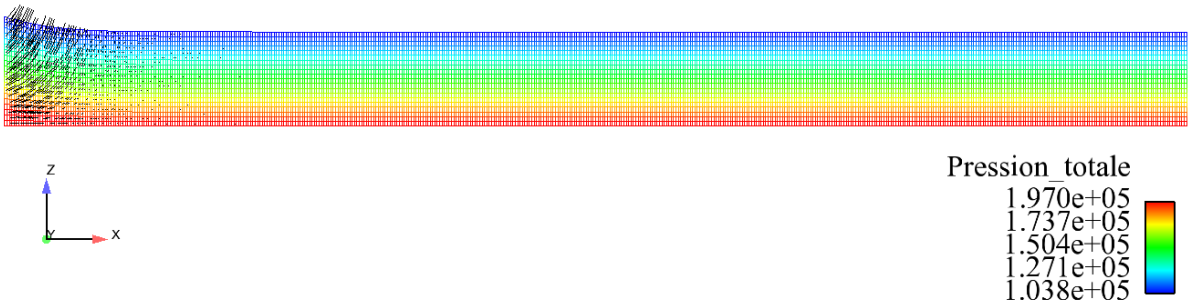

FIGURE 5.2.5-1: FREE SURFACE SHAPE, PRESSURE AND VELOCITY FIELDS AT $T_1 = 8.75$ S

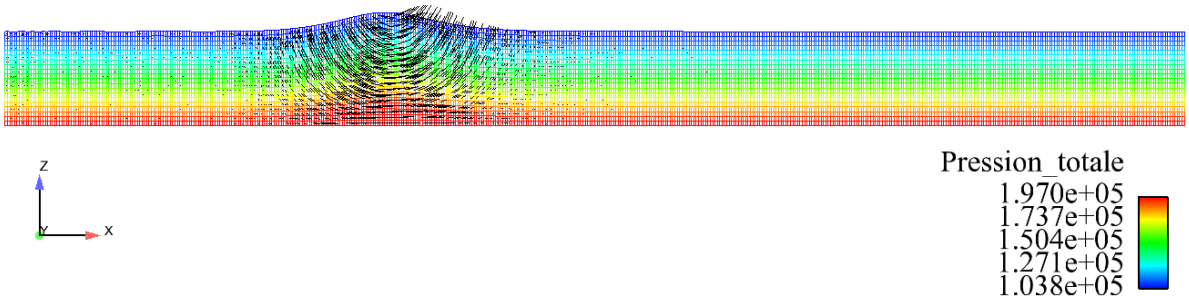

FIGURE 5.2.5-2: FREE SURFACE SHAPE, PRESSURE AND VELOCITY FIELDS AT $T_2 = 25$ S

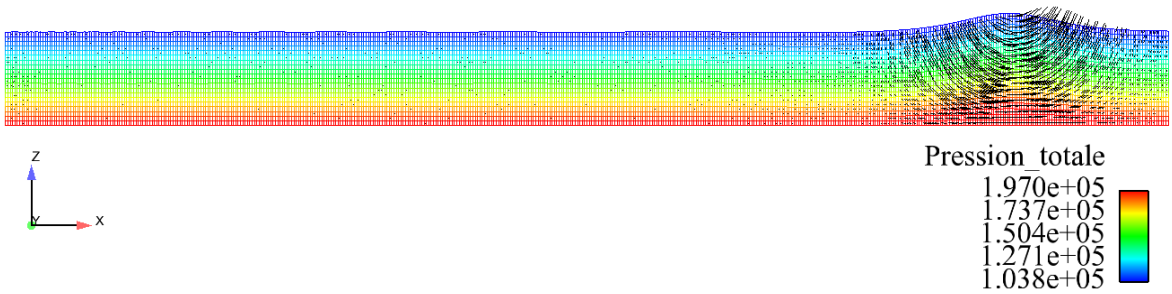

FIGURE 5.2.5-3: FREE SURFACE SHAPE, PRESSURE AND VELOCITY FIELDS AT T_3 = 50 S

In the following two figures (Figure 5.2.5-4 and Figure 5.2.5-5), the computed free surface shape and its analytical solution are plotted at twelve different physical times (from $t = 13.7$ s to $t = 50.0$ s). For all of them, the free surface shape is in good agreement with the analytic solution (23): the computed solitary wave and the analytical one are very similar, they both have the same crest velocity.

Nevertheless the computed amplitude decreases; this is maybe caused by the fact that perturbations are created when the solitary wave enters the computational domain: parasitic oscillations appear and cannot be damped (the water is considered here as an inviscid fluid). It seems that these parasitic oscillations do not occur when both the free surface elevation and the inlet velocity profile are specified, thus they appear to be due to the calculation of the free surface elevation as a function of the mass flow entering the domain.

