AQUAgpusph - 1.5.02 .

Users guide

JL Cercos-Pita <jl.cercos@upm.es>

August 1, 2014

Contents

Chapter 1

Introduction

1.1 About AQUAgpusph

AQUAgpusph (Another QUAlity GPU-SPH) is a free CFD software licensed under GPLv3, and developed by J.L. Cercos-Pita as part of his PhD at CEHINAV-UPM group [1]. AQUAgpusph is based on Lagragian meshfree SPH (Smoothed Particles Hydrodynamics) method.

AQUAgpusph has been accelerated with OpenCL, that allows you to execute it over CPU based platforms, over GPUs based platforms, and eventually, over every platform developed in the future adapted to the OpenCL standard. For the moment the simultaneous use of several platforms in the same simulation is not supported.

The code has been widely validated, used and documented in several publications, for instance Macià et al. [8] where boundary integrals are tested or Pérez-Rojas and Cercos-Pita [13] where SPHERIC benchmark test case number 9 was simulated. Some examples provided within the package have been studied in these publications.

AQUAgpusph has been designed for UNIX like operative system, and tested on GNU/Debian Linux distributions. For the moment no ports has been developed for Windows or Mac operative systems. AQUAgpusph has been developed in C++ and OpenCL[2]. Also Python extensions has been developed.

1.2 License notes

[1]Model Basin Research group, Technical University of Madrid (UPM), `http://canal.etsin.upm.es`

[2]OpenCL is so quite similar to C

1.3 Objectives of the present document

This document has been created aiming to provide a tool that allows the AQUAgpusph users to know:

1. What is AQUAgpusph program.
2. What you can solve with AQUAgpusph software.
3. How exactly is AQUAgpusph performing the simulations.
   (a) Physical and numerical model.
   (b) Algorithmic used.
4. How you can setup your simulation and run it.
5. How you can post-process the results.

However, this document can complement the developers documentation as well, composed mainly by the Doxygen documentation.

This document is currently in process, and therefore some chapters may be outdated. It is strongly recommended to build the examples in order to know how to setup new simulations.

1.4 Required background

Like many other CFD[3], the usage of the program requires a minimum background on fluid mechanics[4] in order to understand it.

At the other hand, in order to understand all the possibilities that AQUAgpusph offers to you, SPH background is strongly recommended. Nevertheless the SPH method, and more specifically the model internally developed in AQUAgpusph CFD, is detailed described later.

1.5 Organization of this document

The document starts with an introduction in the chapter 1 where AQUAgpusph software is presented, and the present document described.

Then the physical and numerical model internally developed in AQUAgpusph is described in chapter 2. The model introduced contains all the possibilities, including corrections or boundary conditions, but should be the user who's select what of them is applied.

In the chapter 3 the AQUAgpusph install process is documented.

All the options, inputs and commands in order to setup, run, and post-process simulations is documented in chapters 4, 5 and 6. The objective of these chapters is to provide a reference in order to users can know where and how exactly different options, commands or inputs is used, but is not designed as practical section, that is rely to chapter 8 about the examples provided with AQUAgpusph package.

Algorithmic topics are described in the chapter 7, where general information about how AQUAgpusph works

[3]Computational Fluid Dynamics
[4]And the involved mathematics

internally is documented. For further details you can access to the Doxygen documentation, and if you need to go in depth you can ever read the source code.

Chapter 2

Physical and numerical model

2.1 General

In this chapter the problems that AQUAgpusph solves approximately are discussed, describing the governing equations, and the numerical model used so in order that users and developers can refer to this chapter in order to know what AQUAgpusph is solving.

The algorithmic details are exposed later, in chapter 7.

AQUAgpusph is a highly modable software[1], so the numerical model presented here corresponds to the standard provided software.

2.2 Governing equations

2.2.1 Field equations

AQUAgpusph uses weakly-compressible SPH (formerly WCSPH or W-SPH) method in order to approximate the solution of the incompressible Navier-Stokes equations, but since it is a highly modable software, you can adapt it to approximate different equations. In this formulation the incompressibility is sought by modelling the flow with a compressible fluid which, in the flow regime expected, presents very small density fluctuations. The fluid is assumed to be barotropic which implies that the internal energy equation is decoupled from the continuity and momentum equations.

The compressible Navier-Stokes equations for a barotropic fluid in Lagrangian formalism are:

$$\frac{D\rho}{Dt} = -\rho \, \mathrm{div}(\boldsymbol{u}) \tag{2.1}$$

$$\frac{D\boldsymbol{u}}{Dt} = \boldsymbol{g} + \frac{\mathrm{div}(\mathbb{T})}{\rho} \tag{2.2}$$

$$p = p(\rho) \tag{2.3}$$

Where (2.1) is referenced to as continuity or mass conservation equation, (2.2) as the linear momentum conservation one, and (2.3) as the equation of state (EOS).

Monaghan [11] discuss the properly EOS for weakly compressible simulations, purposing an equation such

[1]The functionality, or even the equations solved, can be modified through OpenCL scripts modification

that:

$$p = p_0 + \frac{c_s^2 \rho_0}{\gamma} \left(\left(\frac{\rho}{\rho_0} \right)^\gamma - 1 \right) \tag{2.4}$$

In the introduced equations ρ is the fluid density, ρ_0 is the reference density, p is the pressure, p_0 is the ambient pressure, c_s is the sound velocity and $\boldsymbol{g}$ is a generic external volumetric force field. A specific equation of state (EOS) has been selected. Sound speed c_s will be chosen high enough to get density variations below a certain limit.

The flow velocity, $\boldsymbol{u}$, is defined as the material derivative of a fluid particle position $\boldsymbol{r}$:

$$\frac{D\boldsymbol{r}}{Dt} = \boldsymbol{u} \tag{2.5}$$

$\mathbb{T}$ is the stress tensor of a Newtonian fluid:

$$\mathbb{T} = (-p + \lambda \,\mathrm{tr}\, \mathbb{D})\, \mathbb{1} + 2\mu\, \mathbb{D}\,, \tag{2.6}$$

with $\mathbb{D}$ being the rate of strain tensor, i.e. $\mathbb{D} = (\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T)/2$.

Finally, μ and λ are the viscosity coefficients.

Other formulations can be found in order to compute fully incompressible SPH, but WCSPH has the great advantage that an explicit time integrator can be used to advance in time, and hence no linear system of equations has to be solved in order to get the pressure field at each time step.

2.2.2 Boundary conditions

Denoting the fluid domain by Ω, and the boundary by $\partial\Omega$, boundary conditions can be separated in those to be applied on solid boundaries, $\partial\Omega_B$, and on free surfaces, $\partial\Omega_F$, as shown in the figure 2.1.

Solid boundary conditions

In the solid boundaries, an unpenetrability condition must be imposed, meaning that

$$\boldsymbol{u}_n(\boldsymbol{x}) = \boldsymbol{V}_n(\boldsymbol{x}) \qquad \forall \boldsymbol{x} \in \partial\Omega_B \tag{2.7}$$

where $\boldsymbol{u}_n$ is the fluid velocity projected over the solid normal at point $\boldsymbol{x}$, and $\boldsymbol{v}_n$ is the solid velocity projected over the solid normal.

Regarding to the tangential velocity on the solid, a no-slip condition can be imposed:

$$\left.\boldsymbol{u}_t(\boldsymbol{x})\right|_{\mathrm{no-slip}} = \boldsymbol{V}_t(\boldsymbol{x}) \qquad \forall \boldsymbol{x} \in \partial\Omega_B \tag{2.8}$$

where $\boldsymbol{u}_t$ and $\boldsymbol{v}_n$ are the fluid velocity and solid velocity respectively, both of them projected on the solid tangent. But in some cases Free slip boundary condition can be used, letting that tangential velocity at solid be freely computed.

Free surface boundary conditions

Along the free surface a kinematic and a dynamic boundary conditions must be satisfied. The kinematic boundary condition implies that material points already on the free surface must remain in $\partial\Omega_F$ as this region evolves with the fluid flow.

$$\boldsymbol{u}_n(\boldsymbol{x}) = \boldsymbol{V}_n(\boldsymbol{x}) \qquad \forall \boldsymbol{x} \in \partial\Omega_F \tag{2.9}$$

The dynamic free-surface BC is a consequence of the continuity of the stresses across the free surface. Assuming that surface tension is negligible, a free surface does not stand neither perpendicular normal stresses nor parallel/tangential shear stresses. For a Newtonian fluid, by denoting such stress field as $\boldsymbol{t}$, the dynamic free-surface BC can be expressed as:

$$\boldsymbol{t} = \mathbb{T} \cdot \boldsymbol{n} = \left(-p + \lambda \,\mathrm{tr}\, \mathbb{D} \right) \boldsymbol{n} + 2 \mu \mathbb{D} \cdot \boldsymbol{n} = 0 \,. \tag{2.10}$$

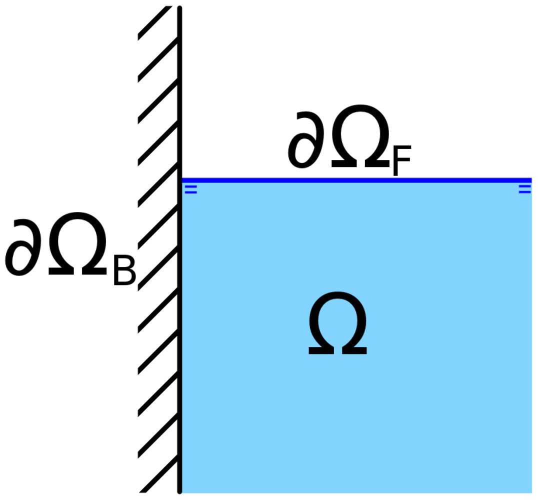


Figure 2.1: Boundary conditions scheme

2.2.3 Initial conditions

Since the governing equations **??** corresponds to an hyperbolic problem, that must be solved as an initial value problem, the pressure, density and velocity fields (right hand sides of the equations involved fields) must be known at a initial time t_0 where a forward time integration will be performed. In this type of problems backward time integration is not generally possible.

The fields in the initial condition must accomplish the governing equations **??**. If the fluid is in rest continuity equation is implicitly accomplished due to the velocity is null, but fields must be selected in order to get null accelerations from momentum equation.

2.3 SPH approximation

2.3.1 General

SPH (Smoothed Particle Hydrodynamics) is a numerical method of simulation created at 1977 independently by Lucy (1977) and Gingold & Monaghan (1977). In this method the fluid is divided into a set of particles for which the fluid equations are written in a Lagrangian form, avoiding mesh requirement. Meshfree character of the method is the most attractive feature because it can be applied to really complex geometries where a mesh generation is not a good option.

SPH was presented as an application of the Monte Carlo method for the resolution of problems of gas dynamics on astrophysics, but was extended to incompressible flows by Monaghan (1994).

2.3.2 Continuous model

In SPH a kernel $W_h(\boldsymbol{x})$ is defined as an even function such that

$$\int_{-\infty}^{\infty} W_h(\boldsymbol{y}) d\boldsymbol{y} = 1 \tag{2.11}$$

For practical purposes, the defined kernel $W_h(\boldsymbol{x})$ vanishes for $|\boldsymbol{x}| > sh$, where s is an integer greater than 0. In AQUAgpusph the Wendland [16] purposed kernel is mainly used, but other ones are supported and can be selected with minor changes in the OpenCL code. In table 2.1 a list with the provided kernels can be found, but the user can incorporate easily other kernels.

The SPH approximation with respect to the kernel $W_h(\boldsymbol{x})$ of a scalar or vector function $f(\boldsymbol{x})$ is defined as

$$\langle f \rangle(\boldsymbol{x}) := \frac{1}{\gamma_h(\boldsymbol{x})} \int_{\boldsymbol{y} \in \Omega} f(\boldsymbol{y}) W_h(\boldsymbol{x} - \boldsymbol{y}) d\boldsymbol{y} \tag{2.12}$$

where $\gamma_h(\boldsymbol{x})$ is the Shepard normalization factor, defined as

$$\gamma_h(\boldsymbol{x}) := \int_{\boldsymbol{y} \in \Omega} W_h(\boldsymbol{x} - \boldsymbol{y}) d\boldsymbol{y} \tag{2.13}$$

with Ω as the ball of radius sh centred on $\boldsymbol{x}$, subtracting eventually the domain out of fluid. In the standard SPH formulation Shepard normalization factor is neglected, considering it $\gamma_h(\boldsymbol{x}) = 1$, that is a good approach far enough of the solid boundary, or near too if fluid extension based boundary conditions are imposed. Shepard correction can produce some instabilities, but improves the consistency and is mandatory for De Leffe's type boundary condition [5, 6], that will be discussed later. On the other hand droping if from the formulation makes it intrinsically conservative [9], that could be a really desirable property.

The gradient and the divergence of functions can be also interpolated

$$\langle \nabla p \rangle(\boldsymbol{x}) = \frac{1}{\gamma_h(\boldsymbol{x})} \int_{\boldsymbol{y} \in \Omega} \nabla p(\boldsymbol{y}) W_h(\boldsymbol{x} - \boldsymbol{y}) d\boldsymbol{y} \tag{2.14}$$

$$\langle \mathrm{div}(\boldsymbol{u}) \rangle(\boldsymbol{x}) = \frac{1}{\gamma_h(\boldsymbol{x})} \int_{\boldsymbol{y} \in \Omega} \mathrm{div}(\boldsymbol{u}(\boldsymbol{y})) W_h(\boldsymbol{x} - \boldsymbol{y}) d\boldsymbol{y} \tag{2.15}$$

In these formulas ∇p and $\mathrm{div}(\boldsymbol{u})$ are not known. Integrating by parts the differential operators can be conveniently moved to the kernel function. We will work only over the gradient operator to show the procedure but analogous treatment is applied to the divergence one.