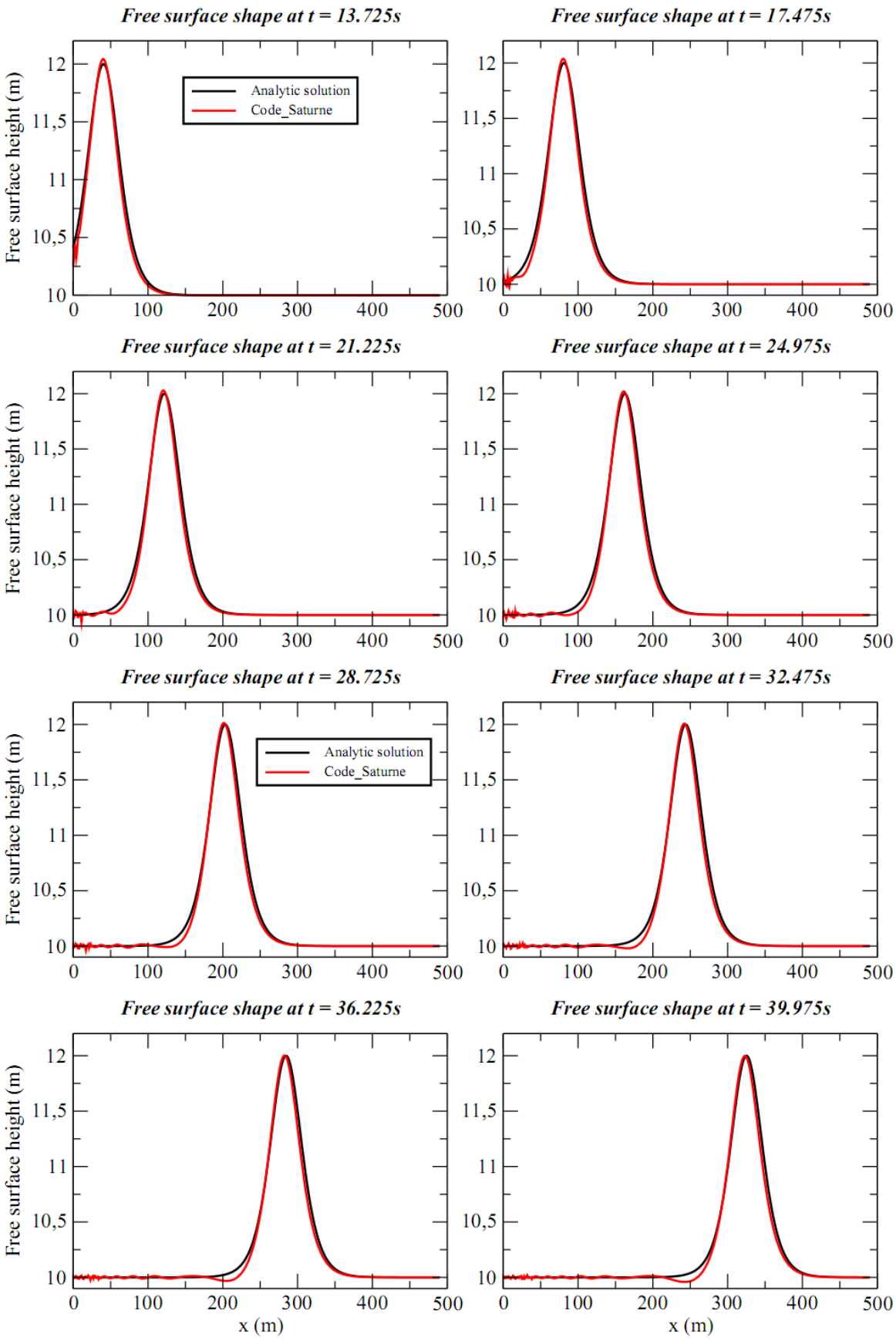

FIGURE 5.2.5-4: FREE SURFACE SHAPES AT 8 DIFFERENT PHYSICAL TIMES

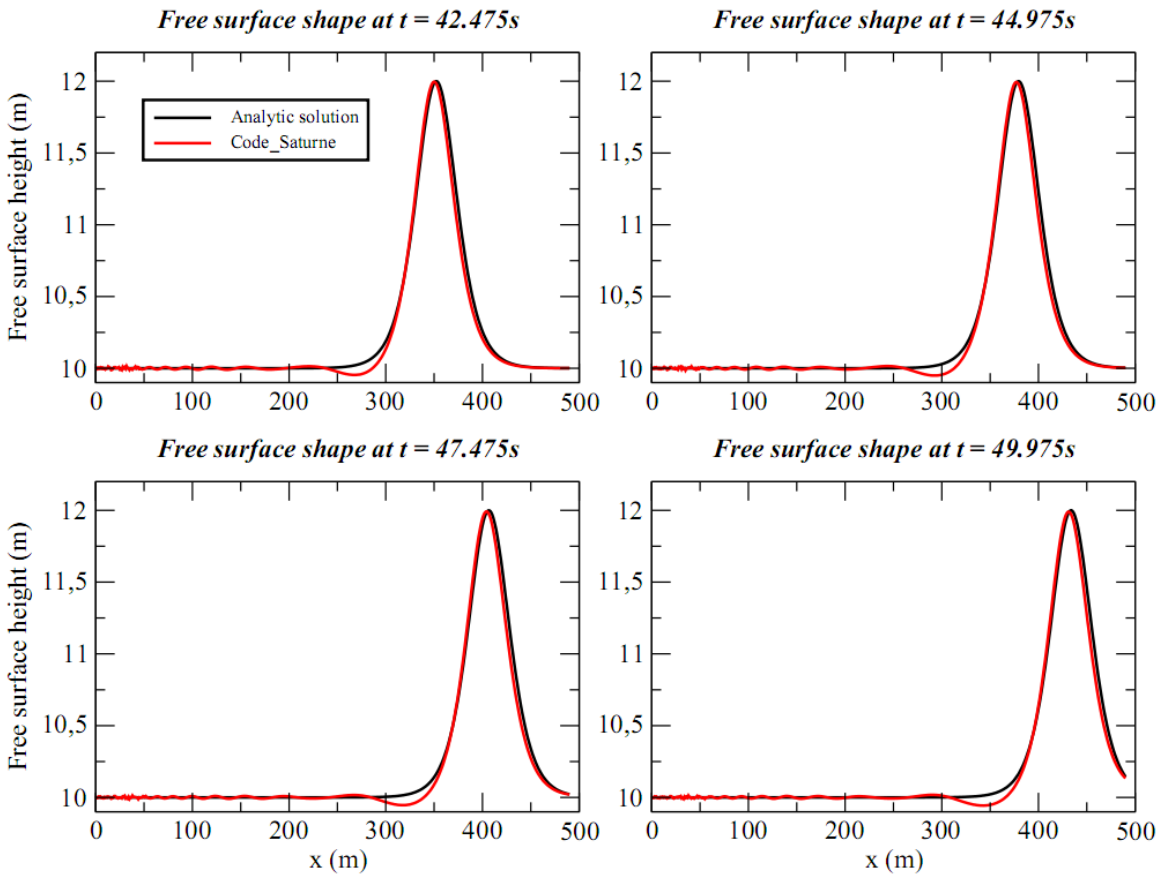

FIGURE 5.2.5-5: FREE SURFACE SHAPES AT 4 DIFFERENT PHYSICAL TIMES

Figure 5.2.5-6 and Figure 5.2.5-7 illustrate the solitary wave shape at two consecutive physical times, $t = 20$ s and $t = 40$ s. For the *Code_Saturne* predictions, two methods are considered:

- Saturne (method-1) in which the mesh velocity is imposed at the free-surface cell-face centres,
- Saturne (method-2) in which the displacement of the free-surface cell-vertices is interpolated as presented in 4.1.2.

The *Code_Saturne* results are compared with the analytical solution (23) and the numerical predictions of the in-house code *STREAM* ([*]).

The results confirm what we saw in the section 3.2.4.B: it is better to impose the displacement of the free-surface cell-vertices. Indeed, using method-1 in *Code_Saturne* is causing a small

[*] As presented by Apsley in [20], STREAM is a finite-volume solver which uses the SIMPLE pressure-correction algorithm to solve the Reynolds-Averaged Navier–Stokes (RANS) equations. For free surface predictions, the ALE method is used and, within each time step, several free-surface updates are realised; each free surface update leads to a mesh adjustment for which several cycles of the SIMPLE algorithm are needed to update the pressure and velocity fields. The solution proceed to the next time step when mass, momentum and free-surface kinematic equations are simultaneously satisfied.

phase lag of the wave displacement, whereas method-2 and STREAM are in good agreement with the analytical solution. From the figures, it is also observed that all three numerical methods appear to underpredict the free surface elevation after the crest.