$$\langle \nabla p \rangle(\boldsymbol{x}) = \frac{1}{\gamma_h(\boldsymbol{x})} \left(\int_{\boldsymbol{y} \in \Omega} \nabla \left(p(\boldsymbol{y}) W_h(\boldsymbol{x} - \boldsymbol{y}) \right) d\boldsymbol{y} - \int_{\boldsymbol{y} \in \Omega} p(\boldsymbol{y}) \nabla W_h(\boldsymbol{x} - \boldsymbol{y}) d\boldsymbol{y} \right) \tag{2.16}$$

If $W_h(\boldsymbol{x})$ is an even function, $\nabla W_h(\boldsymbol{x})$ is an antisymmetric one, so we can use this property to switch the integral terms signs. Also we can use the divergence theorem over the first integral term.

$$\langle \nabla p \rangle(\boldsymbol{x}) = \frac{1}{\gamma_h(\boldsymbol{x})} \left(\int_{\boldsymbol{y} \in \Omega} p(\boldsymbol{y}) \nabla W_h(\boldsymbol{y} - \boldsymbol{x}) d\boldsymbol{y} - \int_{\boldsymbol{y} \in \partial\Omega} p(\boldsymbol{y}) W_h(\boldsymbol{y} - \boldsymbol{x}) \boldsymbol{n}(\boldsymbol{y}) dS(\boldsymbol{y}) \right) \tag{2.17}$$

Where $\partial\Omega$ denotes the boundary of Ω. The contour term has been traditionally neglected in the literature because authors assume that $p = 0 \in \partial\Omega$[2]. In AQUAgpusph this boundary integral and the Shepard correction term can be retained.

Regarding Laplacian operator, The 2 most popular formulations to approximate it are Monaghan and Gingold [10] form and Morris et al. [12] form. Colagrossi et al. [4] demonstrated that only the first one provides right dissipation in the presence of a free surface when boundary terms are discretized, so in AQUAgpusph the second way has not been implemented.

Assuming that the viscosity coefficients are constant all over the fluid domain, the continuous formulation of the Monaghan viscous term is:

$$\langle \triangle \boldsymbol{u} \rangle (\boldsymbol{x}) = \frac{\mu K}{\gamma_a(\boldsymbol{x}) \, \rho(\boldsymbol{x})} \int_{y \in \Omega} \frac{(\boldsymbol{u}(\boldsymbol{y}) - \boldsymbol{u}(\boldsymbol{x})) \cdot (\boldsymbol{y} - \boldsymbol{x})}{|\boldsymbol{y} - \boldsymbol{x}|^2} \nabla W_h(\boldsymbol{y} - \boldsymbol{x}) \, d\boldsymbol{y} \tag{2.18}$$

where K is a parameter depending on the spatial dimension ($K = 6, 8, 15$, respectively in $1D$, $2D$ and $3D$).

This viscous term is valid far enough of the boundary, and provides right dissipation near the free surface. Boundary viscous term is added as described by Macià et al. [8] in order to can impose no-slip boundary conditions in the boundaries.

$$\begin{aligned} \langle \triangle \boldsymbol{u} \rangle (\boldsymbol{x}) \;=\; & \frac{\mu}{\gamma_a(\boldsymbol{x}) \, \rho(\boldsymbol{x})} \left(\int_{y \in \Omega} K \frac{(\boldsymbol{u}(\boldsymbol{y}) - \boldsymbol{u}(\boldsymbol{x})) \cdot (\boldsymbol{y} - \boldsymbol{x})}{|\boldsymbol{y} - \boldsymbol{x}|^2} \nabla W_h(\boldsymbol{y} - \boldsymbol{x}) \, d\boldsymbol{y} \right. \\ & \left. - \int_{y \in \partial\Omega} (\boldsymbol{u}(\boldsymbol{y}) - \boldsymbol{u}(\boldsymbol{x})) \frac{(\boldsymbol{y} - \boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{y})}{|\boldsymbol{y} - \boldsymbol{x}|^2} W_h(\boldsymbol{y} - \boldsymbol{x}) \, dS(\boldsymbol{y}) \right) \end{aligned} \tag{2.19}$$

All this smoothing procedures introduces errors in the representation of functions and operators. These errors goes to zero as $h \rightarrow 0$, but the convergence order may be different [7, 8].

Kernel	Support
Wendland	$2h$
Cubic spline	$2h$
Gaussian	$3h$

Table 2.1: AQUAgpusph provided kernels.

2.3.3 Discretized model

For practical purposes the integrals present on continuous SPH interpolation are solved numerically, so another error must be considered due to the discretization [14, 1]. In order to do it, the space will be discretized placing a set of particles that is treated in a Lagrangian point of view. $d\boldsymbol{y}$ must be then discretized as the volume of the particles, that can be rewritten such that

$$d\boldsymbol{y} \approx \frac{m_b}{\rho_b} \tag{2.20}$$

allowing us to discretize the operators.

$$\langle \nabla p \rangle_a = \frac{1}{\gamma_a} \left(\sum_{b \in \text{Fluid}} \frac{p_b}{\rho_b} \nabla W_{ab} m_b - \sum_{b \in \text{Boundary}} p_b W_{ab} \boldsymbol{n}_b S_b \right) \tag{2.21}$$

[2]The almost used boundary conditions along the solid walls in SPH have consists in fluid extensions, so the contour $\partial\Omega$ has been rely only for the free surface

In the equation 2.21 the discretized version of the SPH gradient operator over the pressure field is shown, where the value is computed interpolating it from the neighbour particles and, eventually, from the neighbour wall elements.

In the figure 2.2 a scheme about the interpolation over a particle labelled "a" is shown, the index b of the first sum moves along all the fluid particles, but since the kernel nullifies at a distance $s\,h$, only the particles near enough will be considered.

In the figure 2.3 2D Wendland kernel value is shown. Both kernel value W and gradient value ∇W nullifies for distances $|\boldsymbol{x}| > 2\,h$. Must be noticed that ∇W nullifies also for $|\boldsymbol{x}| = 0\,\mathrm{m}$, but not the kernel value.

As introduced previously, the weight function W is even, but the gradient ∇W is an odd function.

In the same way Laplacian operator can be also discretized:

$$\begin{aligned}\langle \triangle \boldsymbol{u} \rangle_a &= \frac{\mu}{\rho_a \gamma_a} \left(\sum_{b\in\mathrm{Fluid}} K \frac{(\boldsymbol{u}_b - \boldsymbol{u}_a)\cdot(\boldsymbol{x}_b - \boldsymbol{x}_a)}{\rho_b\, |\boldsymbol{x}_b - \boldsymbol{x}_a|^2} \nabla W_{ab}\, m_b \right. \\ & \left. \sum_{b\in\mathrm{Boundary}} (\boldsymbol{u}_b - \boldsymbol{u}_a) \frac{(\boldsymbol{x}_b - \boldsymbol{x}_a)\cdot \boldsymbol{n}_b}{|\boldsymbol{x}_b - \boldsymbol{x}_a|^2} W_{ab}\, S_b \right)\end{aligned} \tag{2.22}$$

Since the divergence operator is so quite similar to the gradient, is not introduced here due to will be discussed a little bit more in the following section.

So in the SPH discretized model 2 main spatial variables must be selected:

1. Kernel height h: Set the ball where values will be taken in order to perform the smoothed interpolation. as this value goes to zero, the weight function converges to the Dirac's delta one, so converges to the real solution as well.

2. Distance between particles r: Set the resolution of the numerical integrals approximation, so in the limit of $r \to 0\,\mathrm{m}$, the integrals are exactly computed.

Since r must be lower than interaction distance $s\,h$, usually this 2 key values are changed by h and h/r (formerly h_{fact}), or by r and h/r[3]. The kernel height defines the problem resolution therefore, and decreasing this value the number of particles increase improving the result resolution as values smoothed interpolation (considering that h_{fact} is preserved constant). The distance between particles, expressed as the ratio h/r, defines the number of neighbours that will interact with each particle, so increasing this value the number of interactions will increase, and the integrals approximation will be improved.

$$\lim_{h\to 0;\ \frac{h}{r}\to\infty} \langle p \rangle = p \tag{2.23}$$

The problem complexity is $\mathrm{f}\left((1/h)^3 \cdot (h/r)^3\right)$ for a $3D$ simulation (time integration associated complexity is not considered yet) therefore.

[3]In AQUAgpusph this second approach is applied

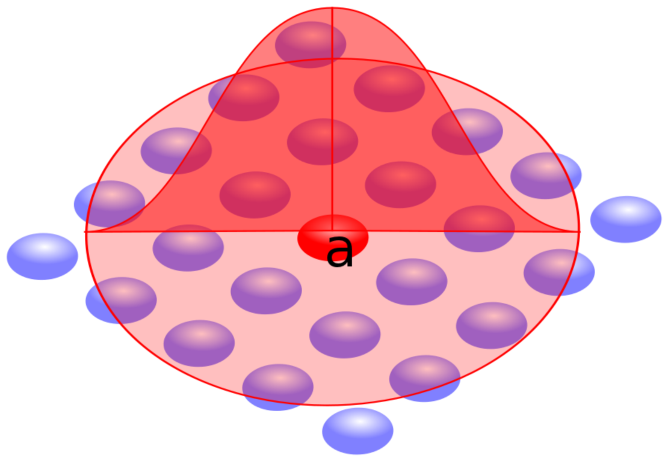


Figure 2.2: Discrete SPH interpolation scheme

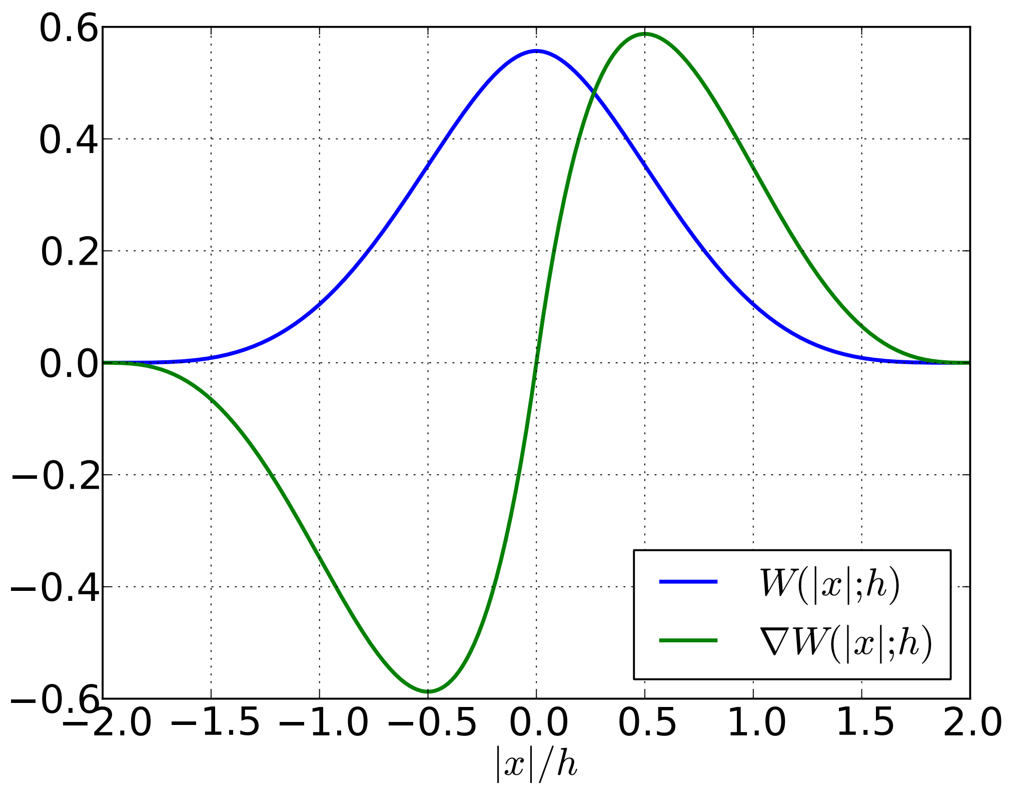


Figure 2.3: 2D Wendland Kernel value (and gradient), $h = 1$m

2.3.4 AQUAgpusph discretized operators used

For conservation considerations, that you can find on [9] and [4], the operators used in AQUAgpusph are not the shown on previous section, but are conveniently modified (formerly simmetrized) as

$$
\begin{aligned}
\left\langle \frac{\nabla p}{\rho} \right\rangle_a &= \frac{1}{\gamma_a} \left(\sum_{b \in \text{Fluid}} \left(\frac{p_a}{\rho_a^2} + \frac{p_b}{\rho_b^2} \right) \nabla W_{ab} m_b \right. \\
&\quad \left. - \sum_{b \in \text{Boundary}} \rho_b \left(\frac{p_a}{\rho_a^2} + \frac{p_b}{\rho_b^2} \right) W_{ab} \boldsymbol{n}_b \, S_b \right) \\
\langle \rho \, \text{div}(\boldsymbol{u}) \rangle_a &= \frac{1}{\gamma_a} \left(\sum_{b \in \text{Fluid}} (\boldsymbol{u}_b - \boldsymbol{u}_a) \nabla W_{ab} m_b \right. \\
&\quad \left. - \sum_{b \in \text{Boundary}} \rho_b \, (\boldsymbol{u}_b - \boldsymbol{u}_a) \cdot n_b W_{ab} \, S_b \right) \\
\langle \triangle \boldsymbol{u} \rangle_a &= \frac{\mu}{\rho_a \, \gamma_a} \left(\sum_{b \in \text{Fluid}} K \frac{(\boldsymbol{u}_b - \boldsymbol{u}_a) \cdot (\boldsymbol{x}_b - \boldsymbol{x}_a)}{\rho_b \, |\boldsymbol{x}_b - \boldsymbol{x}_a|^2} \nabla W_{ab} \, m_b \right. \\
&\quad \left. \sum_{b \in \text{Boundary}} (\boldsymbol{u}_b - \boldsymbol{u}_a) \frac{(\boldsymbol{x}_b - \boldsymbol{x}_a) \cdot \boldsymbol{n}_b}{|\boldsymbol{x}_b - \boldsymbol{x}_a|^2} W_{ab} \, S_b \right)
\end{aligned}
\tag{2.24}
$$

The operators shown above results from the divergence theorem application over the operators to interpolate, and have the main advantage that are ever convergent for constant fields (of pressure and velocity).