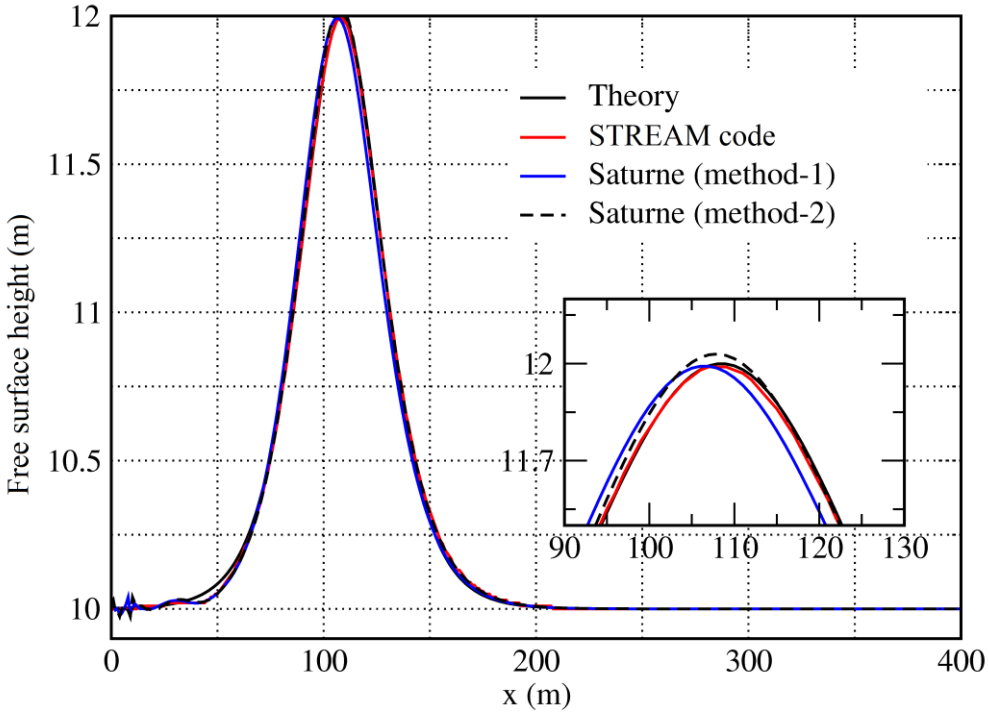

FIGURE 5.2.5-6: WAVE PROFILE AT PHYSICAL TIME T=20S

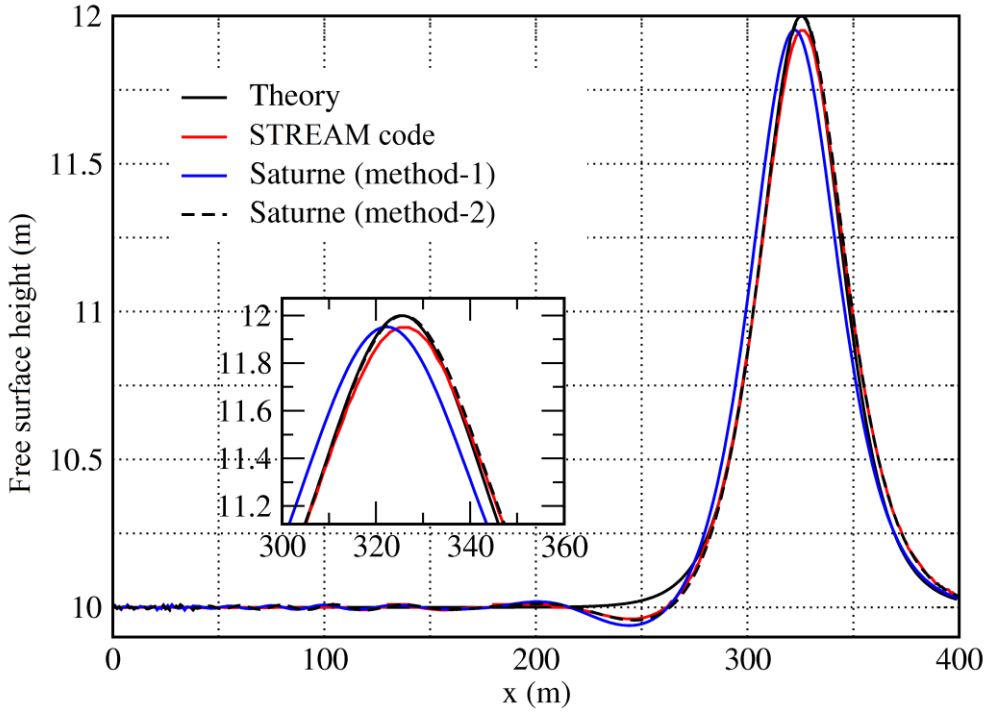

FIGURE 5.2.5-7: WAVE PROFILE AT PHYSICAL TIME T=40S

To sum up, the results of this test case confirmed the validation base on the standing wave in section 5.1: the motion of the computed solitary wave (created here by an imposed velocity profile on the inlet) within the tank is in good agreement with the analytic solution and *STREAM* predictions.

It could be interesting to study how the parasitic oscillations are created by the inlet and why the free surface elevation after the crest is underpredicted.

5.2.6. Effectiveness of parallel computing

In order to evaluate the effectiveness of the free surface module for parallel computing, the main computation presented in 5.2.4 was also run on a single processor core. For each computation, the table hereafter shows the average computation time spent per time step and the average number of sub-iterations required to converge within the free-surface loop.

Number of cores	Average sub-iteration number	Average computation time
1	4.8	4.20 s
2	4.8	2.32 s

The average sub-iteration number is exactly the same for the two computations: that is the proof that, in this case, the parallelisation of the code works properly given that the results of the two computations are not influenced by the parallelisation. In addition, the parallelisation seems to be quite effective: from 4.20 s required for the computation on a single core, the average computation time decreases to 2.32 s for the computation on two cores, it is almost two times faster.

5.3. Duncan's hydrofoil

This test case considers the flow over a hydrofoil under a free surface. This flow was studied experimentally by J.H. Duncan [24] and numerically by S. Muzaferija et al. [25]. It is an interesting case because it eventually converges to a steady state solution. Two codes are considered: *Code_Saturne* with its ALE technique and the commercial code *STAR-CD* with its VOF approach.

5.3.1. Presentation

The hydrofoil of length $L_{NACA} = 20.3$ cm has a NACA 0012 profile, a $80\ \mathrm{cm.s^{-1}}$ fluid velocity, and 5° angle of attack; the undisturbed water above the profile is 21.0 cm, and the Froude Number is 0.567.

The numerical domain is shown in Figure 5.3.1-1 (the dimensions are normalized according to the hydrofoil length L_{NACA}):

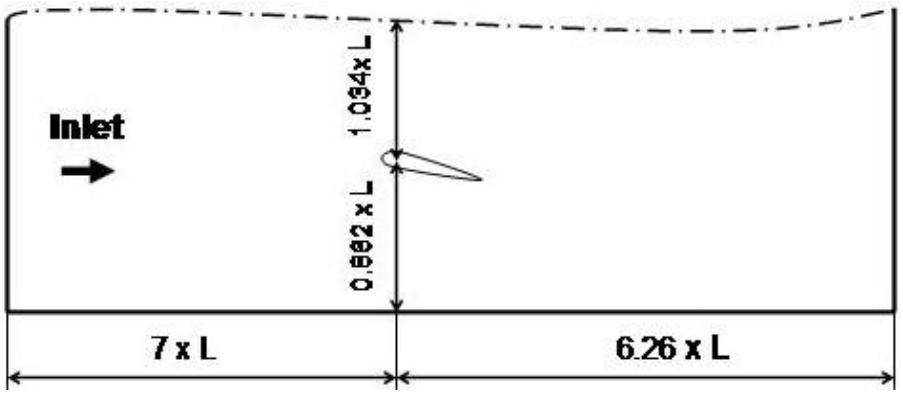

FIGURE 5.3.1-1: SCHEMATIC OF NACA FOIL WITH NORMALIZED DIMENSIONS

The free surface is initially undisturbed and the resistance of the hydrofoil creates progressively a wave downstream; this wave should converge towards a steady solution, according to Duncan's experimental results [24].

5.3.2. Mesh characteristics

Figure 5.3.2-1 is the original mesh used in this test case for the computation with *Code_Saturne*:

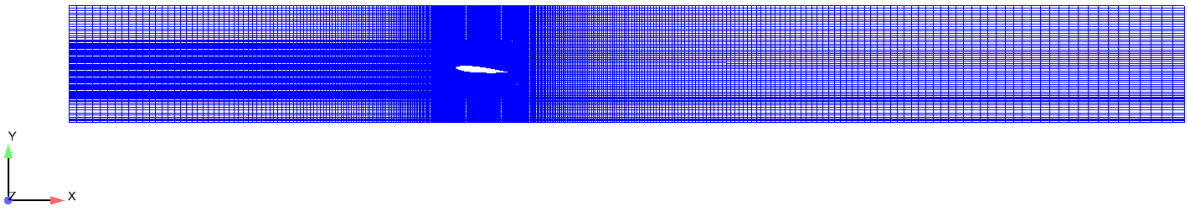

FIGURE 5.3.2-1: ORIGINAL MESH (CODE_SATURNE VERSION – LONG DOMAIN)

This mesh is particularly refined near the hydrofoil, as the Figure 5.3.2-2 shows:

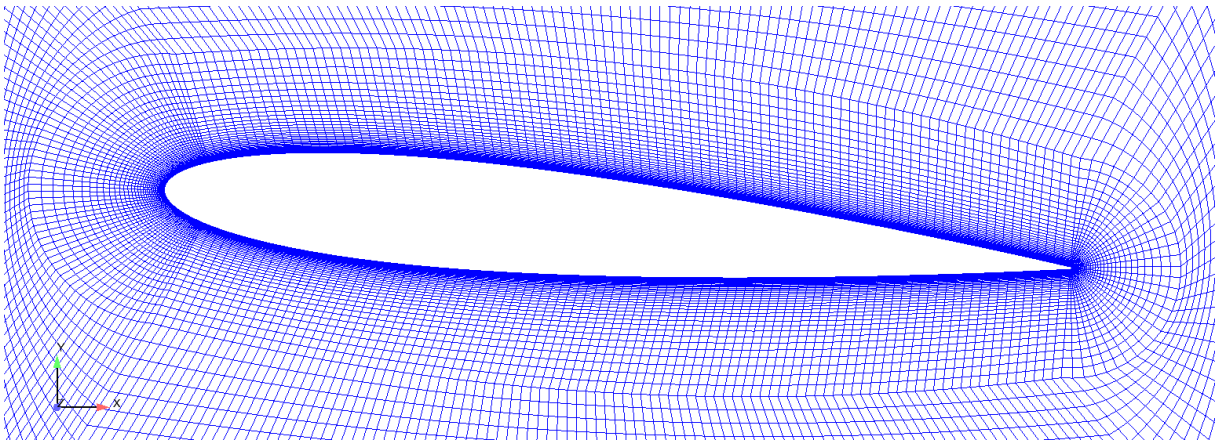

FIGURE 5.3.2-2: MESH NEAR THE NACA 0012 HYDROFOIL (CODE_SATURNE VERSION)

5.3.3. Physical characteristics

The test case is run with the following physical parameters:

- fluid density: $\rho = 10^3$ kg. m^{-3} (water value),
- fluid viscosity: $\mu = 10^{-3}$ kg. m^{-1}. s^{-1} (water value),
- gravity: $g = 9.81$ m. s^{-2},
- NACA length: $L_{NACA} = 20.3$ cm,
- inlet fluid velocity: $V = 80$ cm. s^{-1},
- Froude number: $F_r = \frac{V}{\sqrt{g.L_{NACA}}} = 0.567$,
- Reynolds number: $R_e = \frac{\rho.V.L_{NACA}}{\mu} = 162\,400$.