Weakly compressible SPH formulation carry some high frequency pressure oscillations caused by the small density fluctuations with the huge incompressibility requirements. In order to solve partially this effect the density field, that results from a evolution process, can be reinitialized from particles positions using the property

$$
\langle \rho \rangle_a = \frac{1}{\gamma_a} \sum_{b \in \text{Fluid}} W_{ab} m_b \tag{2.25}
$$

That allows to recover a smooth density field. This correction is not usually recommended due to can carry some instabilities, but when applied, in almost cases this correction is only used letting some time steps between corrections. In AQUAgpusph you can select how many steps must run before apply this correction, or simply don't apply it.

2.3.5 Boundary conditions

Free surface boundary conditions

Free surface kinematic and dynamic boundary conditions are automatically accomplished in the W-SPH formulation, so only the consistency of the operators must be guaranteed. Considering null the ambient pressure p_0 in the state equation of the governing ones **??** no pressure extra condition has to be imposed at the free surface as Colagrossi et al. [4] demonstrates.

Unfortunately near the free surface gradient, divergence an Laplacian operators shown in the equations 2.24 are convergent, but not consistent due to the result converges to a wrong value. The half good news is that, since the results converges, and the region where the results are inconsistent is confined to a distance $s\,h$ of the free surface, in a integral point of view the operators introduced converges to the right solution with order

$O(h)$.

In order to get consistent gradient, divergence an Laplacian operators near the free surface, as the most desirable ones, track process over the free surface is needed, with a significant regression in the computational efficiency and robustness.

So, assuming an inconsistency near the free surface that is bounded, with an ambient pressure such that $p_0 = 0\,\mathrm{Pa}$, no additional conditions needs to be imposed in the free surface therefore.

All these assumptions are refereed to monophasic simulations, that are the mainly used in AQUAgpusph simulations.

Solid boundary conditions

Regarding solid body boundary conditions AQUAgpusph allows to use most popular SPH to modeling techniques:

1. **Fixed particles**: Also known as dummy particles, this method consists of extending the fluid domain across the wall with fluid particles, fixing their motion. This is the boundary condition traditionally used on SPH method.

2. **Ghost particles**: This method consist on extending the fluid domain across the wall as well, but in this case the fluid at the other side is obtained as mirroring process over the fluid domain. It has been developed as an improvement of the previous one.

3. **Elastic bounce**: In this method the particles near to the wall, that will trespass it, are treated with an elastic bounce model, where an elastic bounce factor is used to set the amount of energy conserved in the interaction. This boundary condition is not designed to use it alone, but can be mixed with the other ones in order to can manage the particles that will cross over the walls without crashing the simulation.

4. **Boundary integrals**: Introduced by De Leffe et al. [5], and formalized by Ferrand et al. [6], in this method the boundary integrals are numerically solved along the walls. Is the most recent one, but the consistency has been probed by Macià et al. [8], with the examples performed with AQUAgpusph software.

Each boundary condition type has some advantages, and some combinations of them can be applied in your simulations. Boundary conditions introduced are described and discussed in detail at section 7.7, where their algorithmic details are discussed.

2.3.6 Initial condition

As introduced in section 2.3.3, you must provided a set of particles in the initial time instant t_0. The initial condition must accomplish the governing equations **??**. The most frequently used initial condition is a fluid in rest, so we are focused in this particular case, describing the process to generate the particles.

Since the fluid is in rest, and velocity is null therefore, so the continuity equation is implicitly accomplished. Regarding the momentum equation, if a volumetric force exist (for instance the gravity force), a pressure gradient must be forced in order to get null accelerations (formerly known as the hydrostatic pressure), such that

$$p = p_0 + \rho_0\, \boldsymbol{g} \cdot \boldsymbol{z} \tag{2.26}$$

Where $p_0 = 0\,\text{Pa}$, and $z = 0\,\text{m}$ in the free surface. But the state equation impose a relation between the density field and the pressure field, so the density field provided by the user must be set as follows (pressure field is not an input variable but is computed using the EOS 2.4 when needed):

$$\rho = \rho_0 \left(\frac{\gamma \boldsymbol{g} \cdot \boldsymbol{z}}{c_s^2} + 1 \right)^{\frac{1}{\gamma}} \tag{2.27}$$

SPH method, as described in section 2.3.2, is based on the hypothesis that the differential of volume $d\boldsymbol{y}$ is constant, so the particles volume must be constant as well. In order to preserve this main condition the particles mass must be computed as follow:

$$m = \frac{\mathcal{V}}{\rho} \tag{2.28}$$

Where $\mathcal{V}$ is the volume associated to each particle, that in the case of a 3D Cartesian particles distribution with a distance between particles $\boldsymbol{r}$, allows us to write that

$$\mathcal{V} = \boldsymbol{r}^3 m = \frac{\boldsymbol{r}^3}{\rho} \tag{2.29}$$

2.3.7 Time numerical integration

In order to approximate numerically the solution of the problem the time integration is discretized as well.

One of the key features of the W-SPH model is that is a purely explicit formulation, so a variable can be defined in a time instant t_{n+1} as function of known data in a time instant t_n, i.e.:

$$X\bigg|_{t_{n+1}} = \mathrm{F}\left(X\bigg|_{t_n}, \frac{dX}{dt}\bigg|_{t_n}, \frac{d^2X}{dt^2}\bigg|_{t_n}, \cdots ; \Delta t \right) \tag{2.30}$$

The function F is known as the scheme. Better schemes increase the convergence rate, so allows to increase the time step in order to get similar errors[4], however almost high order time integrating schemes requires several derivatives computation, that is too computational expensive in SPH. In AQUAgpusph a quasi-second order method, that only requires one derivative computation per time step, Leap-Frog method is used [15], with the main advantage that only one time derivative is needed by each time step.

1. Predictor:

$$\begin{aligned} \dot{\boldsymbol{x}}_{t+dt}^{\text{pred}} &= \dot{\boldsymbol{x}}_t + dt\,(\ddot{\boldsymbol{x}}_t + g) \\ \boldsymbol{x}_{t+dt}^{\text{pred}} &= \boldsymbol{x}_t + dt\,\dot{\boldsymbol{x}}_t + \frac{dt^2}{2}\,(\ddot{\boldsymbol{x}}_t + g) \\ \rho_{t+dt}^{\text{pred}} &= \rho_t + dt\,\dot{\rho}_t \end{aligned} \tag{2.31}$$

2. SPH interactions:

$$\begin{aligned} \ddot{\boldsymbol{x}}_{t+dt} &\leftarrow \text{SPH} \\ \dot{\rho}_{t+dt} &\leftarrow \text{SPH} \end{aligned} \tag{2.32}$$

3. Corrector:

$$\begin{aligned} \dot{\boldsymbol{x}}_{t+dt} &= \dot{\boldsymbol{x}}_{t+dt}^{\text{pred}} + \frac{dt^2}{2}\,(\ddot{\boldsymbol{x}}_{t+dt} - \ddot{\boldsymbol{x}}_t) \\ \boldsymbol{x}_{t+dt} &= \boldsymbol{x}_{t+dt}^{\text{pred}} \\ \rho_{t+dt} &= \rho_{t+dt}^{\text{pred}} + \frac{dt}{2}\,(\dot{\rho}_{t+dt} - \dot{\rho}_t) \end{aligned} \tag{2.33}$$

[4]One of the critical points in the SPH performance is the small time steps involved

In order to get a stable time integration time step must be selected such that

$$dt \leq \frac{h}{\max(10\,|\boldsymbol{u}|,\; c_s)} \tag{2.34}$$

In the section 2.3.3 the complexity of one time step has been introduced, but of course this complexity increases linearly with the time steps required, so 3 algorithmic complexities must be considered:

1. Number of particles N: The number of particles is conditioned by the kernel height h, and defines the spatial resolution implying a complexity of $O(3)$.
2. Number of neighbours M: The number of neighbours is conditioned by the ratio h/r, and defines the quality of the numerical integrals approximation implying a complexity of $O(3)$.
3. Number of time steps: The number of time steps is conditioned by the time step dt, that depends directly on kernel height h, implying a complexity of $O(1)$.

So, if we increase the number of particles but not the number of neighbours, time step will be reduced and then complexity goes as N^4.

At the other hand, if the number of particles is constant but the number of neighbours is increased, kernel height increases therefore resulting in higher time steps, so the complexity goes as M^2.

So the method complexity goes as $N^4\,M^2$, that shows that you must be careful with the convergence because time consumed by the simulations will grows too fast ($O(6)$ in the worst case).

Chapter 3

Install AQUAgpusph

3.1 General

AQUAgpusph can be downloaded from

`http://canal.etsin.upm.es/aquagpusph/descargas.php`

The process to configure, build and install is described below. The files uploaded to this web page are the latest stable version, but optionally you can download the developers version, that has more fresh features, but can carry some bugs and unstabilities. Developers version is allocated in github, so you must have git installed on your system to access to the source code. To download the developers package version execute

```
git clone git://github.com/jlcercos/aquagpusph.git
```

that will generate a folder called “aquagpusph” with the package inside.

The instructions included below are valid for both the stable and the developers versions, but some additional dependencies and configuration can be missed.

For the moment no specific GNU/Linux distributions installable packages have been created.

3.2 Dependencies

AQUAgpusph has some dependencies that must be installed before. Almost dependencies are mandatory, but exist other dependencies that can be optionally ignored depending on the needed capabilities on AQUAgpusph built program.

Dependencies are classified in 4 categories therefore:

1. Mandatory: AQUAgpusph needs the dependency available on system in order to run.
2. Optional: If dependency is not available, AQUAgpusph can be built, but some features will be disabled.
3. Recommended: If dependency is not available, AQUAgpusph can be built with all the features, but some additional package characteristics may not run.
4. Suggested: AQUAgpusph can have some synergies with other softwares, for preporcessing or postprocessing for instance.

In table 3.1 you can see the list of dependencies required. Each dependency can require more other dependencies that will not be documented. Take into account the following considerations:

1. OpenCL is a little bit special dependency due to is really hardware depending, therefore at the web page only OpenCL specification will found. We encourage you to contact your hardware provider in order to know the best way to install it.

2. NCurses can be used to perform smart screen output, that can revert on slightly reduced computational time. Any relevant functionality will be lost if AQUAgpusph is compiled without NCurses support therefore.

3. While H5Part was the main AQUAgpusph output format in previous versions, it has been replaced by VTK files (H5Part has been declared as an outdated format).

4. Doxygen and Graphviz are only needed if you want to build Doxygen developers documentation. This documentation is mainly oriented to developers, but can be really useful for users in some number of cases, so is strongly recommended to build it.

Dependency	Status	Web page
CMake	Mandatory	`http://www.cmake.org`
xerces-c	Mandatory	`http://xerces.apache.org/xerces-c`
Python	Mandatory	`http://www.python.org`
OpenCL	Mandatory	`http://www.khronos.org/opencl`
Eigen3	Mandatory	`http://eigen.tuxfamily.org/index.php`
libmatheval	Mandatory	`https://www.gnu.org/software/libmatheval`
Ncurses	Optional	`http://www.gnu.org/software/ncurses/ncurses.html`
VTK	Optional	`http://www.vtk.org`
Doxygen	Recommended	`http://www.doxygen.org`
Graphviz	Recommended	`http://www.graphviz.org`
matplotlib	Suggested	`http://matplotlib.org`
numpy	Suggested	`http://www.numpy.org`
PyQt4	Suggested	`http://www.riverbankcomputing.com/software/pyqt/intro`
gnuplot	Suggested	`http://www.gnuplot.info`
ParaView	Suggested	`http://www.paraview.org/`

Table 3.1: AQUAgpusph dependencies.

3.3 Install

3.3.1 General

AQUAgpusph use CMake to the configure, build and install process. CMake installation process is performed in 3 steps therefore:

1. **CMake configuration**: In this stage main options and features that the built version of AQUAgpusph will have.

2. **Compilation**: In this step the source code is compiled into a binary ready to be launched.

3. **Install**: This step is optional, aiming to install the software on the system in order to be available for all the users of the computer.

To perform the CMake configuration process 2 ways are described.

3.3.2 CMake configuration

To start configuring CMake you can run the following command on the folder where you downloaded AQUAgpusph sources:

```
cmake .
```

That will configure AQUAgpusph with default options. You can modify some options, for instance, to change installation prefix from /usr/local to /usr folder, and build Release version, you can launch following command:

```
cmake . -D CMAKE_INSTALL_PREFIX:PATH=/usr -D CMAKE_BUILD_TYPE:STRING=Release
```

In `http://www.cmake.org/Wiki/CMake_Useful_Variables` web page you can find a list with the most commonly CMake used options. Also AQUAgpusph provides some additional options:

1. **AQUAGPUSPH_3D**: ON to build 3D AQUAgpusph version, OFF to build 2D version. AQUAgpusph 2D and 3D versions can be installed on the same system. Be mindful that 3D version must not be used to perform 2D simulations.
2. **AQUAGPUSPH_BUILD_DOC**: ON to build Doxygen developers documentation, OFF otherwise. Requires Doxygen and Graphviz packages installed in the system.
3. **AQUAGPUSPH_BUILD_EXAMPLES**: ON to generate the examples. Generated examples are the associated to the selected 2D/3D version.
4. **AQUAGPUSPH_USE_NCURSES**: ON to build AQUAgpusph with NCurses output terminal, OFF otherwise. NCurses has more efficient screen streamed functionality, so you can win some performance using it.
5. **AQUAGPUSPH_USE_VTK**: ON to build AQUAgpusph with VTK output support, OFF otherwise. In AQUAgpusph - 1.5.02 VTK is the only valid output format for the particles files, so it is mandatory to can process visualizations of the simulations.