5.3.4. Boundary conditions

The boundary conditions for the fluid velocity are:

- free surface: homogeneous Neumann condition,
- inlet: constant inlet velocity,

- outlet: convective boundary condition for *Code_Saturne* (*) and standard outlet for the commercial code *STAR-CD*,
- tank bottom and hydrofoil: homogeneous Dirichlet condition,
- symmetric walls: slip condition.

The boundary conditions for the mesh velocity are:

- free surface: Dirichlet condition according to the formula (9) page 18,
- tank bottom and hydrofoil: homogeneous Dirichlet condition,
- symmetric walls, inlet and outlet: slip condition.

The boundary conditions for the pressure are:

- free surface: homogeneous Dirichlet condition according to the formula (8) page 18,
- outlet: convective boundary condition for *Code_Saturne* (*) and standard outlet for the commercial code *STAR-CD*,
- tank bottom, hydrofoil, symmetric walls and inlet: Neumann condition.

5.3.5. Main computations

To predict the free-surface shape, two codes are considered: *Code_Saturne* (version 1.3.3) with the ALE technique and the commercial code *STAR-CD* (version 4.02) with the VOF approach.

The simulations reported in section 5.3.6 were completed in collaboration with Dr Yacine Addad and a report presenting the wave profile and pressure coefficients around the NACA hydrofoil was co-authored [26]:

- First, numerical tests were conducted with the code *STAR-CD* using the VOF technique in order to examine the effects of a turbulence model on the flow predictions at such a low Reynolds number. ($R_e = 162\,400$).
- Then, an extended domain ($15\ L_{NACA}$ canal length behind the hydrofoil instead of $6.26\ L_{NACA}$) was used in the comparative runs to avoid boundary effects on the zone of interest.

* The convective boundary condition was implemented in Code_Saturne by Dr Yacine Addad (University of Manchester); it consists in satisfying an equation of the form $\frac{\partial \Phi}{\partial x} + C\ \frac{\partial \Phi}{\partial n} = 0$ for all the variables Φ including the pressure. A detailed description can be found in [26].

5.3.6. Results

Figure 5.3.6-1 shows the free surface shape and the velocity field for the global domain while Figure 5.3.6-2 focuses on the velocity and pressure fields near the hydrofoil, both at the physical time t = 25.0 s for the *Code_Saturne* computation in the extended domain. In both figures, we can see that the fluid flow is disturbed by the hydrofoil in its proximity, which creates a wave downstream.

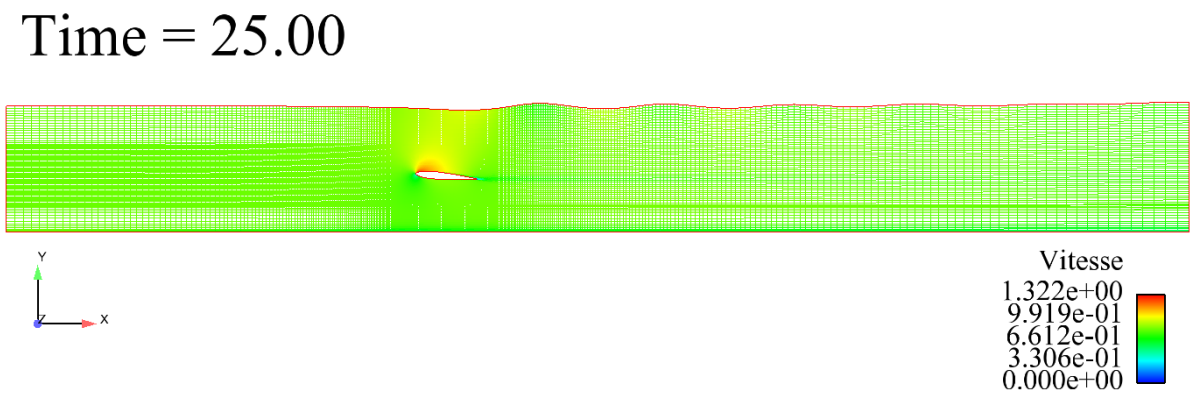

FIGURE 5.3.6-1: FREE SURFACE SHAPE AND VELOCITY FIELD AT T = 25 S

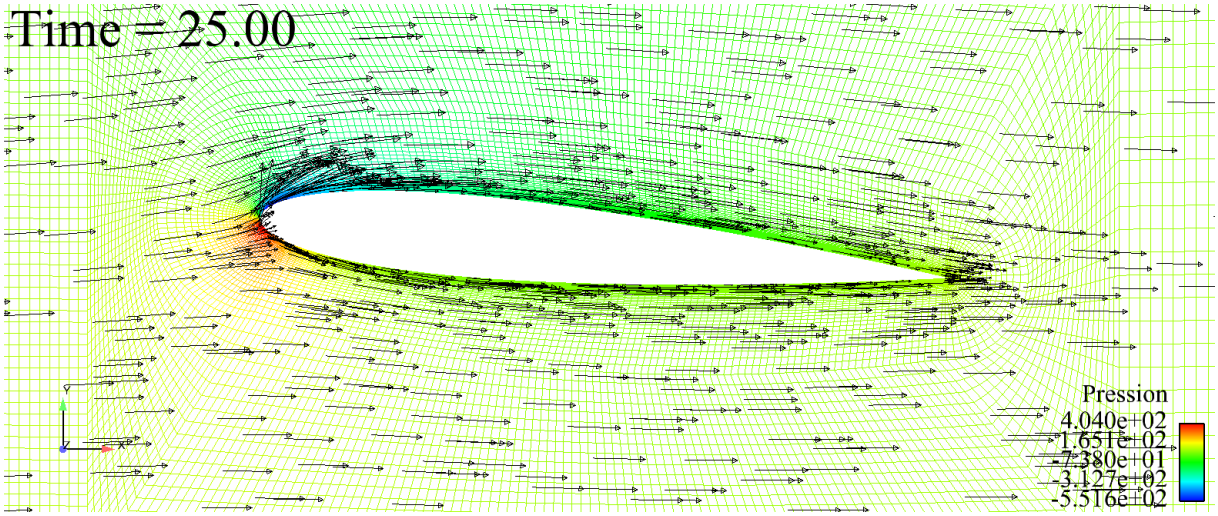

FIGURE 5.3.6-2: PRESSURE AND VELOCITY FIELDS NEAR THE HYDROFOIL AT T = 25 S

The results of wave profile, presented in Figure 5.3.6-3, reveal that the activation of a turbulence model with the VOF method in the short domain ("k-ε" and "no model" computations) has only small effects on the flow predictions, thus justifying the validity of the computations carried out with no turbulence model (i.e. assuming a laminar regime).

As illustrated in Figure 5.3.6-3, both ALE (in *Code_Saturne)* and VOF (in *STAR-CD*) methods under-predict the wave amplitude for the same "long domain" mesh resolution but are well in phase with the experimental measurements.

The first wave is better predicted with the VOF method, while further downstream, the waves predicted with the VOF method dissipate much faster than those obtained with the ALE method which remain at the same amplitude. This is maybe due to the fact that computations run with *Code_Saturne* were performed using the second-order centred difference scheme (CD) for convection while a first order Upwind scheme was selected for the *STAR-CD* code in order to enhance stability (as recommended in the code documentation).

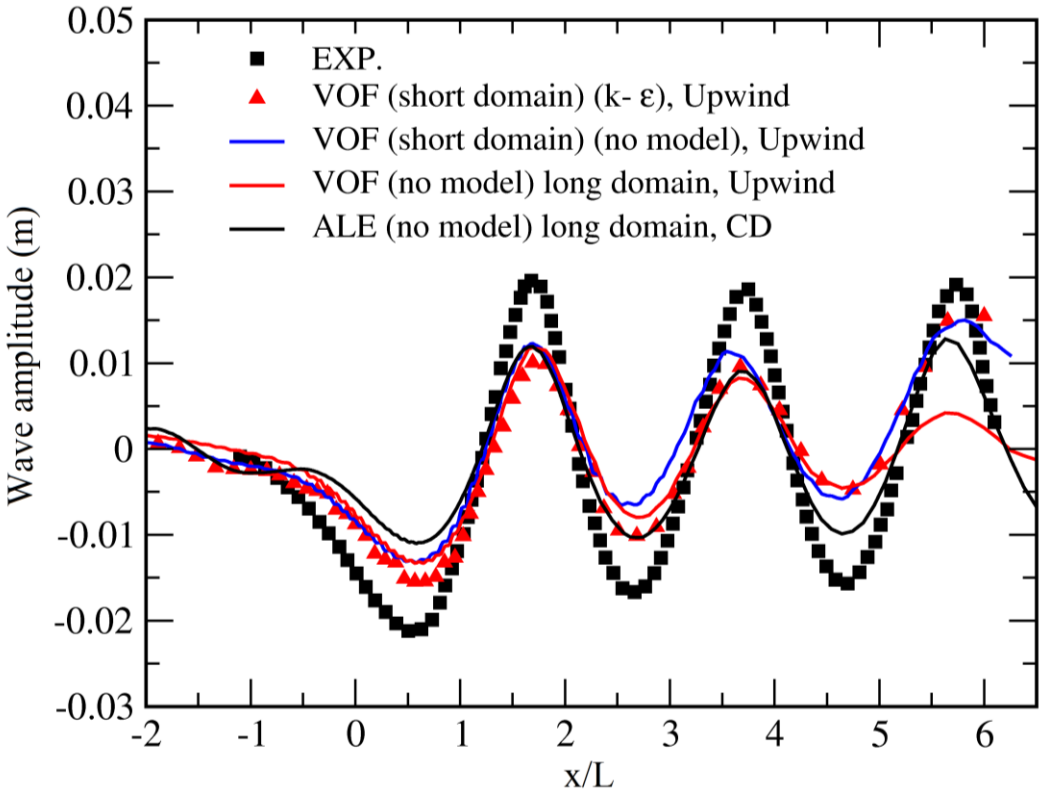

FIGURE 5.3.6-3: WAVE PROFILE FOR A DEPTH OF SUBMERGENCE OF 21.0 CM

To sum up, the *Code_Saturne* ALE results are in a fairly reasonable agreement with the *STAR-CD* VOF approach predictions. Discrepancies between flow-solvers predictions and experimental values, particularly for the wave amplitude, prove that the case is more challenging in the sense that many factors need to be taken into account such as grid resolution, numerical schemes, boundary conditions, turbulence models. Identifying the main reasons of these discrepancies is an interesting topic for further research.

Chapter 6

Limits of existing module and proposal for a new version

6.1. Local volume conservation

We saw that, because of the inaccurate *Code_Saturne* interpolation between the free-surface cell-vertices displacement and the free-surface cell-face centres velocities, we have to compute "by hand" the cell-vertices displacement. The easiest way to do that is to consider an explicit formulation, but it has the disadvantage of not satisfying the local mass conservation.

As seen in section 3.2.4.B, the interpolation inaccuracy occurs as an edge effect and is probably caused by the way the boundary conditions for the mesh velocity are treated within *Code_Saturne*. If this problem is solved, the local volume conservation will then be met and that will improve the code accuracy in predicting free surface flows.

6.2. Parallel computation

The last version of the free surface module does support parallel computation considering some conditions:

- The global domain cannot be divided anyhow: in the global mesh, all the internal cell-vertices under a free-surface cell-vertex (considering the gravity vector) must stay in the same parallelised domain.
- The moving boundary conditions have to match the parallelised domain.
- The post processing outputs need to be implemented considering the parallel constraints.

It will be useful to smooth over these constraints; this way the range of available test-cases will be larger.

6.3. Convergence loop

Sometimes, the convergence loop does not converge towards one unique value, but towards two different values. Even the increase of the maximum iteration number of the convergence loop does not solve the problem; however, by reducing the time step, the convergence loop seems to converge eventually towards one unique value.

Therefore, the implementation of a variable time step method based on both the Courant number and the convergence loop accuracy seems necessary.

6.4. Energy conservation

In the standing wave test-case, the global energy (that is to say the gravitational potential energy plus the kinetic energy) tends to vary slightly sometimes. No energy conservation law is solved, so the global energy conservation is not inherent, but it would be interesting to understand the causes of this phenomenon and its possible solutions.

6.5. Support of irregular mesh with different types of cell

In the free surface module, the actual cell-vertices displacement method consists in an explicit formulation which works only with regular meshes (ordered rectangular grid on the free surface, similar to a chessboard as presented in Figure 4.1.2-1). This limits the range of test cases compatible with *Code_Saturne*. Actually, if the mesh velocity interpolation within *Code_Saturne* is made reliable, unstructured and non-conform meshes will then be supported.

6.6. CFL condition

The Courant number must stay less than one: $\mathrm{Co} = \frac{u.\Delta t}{\Delta x} < 1$. Otherwise the free surface loop has difficulties to converge, and that can lead to a computation crash.

Conclusion

The feasibility of the implementation within *Code_Saturne* of a free surface module based on the ALE method was demonstrated. Indeed, the original ALE module within *Code_Saturne* has been adapted to free-surface flows by adding a convergence loop to perform the free surface movement incrementally within each time step. The geometry was then updated at the end of the time step, thanks to the implemented method, which explicitly computes the displacement of each cell-vertex within the global domain; this allows the module to be globally volume conservative.

Several computations on free surface configurations were run. The first test case is a standing wave in a tank; results are quite good: the time period of the standing wave oscillations and the free surface shape are close to the theoretical values, the mass conservation is met. Convergence in time shows a good behaviour; on the contrary, convergence in space is odd: a lack of convergence is observed with decreasing spatial resolution. This would need a further analysis to be complete: the case should be initially set and compared to a theory more accurate than the second order theory of Chabert d'Hières.

The second configuration deals with a solitary wave in a water tank; the results are in good agreement with the analytic solution – in terms of crest amplitude, crest velocity and solitary wave shape. Parasitic oscillations are created when the solitary wave enters the computational domain and, to get rid of them, it would be interesting to understand how they are created. Further research is needed to understand why the free surface elevation after the crest is underpredicted for both *Code_Saturne* and *STREAM* code.

For the third test case, the flow over a hydrofoil under a free surface, *Code_Saturne* is in fairly good agreement with the *STAR-CD* VOF approach predictions and the experimental measurements of J.H. Duncan, but this case proves to be more challenging: discrepancies between flow-solvers predictions and experimental values, particularly for the wave amplitude, show that many factors would need to be further analyzed – such as grid resolution, numerical schemes, turbulence models and boundary conditions – in order to get better predictions.

To sum up, these first results are encouraging even if, for now, the method shows some important limitations: the local volume conservation law is not met, parallel computations are only available for few configurations and irregular meshes are not supported anymore. Most of these limitations could already be overcome if the *Code_Saturne* interpolation between the free-surface cell-vertices displacement and the free-surface cell-face centres velocities is made reliable; this is definitely the first improvement to do in the next stages of the project. Indeed, Code_Saturne will then be able to run complex 3D applications, such as the flow around a marine turbine with the presence of free-surface effects, a simulation required for the ReDAPT project.