3.3.3 CMake configuration with ccmake

I order to simplify the process you can use ccmake[1] to configure AQUAgpusph with a more friendly user interface (Of course you must install ccmake before). Simply launch the command:

```
ccmake .
```

If you did not configure AQUAgpusph before, an empty cache page will shown up, press **'c'** key to perform initial default configuration. After some seconds working, errors, warnings, and most relevant selected options report will be returned; press **'e'** key to continue.

Now some options are shown in order to allow you to edit them. Probably you want to edit CMAKE_INSTALL_PREF too. Take care about CMAKE_BUILD_TYPE, Debug mode is a lot of more resources consuming and slow, therefore it's strongly recommended to let Release mode for almost users.

After this, press **'c'** key in order to configure the package again. Repeat the process until no new options are shown (that are marked with an asterisk), and no more errors are reported. At this point ccmake will offer generating final project pressing **'g'** key.

Same options introduced in 3.3.2 can be set using ccmake.

[1]Or eventually CMake-GUI

3.3.4 Compilation

After the configuration process you can start the compilation in order to create the binary. You can launch the following command:

```
make all
```

Since AQUAgpusph build process is not really time consuming parallel compiling process is not generally needed, but of course you can set several processors building simultaneously. For instance, if you want to use 8 cores you can type:

```
make -j8
```

Additionally, if you want to get more info about the building process, you can use VERBOSE flag:

```
make VERBOSE=1
```

This will report the command executed to compile/generate each file.

3.3.5 Install

AQUAgpusph can be used without installing it, see generated examples to learn more about this, but if AQUAgpusph must be provided to several users probably you want to install it typing:

```
make install
```

Note that depending the selected installation folder you may need administrative permissions.

Chapter 4

Case setup options glossary

4.1 General

AQUAgpusph has a powerful XML files input interface used to setup simulation cases. Those files can be easily structured as a tree with the including capabilities. XML is a standard file format where a schematic structure of the files is allowed. You can learn more about XML syntax and possibilities at next web page:

`http://www.w3.org/XML`

The objective of this chapter is to show all the available settings and switches that can be established for a AQUAgpusph simulation, but for a practical approach to generating and running simulations please refer to chapter 8, where AQUAgpusph package examples are discussed.

4.2 XML definition

4.2.1 General

Each XML file must have the following structure:

```
<?xml version="1.0" ?>
<sphInput>
    ...
</sphInput>
```

Of course all the XML valid syntax can be used in these files (i.e.- Comments, ...). In the table 4.1 all the valid options that can be set at sphInput group are described. In the table 4.2 the valid sections into sphInput group are described. You can add several several times each section which, depending on the settings affected, will overwrite or add the information. The different group tags are now discussed.

Tag	Attributes	Valid values	Description
Include	file	XML file path	Specified XML file will processed before continuing parsing this file. You can include as many files as you want.

Table 4.1: Valid option tags in the sphInput group.

4.2.2 Settings group

This group is used to set some AQUAgpusph general settings, that are not really related to specific simulation. Table 4.3 shows all the valid options that can be set in this group.

4.2.3 OpenCL

This group is used to set the OpenCL kernels that will be used to perform the simulation. Table 4.4 shows all the valid options that can be set in this section.

AQUAgpusph provides a version of each OpenCL kernel, but you can create custom kernels in order to modify the software behaviour conveniently, with the restriction that new kernels must preserve input parameters (sorted as the original one).

4.2.4 Timing group

This group is used to set simulation time stepping controls. Table 4.5 shows all the valid options that can be set in this section.

The minimum time step and the Courant factor are not affecting if the time step is not selected as "Variable".

4.2.5 SPH

This group is used to set SPH method related settings. Table 4.6 shows all valid options that can be set in this section.

γ value is a factor involved in the EOS 2.4.

Sound speed must be selected, at least, 10 times greater than the maximum expected fluid velocity along the simulation. Sound speed affects directly to the fluid compressibility, so higher sound speed implies simulation

Tag	Attributes	Valid values	Description
Settings	*none*	*none*	General AQUAgpusph parameters can be set here.
OpenCL	*none*	*none*	OpenCL kernels to compute can be selected here.
Timing	*none*	*none*	Simulation time control directives are set here.
SPH	*none*	*none*	SPH method relevant data must be provided here.
Fluid	n	Number of particles	Fluid data must be provided in this section. Each Fluid section will add new fluid to simulation.
Sensors	*none*	*none*	Sensor (or probes) can be specified in this section.
Movements	*none*	*none*	Motions to apply to the boundaries can be set here.
GhostParticles	*none*	*none*	Ghost particles boundary can be set here.

Table 4.2: Valid section tags in the sphInput group.

Tag	Attributes	Valid values	Description
Verbose	level	0	No verbose.
		1	Relevant data will be printed each frame.
		2	Relevant data will be printed each time step.
Device	platform	id	The OpenCL platform to use.
	device	id	The device (inside the platform) to use.
	type	ALL	All the types of variables can be chosen.
		CPU	Just CPU devices will be available.
		GPU	Just GPU devices will be available.
		ACCELERATOR	Just accelerators will be available.
		DEFAULT	Just the default OpenCL device will be available.

Table 4.3: Valid settings option tags.

Tag	Attributes	Valid values	Description
Predictor	file	File path	Kernel file used at Leap-frog predictor stage. File extension (.cl) will appended automatically.
LinkList	file	File path	Kernel file used to perform the Link-List, i.e.- In-cell particles localization. File extension (.cl) will appended automatically.
Rates	file	File path	Kernel file used to perform particles interaction. File extension (.cl) will appended automatically.
Corrector	file	File path	Kernel file used at Leap-frog corrector stage. File extension (.cl) will appended automatically.
TimeStep	file	File path	Kernel file used to compute time step. File extension (.cl) will appended automatically.
Reduction	file	File path	Kernel file used to perform reductions of variables File extension (.cl) will appended automatically.
RadixSort	file	File path	Kernel file used to sort particles by cell location. File extension (.cl) will appended automatically.
Shepard	file	File path	Kernel file used to perform 0th order correction. File extension (.cl) will appended automatically.
Domain	file	File path	Kernel file used to find and destroy particles out of domain. File extension (.cl) will appended automatically.
ElasticBounce	file	File path	Kernel file used to perform simple wall boundary condition. File extension (.cl) will appended automatically.
DeLeffe	file	File path	Kernel file used to perform DeLeffe wall boundary condition. File extension (.cl) will appended automatically.
DensInterpolation	file	File path	Kernel file used to perform density reinitialization. File extension (.cl) will appended automatically.
Torque	file	File path	Kernel file used to compute force and torque. File extension (.cl) will appended automatically.
Bounds	file	File path	Kernel file used to compute fluid bounds. File extension (.cl) will appended automatically.
Energy	file	File path	Kernel file used to compute fluid energy. File extension (.cl) will appended automatically.
Portal	file	File path	Kernel file used to perform particles teleporting. File extension (.cl) will appended automatically.