Appendices

Appendix 1: Successive stages within a time step

Calculation of the physical properties
Boundary Conditions
- condli
  - clptur "turbulent" conditions at the wall
  - clsyvt symmetry conditions for the vectors and the tensors

Navier-Stokes solution
- navsto
  - **Velocity prediction**
    - preduv
      - vissec momentum source terms related to the transposed gradient of the velocity
      - viscfa calculation of the viscosity at the faces
      - codits iterative solution of the system using an incremental method
  - **Pressure correction**
    - resolp
      - viscfa calculation of the time step at the faces...
      - visort ...according to the selected options
      - matrix calculation of the Poisson equation matrix
      - inimas initialisation of the mass flow rate
      - itrmas update of the mass flow rate
  - **Velocity correction**
    - standard method or ...
    - recvmc ... least square method

k − epsilon model
- turbke
  - viscfa preliminary steps before...
  - bilsc2 ...source terms coupling
  - viscfa calculation of the viscosity at the faces
  - codits iterative solution of the systems using an incremental method

Reynolds stress model
- turrij
  - visort calculation of the viscosity at the faces
  - codits iterative solution of the systems using an incremental method

Equations for the scalars
- covofi
  - viscfa calculation of the viscosity at the faces
  - codits iterative solution of the systems using an incremental method

Appendix 2: Implementation of the new module

Structure of the convergence loop (routines independent of the test case)

The following routines manage the convergence loop: they are independent of the test case simulated, and a basic new user would not need to modify them.

cs_ale.h (header file to declare parallelisation functions – modified file):

The new parallelisation function ALEFRS is declared here: this function computes the free surface nodes displacement for the parallelised border nodes.

cs_ale.c (C file to define parallelisation functions – modified file):

The new parallelisation function ALEFRS is defined here: this function computes the free surface nodes displacement for the parallelised border nodes by creating a table of connectivity between the border nodes and the parallelised domains they belong to.

albase.h (include file to declare global variables – modified file):

Three new global variables are created:

- ACTIFS (integer): use of an iterative scheme to compute the free surface mesh velocities (equal to 0 when disabled, equal to 1 when enabled),
- NBITFS (integer): maximum iterations number for the computation of the free surface mesh velocities (i.e. when ACTIFS = 1),
- EPALFS (real): relative precision for the computation of the free surface mesh velocities.

usalin.F (user's routine for ALE's settings – modified routine):

The three new global variables required to settle the convergence loop are initialized here.

```
C     Activation of the iterative scheme for the computation of the free surface mesh velocities
      ACTIFS = 1
C     Maximum iterations number for the computation of the free surface mesh velocities when ACTIFS = 1
      NBITFS = 10
C     Relative precision for the computation of free surface mesh velocities
      EPALFS = 1.D-6
```

ustbus.F (user's routine to define the dimensions of the user's and developer's tables – modified routine):

The dimensions of the user's and developer's tables are required: the developer's table is used to store the boundary conditions and mass flows values, whereas the user's table stores data is needed for post processing.

```
NITUSE = NNOD
NIDEVE = 5*NFABOR
NRTUSE = 4*NFABOR+5+2*NNOD
NRDEVE = NFAC+9*NFABOR+1+NNOD
```

tridim.F (solving of Navier-Stokes and scalar equations for one time step – modified routine):

This routine sets the structure of the free surface convergence loop.

If the iterative scheme to compute the free surface mesh velocities is enabled and the ALE initialisation iteration already occurred, then the convergence loop can start.

```
IF (ACTIFS.EQ.1 .AND. ITRALE.NE.0) THEN
 ITERFS = 1
ELSE
 ITERFS = -1
ENDIF
```

Next, the initial boundary conditions values for the variables are saved at the first iteration.

```
     IF (ACTIFS.EQ.1 .AND. ITERFS.EQ.1) THEN
      CALL FSSAVE
C       ===========
   & ( IFINIA , IFINRA , ITRALE , ITALIM , INEEFL ,
   &   NDIM   , NCELET , NCEL   , NFAC   , NFABOR , NFML   , NPRFML ,
   &   NNOD   , LNDFAC , LNDFBR , NCELBR ,
   &   NIDEVE , NRDEVE , NITUSE , NRTUSE ,
   &   IFACEL , IFABOR , IFMFBR , IFMCEL , IPRFML,
   &   IPNFAC , NODFAC , IPNFBR , NODFBR ,
   &   IA(IIMPAL)     ,
   &   IDEVEL , ITUSER , IA     ,
   &   XYZCEN , SURFAC , SURFBO , CDGFAC , CDGFBO ,
   &   XYZNOD , VOLUME ,
   &   RTP    , RTPA   , PROPCE , PROPFA , PROPFB ,
   &   COEFA  , COEFB  ,
   &   RDEVEL , RTUSER , RA   )
C
     ENDIF
```

And lastly, if the convergence criterion is not met, the initial boundary conditions and variables values are reloaded and the convergence loop starts a new iteration.

```
   IF (ACTIFS.EQ.1) THEN
    CALL FSLOAD
C      ===========
   & ( IFNIA1 , IFINRA ,
```

```
     &   ITRALE , ITERFS ,
     &   NDIM   , NCELET , NCEL   , NFAC   , NFABOR , NFML   , NPRFML ,
     &   NNOD   , LNDFAC , LNDFBR , NCELBR , NVAR   ,
     &   NIDEVE , NRDEVE , NITUSE , NRTUSE ,
     &   IFACEL , IFABOR , IFMFBR , IFMCEL , IPRFML ,
     &   MAXELT , IA(ILS),
     &   IPNFAC , NODFAC , IPNFBR , NODFBR ,
     &   IDEVEL , ITUSER , IA     ,
     &   XYZCEN , SURFAC , SURFBO , CDGFAC , CDGFBO , XYZNOD , VOLUME ,
     &   DT     , RTP    , RTPA   , PROPCE , PROPFA , PROPFB ,
     &   COEFA  , COEFB  ,
     &   RDEVEL , RTUSER ,
     &   RA     )
C
      IF (ITERFS.NE.-1) THEN
        ITERFS = ITERFS + 1
        GOTO 300
      ENDIF
    ENDIF
```

fssave.F (save of the mass flows and boundary conditions values when the free surface module is enabled – new routine):

The values of the mass flows and fluid velocity and pressure boundary conditions are saved in the developer's real table RDEVEL.

navsto.F (solving of the Navier-Stokes equations for one time step – modified routine):

During the solving of the Navier-Stokes equations, in the correction step, the values of the mass flows through the free surface are saved just before the addition of the mesh velocity to the convective flux; these mass flows actually count the fluid supposed to go through the free surface if the free surface is considered as fixed. These values are stored in the user's real table RTUSER.

```
    IF (ACTIFS.EQ.1) THEN
      CALL GETFBR('FREE_SURF',NLELT,LSTELT)
C     ===========
      DO ILELT = 1, NLELT
        IFAC = LSTELT(ILELT)
        RTUSER(4*IFAC) = RTUSER(4*IFAC+1)
        RTUSER(4*IFAC+1) = PROPFB(IFAC,IPPROB(IFLUMA(IU(1))))
      ENDDO
    ENDIF
```

fsload.F (possible back to the saved values of the variables and boundary conditions (fssave.F) when the free surface with iterative prediction of the free surface mesh velocities is enabled – new routine):

The difference between the new mass flows RTUSER(4*IFAC+1) on the free surface and the ones of the previous iteration of the convergence loop RTUSER(4*IFAC) is calculated:

```
      DELTA = 0.D0
      CALL GETFBR('FREE_SURF',NLELT,LSTELT)
C     ===========
      DO ILELT = 1, NLELT
        IFAC = LSTELT(ILELT)
        IF(DELTA.LT.(SQRT((RTUSER(4*IFAC+1)-RTUSER(4*IFAC))**2)/ABS(RTUSER(4*IFAC+1))))
THEN
          DELTA=(SQRT((RTUSER(4*IFAC+1)-RTUSER(4*IFAC))**2)/ABS(RTUSER(4*IFAC+1)))
        ENDIF
      ENDDO
```

That gives us a value DELTA which can be compared with EPALFS:

```
      IF (DELTA.LE.EPALFS) THEN
        ITERFS = -1
        WRITE(NFECRA,*) 'Convergence', DELTA
      ELSEIF (ITERFS.EQ.NBITFS) THEN
        CALL GETFBR('FREE_SURF',NLELT,LSTELT)
C       ===========
        DO ILELT = 1, NLELT
          IFAC = LSTELT(ILELT)
          RTUSER(4*IFAC+1) = (RTUSER(4*IFAC+1)+RTUSER(4*IFAC))/2
        ENDDO
        WRITE(NFECRA,*) 'Non Convergence', DELTA
      ELSEIF (ITERFS.GT.NBITFS) THEN
        ITERFS = -1
      ENDIF
```

If the convergence criterion is met (i.e. DELTA < EPALFS), the convergence loop can be ended directly by putting ITERFS = -1.

Otherwise, as long as the convergence criterion is not met, the convergence loop carries on until the iteration number ITERFS raises the value of NBITFS. At this point, a final iteration is added where the mass flow value is the average of the two previous ones.

If ITERFS is positive, the convergence loop is still active, which means that the code has to come back to the previous values of the variables and boundary conditions saved in fssave.F (except for the mass flows):

```
      IF (ITERFS.NE.-1) THEN
        DO II = 1, NVAR
```

```
      DO IEL = 1, NCELET
        RTP(IEL,II) = RTPA(IEL,II)
      ENDDO
    ENDDO
    DO IFAC = 1, NFABOR
      COEFA(IFAC,ICLP) = RDEVEL(NFAC+NFABOR+IFAC)
      COEFA(IFAC,ICLU) = RDEVEL(NFAC+2*NFABOR+IFAC)
      COEFA(IFAC,ICLV) = RDEVEL(NFAC+3*NFABOR+IFAC)
      COEFA(IFAC,ICLW) = RDEVEL(NFAC+4*NFABOR+IFAC)
      COEFB(IFAC,ICLP) = RDEVEL(NFAC+5*NFABOR+IFAC)
      COEFB(IFAC,ICLU) = RDEVEL(NFAC+6*NFABOR+IFAC)
      COEFB(IFAC,ICLV) = RDEVEL(NFAC+7*NFABOR+IFAC)
      COEFB(IFAC,ICLW) = RDEVEL(NFAC+8*NFABOR+IFAC)
    ENDDO
  ENDIF
```

Boundary conditions (routines dependent of the test case)

The following routines depend on the simulated test case; the code lines presented here focuses on the implementation of the free surface boundary conditions – for the standing wave test case.

usalcl.F (user's routine for the loading of the boundary conditions for the mesh velocity):

The displacement of the free surface nodes is imposed by using the DEPALE array. This way, the problems caused by the use of the RCODCL are avoided (see the section 3.2.4.B).

First, for the colour “FREE_SURF”, that is to say the free surface, an imposed mesh velocity is set.

```
  CALL GETFBR('FREE_SURF',NLELT,LSTELT)
C   ===========
  DO ILELT = 1, NLELT
    IFAC = LSTELT(ILELT)
    IEL = IFABOR(IFAC)
    IPHAS = 1
    IALTYB(IFAC) = IVIMPO
    RCODCL(IFAC,IUMA,1) = 0.D0
    RCODCL(IFAC,IVMA,1) = 0.D0
```

The vertical free surface velocity is directly computed as the ratio between the mass flow through the free surface face IFAC (value saved in the RTUSER user’s real table), and the density and the vertical component of the face surface vector.

```
    RCODCL(IFAC,IWMA,1) =

        (RTUSER(4*IFAC+1)/(PROPFB(IFAC,IPPROB(IROM(IPHAS)))*SURFBO(3,IFAC)))
```

At this point, the code has to compute the free surface nodes displacements; there are two ways of doing that:

- the first scheme is simple and works for all the test cases (2D and 3D): each free surface nodes displacement is computed explicitly and is the average of the mesh velocities at the closest free surface face centres, except for the side nodes. This scheme does meet the global conservation of the mass, but not the local one.

```
 IF(NTCABS.GT.0) THEN
  DO II = IPNFBR(IFAC), IPNFBR(IFAC+1)-1
   INOD = NODFBR(II)
   IMPALE(INOD) = 1
   DEPALE(INOD,1) = 0.D0
   DEPALE(INOD,2) = 0.D0
   RTUSER(4*NFABOR+5+INOD) = RTUSER(4*NFABOR+5+INOD) +
* RCODCL(IFAC,IWMA,1)*DTREF
   RTUSER(4*NFABOR+5+NNOD+INOD)=RTUSER(4*NFABOR+5+NNOD+INOD)+1
   DEPALE(INOD,3) = XYZNOD(3,INOD)-XYZNO0(3,INOD) +
*RTUSER(4*NFABOR+5+INOD)/RTUSER(4*NFABOR+5+NNOD+INOD)
  ENDDO
 ENDIF
ENDDO
```

- the second way consists in computing an implicit free surface node displacement by solving a system of equations between nodes displacement and face centre velocities. This scheme does satisfy its local mass conservation; it is only available for 2D test cases with very simple meshes and is not parallelised (at least for the moment).

This scheme is activated when IUTILE = 1, then the creation of a free surface vertices/faces linking table is necessary and created in the developer's integer table IDEVEL during the initialisation step.

```
IF(NTCABS.GT.0 .AND. IUTILE.EQ.1) THEN
 DO IND = 0, NLELT
  IFAC = IDEVEL(3*IND)
  DO II = 1, 2
   INOD = IDEVEL(3*IND+II)
   IMPALE(INOD) = 1
   DEPALE(INOD,1) = 0.D0
   DEPALE(INOD,2) = 0.D0
   IF(IND.EQ.0) THEN
```

Actually there is one more unknown node displacement than the number of equations, so one more arbitrary equation is necessary to be able to compute the free surface nodes displacement (e.g. one can add an additional equation on a side node).

```
          IFAC = IDEVEL(3)
          DEPALE(INOD,3) = XYZNOD(3,INOD)-XYZNO0(3,INOD)+RCODCL(IFAC,IWMA,1)*DTREF
          RDEVEL(NFAC+9*NFABOR+1+INOD) = RCODCL(IFAC,IWMA,1)
         ELSE
          INOD2 = IDEVEL(3*IND+II-3)
          DEPALE(INOD,3) = XYZNOD(3,INOD)-XYZNO0(3,INOD)
    *+ (2*RCODCL(IFAC,IWMA,1)-RDEVEL(NFAC+9*NFABOR+1+INOD2))*DTREF
          RDEVEL(NFAC+9*NFABOR+1+INOD) = (2*RCODCL(IFAC,IWMA,1)
    *- RDEVEL(NFAC+9*NFABOR+1+INOD2))
         ENDIF
        ENDDO
       ENDDO
      ENDIF
```

Then the parallelisation of the code has to be ensured: the free surface border nodes displacement must have exactly the same value on every parallelised domain.

```
      IF(IRANGP.GE.0) THEN
       CALL ALEFRS
C      ===========
     & ( IFACEL , IFABOR , IPNFAC , NODFAC , IPNFBR , NODFBR ,
     &   RTP(1,IUMA), RTP(1,IVMA), RTP(1,IWMA),
     &   COEFA(1,ICLUMA), COEFA(1,ICLVMA), COEFA(1,ICLWMA),
     &   COEFB(1,ICLUMA), COEFB(1,ICLVMA), COEFB(1,ICLWMA),
     &   DT, RTUSER(4*NFABOR+6+NNOD), RTUSER(4*NFABOR+6) )
C
       DO INOD=1, NNOD
        DEPALE(INOD,3) = XYZNOD(3,INOD)-XYZNO0(3,INOD) +
     *RTUSER(4*NFABOR+5+INOD)/RTUSER(4*NFABOR+5+NNOD+INOD)
       ENDDO
      ENDIF
```

Given that the nodes just move in one direction and the mesh is regular, for all the internal nodes, their displacement is directly linked to the free surface nodes displacement, according to the ratio between heights.

```
      IF(NTCABS.GT.0) THEN
       DO ILELT = 1, NLELT
       IFAC = LSTELT(ILELT)
        DO II = IPNFBR(IFAC), IPNFBR(IFAC+1)-1
         INOD = NODFBR(II)
         DO INOD2 = 1, NNOD
          IF(ABS(XYZNO0(1,INOD2)-XYZNO0(1,INOD)).LT.PRECIS
     *.AND.ABS(XYZNO0(2,INOD2)-XYZNO0(2,INOD)).LT.PRECIS
     *.AND.ABS(XYZNO0(3,INOD2)-XYZNO0(3,INOD)).GT.PRECIS) THEN
            IMPALE(INOD2) = 1
            DEPALE(INOD2,1) = DEPALE(INOD,1)
            DEPALE(INOD2,2) = DEPALE(INOD,2)
```

```
            DEPALE(INOD2,3) = DEPALE(INOD,3)*XYZNO0(3,INOD2)
     */XYZNO0(3,INOD)
           ENDIF
         ENDDO
        ENDDO
       ENDDO
      ENDIF
```

usclim.F (user's routine for the loading of the boundary conditions for the unknown variables):

For the colour "FREE_SURF" free surface, a free outlet is set, with a Neumann condition for the velocities and a Dirichlet condition for the pressure:

```
      CALL GETFBR('FREE_SURF',NLELT,LSTELT)
C     ===========
      DO ILELT = 1, NLELT
        IFAC = LSTELT(ILELT)
        DO IPHAS = 1, NPHAS
          ITYPFB(IFAC,IPHAS) = ISOLIB
          ICODCL(IFAC,IU(IPHAS)) = 3
          RCODCL(IFAC,IU(IPHAS),3) = 0.D0
          ICODCL(IFAC,IV(IPHAS)) = 3
          RCODCL(IFAC,IV(IPHAS),3) = 0.D0
          ICODCL(IFAC,IW(IPHAS)) = 3
          RCODCL(IFAC,IW(IPHAS),3) = 0.D0
          ICODCL(IFAC,IPR(IPHAS)) = 1
          RCODCL(IFAC,IPR(IPHAS),1) = P0(IPHAS)
        ENDDO
      ENDDO
```

usini1.F (user's routine to set computational parameters):

If we want to adopt a second order time scheme (*Code_Saturne* is first order accurate in time by default, some turbulence models cannot work with the second order accuracy in time), we have to add the following line:

```
      ISCHTP(1) = 2
```

In addition, the gravity field and the fluid's properties are set here.

Appendix 3: Solitary test case – paddle movement

The solitary wave can be created by a paddle movement coded in the routine usalcl.F:

```
g=sqrt(GX**2+GY**2+GZ**2)
H_over_h0 = 0.2d0
depth = 10.d0
waveHeight = H_over_h0*depth
kappa = sqrt(3.0d0*waveHeight/(4.0d0*(depth**3)))
celerity = sqrt(g*(depth + waveHeight))
x_paddle_initial = 0.0d0
stroke = 2.0d0*H_over_h0/kappa
t_0 = 3.8d0/(kappa*celerity)   !3.8 = atanh(1.0)
tau = 2.0d0*t_0 + stroke/celerity
theta = tau*kappa*celerity*((TTCABS/tau)-0.5d0) + H_over_h0
PD = x_paddle_initial + 0.5d0*stroke*(1.0d0 + tanh(theta))
U_P = 0.5d0*stroke*kappa*celerity/((cosh(theta))**2)
XINLET = PD
VITINL = U_P
```

Thus the displacement of the free surface nodes is both in the vertical and horizontal directions – it will be interesting to check that the Discrete Geometric Conservation Law (DGCL) is still met here. The created solitary wave presents good characteristics even if the resulting wave height is higher than its setting.

References

[1] N. Chini, et al., "The impact of sea level rise and climate change on inshore wave climate: A case study for East Anglia (UK)," *Coastal Engineering*, vol. 57, pp. 973-984, 2010.

[2] M. S. Longuet-Higgins and E. D. Cokelet, "The deformation of steep surface waves on water, I. A numerical method of computation," *Proc. R. Soc. Lond. A*, vol. 350, pp. 1-26, 1976.

[3] W. Tsai and D. K. P. Yue, "Computation of nonlinear free-surface flows," *Annual Review of Fluid Mechanics*, vol. 28, pp. 249-278, 1996.

[4] J. M. Floryan and H. Rasmussen, "Numerical methods for viscous flows with moving boundaries," *Applied Mechanics Reviews*, vol. 42, pp. 323-341, 1989.

[5] J. H. Ferziger and M. Peric, *Computational methods for fluid dynamics*, 3rd ed. Springer-Verlag Berlin, 2002.

[6] C. W. Hirt and B. D. Nichols, "Volume of fluid (VOF) method for the dynamics of free boundaries," *Journal of Computational Physics*, vol. 39, pp. 201-225, 1981.

[7] F. H. Harlow and J. E. Welch, "Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface," *Physics of Fluids*, vol. 8, pp. 2182-2189, 1965.

[8] S. Mayer, "A fractional step method for unsteady free-surface flow with applications to non-linear wave dynamics," *International Journal for Numerical Methods in Fluids*, vol. 28.2, pp. 293-315, 1998.

[9] J. L. Thé, G. D. Raithby, and G. D. Stubley, "Surface-adaptative finite-volume method for solving free surface flows," *Numerical Heat Transfer*, vol. Part B, Fundamentals 26.4, pp. 367-380, 1994.

[10] D. Violeau and R. Issa, "Numerical modelling of complex turbulent free-surface flows with the SPH method: an overview," *International Journal for Numerical Methods in*

Fluids, vol. 53, p. 277–304, 2007.

[11] F. Archambeau, N. Méchitoua, and M. Sakiz, "Code_Saturne: a finite volume method for the computation of turbulent incompressible flows - industrial applications," *International Journal on Finite Volumes*, vol. 1.1, p. 1–62, 2004.

[12] H. K. Versteeg and W. Malalasekera, *An introduction to computational fluid dynamics: the finite volume method*, 2nd ed. Pearson, 2007.

[13] P. J. Zwart, "The integrated space-time finite volume method," Ph.D. thesis, University of Waterloo, 1999.

[14] C. M. Rhie and W. L. Chow, "Numerical study of the turbulent flow past an airfoil with trailing edge separation," *AIAA Journal*, vol. 21, pp. 1525-1532, 1983.

[15] R. K. C. Chan, "A generalized arbitrary Lagrangian-Eulerian method for incompressible flows with sharp interfaces," *Journal of Computational Physics*, vol. 17, pp. 311-331, 1975.

[16] F. Archambeau and V. Guimet, "Description et mise en œuvre d'un prototype de module ALE dans le Solveur Commun," Rapport EDF HE-41/99/030/A, 1999.

[17] C. Farhat, "The Discrete Geometric Conservation Law and the Nonlinear Stability of ALE Schemes for the Solution of Flow Problems on Moving Grids," *Journal of Computational Physics*, vol. 174.2, pp. 669-694, 2001.

[18] M. Souli and J. P. Zolesio, "Arbitrary Lagrangian-Eulerian and free surface methods in fluid mechanics," *Computer Methods in Applied Mechanics and Engineering*, vol. 191.31, pp. 451-466, 2002.

[19] I. Demirdzic and M. Peric, "Finite volume method for prediction of fluid flow in arbitrarily shaped domains with moving boundaries," *International Journal for Numerical Methods in Fluids*, vol. 10.7, pp. 771-790, 1990.

[20] D. Apsley and W. Hu, "CFD simulation of two- and three-dimensional free-surface flow," *International Journal for Numerical Methods in Fluids*, vol. 42.5, pp. 465-491, 2003.

[21] F. Archambeau, V. Guimet, and G. Bastin, "Application du prototype de module ALE du

Solveur Commun à des cas de surface libre," Rapport EDF HE-41/99/054/A, 1999.

[22] G. Chabert D'Hières, "Calcul approché du troisième ordre d'un clapotis parfait monochromatique," *Compte rendus de l'Académie des Sciences*, vol. 244, p. 2573, 1957.

[23] R. G. Dean and R. A. Dalrymple, "Water wave mechanics for engineers and scientists," *World Scientific Pub Co Inc*, 1991.

[24] J. H. Duncan, "The breaking and non-breaking wave resistance of a two-dimensional hydrofoil," *Journal of Fluid Mechanics*, vol. 126, pp. 507-520, 1983.

[25] S. Muzaferija, M. Peric, and S. D. Yoo, "Computation of free-surface flows using moving grids," *11th International Workshop on Water Waves and Floating Bodies*, 1996.

[26] Y. Addad and O. Cozzi, "Progress report on the free surface validation with Code_Saturne," University of Manchester ReDAPT Project, July 2010, 2010.

[27] F. Billard, "Near-wall turbulence RANS modeling and its applications to industrial cases," M.Phil. thesis, University of Manchester, 2007.

[28] J. R. Chaplin, "Nonlinear forces on a horizontal cylinder beneath waves," *Journal of Fluid Mechanics*, vol. 147, pp. 449-464, 1984.

[29] T. F. Ogilvie, "First- and second-order forces on a cylinder submerged under a free surface," *Journal of Fluid Mechanics*, vol. 16, pp. 451-472, 1963.