Table 4.4: OpenCL kernels tags.

Tag	Attribute	Values	Attributes	Values	Attributes	Values
Option	name	Start			value	Starting time.
Option	name	Stop	type	Time	value	End time.
				Frames	value	Number of frames.
						(Can be mixed with time stop criteria)
	name	Timestep	value	Variable		(Time step recomputed)
				Fix	value	(Time step computed at start)
				time [s]		
	name	MinTimeStep			value	Minimum allowed time step
	name	Courant			value	Courant factor
	name	LogFile	type	No		
				FPS	value	Frames per second.
				IPF	value	Iterations per frame.
						(Can be mixed with FPS)
	name	EnFile	type	No		
				FPS	value	Frames per second.
				IPF	value	Iterations per frame.
						(Can be mixed with FPS)
	name	BoundsFile	type	No		
				FPS	value	Frames per second.
				IPF	value	Iterations per frame.
						(Can be mixed with FPS)
	name	Output	type	No		
				FPS	value	Frames per second.
				IPF	value	Iterations per frame.
						(Can be mixed with FPS)

Table 4.5: Time control valid tags.

improvements, but is too much computing expensive (due to a time step reduction).

In the equation 2.34 the time step that must be selected is described (how to setup the time step is described in the section 4.2.4), but in order to reduce the Courant number you can setup a time step divisor **DivDt**.

LLSteps steps is the number of time steps between Link-List computations, so a small performance can be win. For almost cases is strongly recommended to let **LLSteps** = 1.

The density interpolation (formerly density reinitialization) has been described in the section 2.3.4. With **DensSteps** option you may set the number of time steps that will be computed before each density interpolation.

The boundary conditions has been introduced in the section 2.3.5.

Finally domain can be used to destroy the particles out of it, transforming them into 0 mass fixed particles.

Tag	Attribute	Values	Attributes	Values
Option	name	gamma	value	γ Batchelor 67 value
		g	x	x vector component.
			y	y vector component.
			z	z vector component.
		hfac	value	$\frac{h}{dx}$
		deltar	x	dx
			y	dy
			z	dz
		cs	value	sound speed
		DivDt	value	dt scale factor
		refp	value	p_0
		LLSteps	value	Number of steps
		DensSteps	value	Number of steps (0 to disable it)
		Boundary	value	ElasticBounce FixedParticles DeLeffe
		SlipCondition	value	NoSlip FreeSlip
		BoundDist	value	Minimum distance to wall.
		BoundElasticFactor	value	Elastic interaction factor.
		Shepard	value	None Force Dens ForceDens
		Domain	x	Starting x coordinate
			y	Starting y coordinate
			z	Starting y coordinate
			l	Length
			b	Breadth
			h	Height

Table 4.6: SPH method available options.

4.2.6 Fluid

This group is used to add fluids, i.e. each instance of this group will be considered as a new fluid. **Fluid** tag must have the attribute **n** specifying the total number of particles and vertices involved. Table 4.7 shows all valid options that can be set in this section.

μ must be the real dynamic viscosity of the fluid, but since weakly compressible SPH is a purely explicit method, will turn unstable if the α value don't reach a minimum value, μ can be corrected depending on α value set [1].

Regarding the particles file allowed formats are discussed in the section 4.3.

Tag	Attribute	Values	Attributes	Values
Option	name	gamma	value	γ Batchelor 67 value
		refd	value	ρ_0
		Viscdyn	value	μ
		alpha	value	$\alpha \geq 8\frac{\mu_{sim}}{\rho c_s h}$
Load	file	Fluid particles and vertices file.		

Table 4.7: Fluid generation options.

4.2.7 Sensors

This group is used to set sensors. Sensors interpolates and print several fields on the selected point, naming:

1. **Position**: Sensor position.
2. **Pressure**: Pressure [Pa].
3. **Density**: Density [kg/m^3].
4. **Kernel completion**: $\sum\limits_{b\in\text{fluid}} \frac{W_{ab}}{\rho_b} m_b$.

You can add as many sensors as you want adding several instances of the **Sensor** tag. AQUAgpusph provides an OpenCL kernel for sensors, but you can optionally change it for your own version in order to change interpolation method (for instance MLS as described by Colagrossi and Landrini [3]). Provided version of OpenCL kernel can interpolate the value on usual SPH way (using Shepard correction as is described in section 6.4), or return maximum values detected.

Tag	Attribute	Values
FPS	value	Output frames per second
Script	file	OpenCL kernel file
Sensor	x	x coordinate
	y	y coordinate
	z	z coordinate
	type	Interpolated Maximum

Table 4.8: Sensors set options.

[1]Minimum suggested value is 0.01, but is recommended to get 0.03 at least

4.2.8 Movements

This group is used to set motions. Movements will be used to set fixed particles, DeLeffe area elements, and Ghost particles wall corners motion along the simulation time. You can add as many motions as you want adding several instances of the **Movement** tag, and final motion will be computed as the superposition of them.

Each added movement must have its own XML definition file. The data that must be provided in this file depends on the movement selected type. Valid types of movement are:

1. **LIQuaternion**: In this movement you must provide a data table file containing the solid quaternion at several time instants along the simulation. Quaternion is defined by center and axes, i.e. in 2D you must provide at least the center and the x axis, and in 3D the center and the x and y axes. Quaternion at each time step will be linearly interpolated from the provided data table.

2. **ScriptQuaternion**: This movement requires a Python script that will be executed each time step in order to receive the quaternion.

Quaternion based motions are usually the best choice because they provide full control of the motion, but has the disadvantage that will overwrite all the previous computed motions, so motions superposition is not supported.

Actually only Quaternion based motions are supported.

See examples chapter 8 in order to know how the motion definition file must be written for each type of movement.

Tag	Attribute	Values
Movement	type	LIQuaternion
		ScriptQuaternion
	file	XML definition file

Table 4.9: Movements set options.

4.2.9 GhostParticles

This group is used to set Ghost particles boundary condition. Ghost particles will be discussed on section 7.7.5.

In the table 4.10 all the valid options for Ghost particles are shown. Default values are 'SSM' (symmetric model) for the pressure and the tangent velocity, and 'ASM' (antisymmetric model) for the normal velocity.

TangentUModel can be selected in order to impose free-slip or no-slip. Free-slip condition is imposed with a 'SSM' model, and being consistent for a large number of cases, and no-slip is imposed with 'ASM' or with 'Takeda' (not implemented yet).

The consequences of selecting each mirroring model for each fields can be found in the reference [7].

In the table 4.11 the subgroups that can be used are listed. Ghost particles algorithm is quite different of Fix particles or boundary integrals, and require to set the walls separately, but not within the particles definition, so it's strongly recommended not using ghost particles with large number of solid boundary walls.

In 3D cases, each wall must have 3 or 4 (triangular or quadrangular walls respectively) ‘Vertex’ tags, with the x, y, z coordinates attributes, and in 2D simulations 2 ‘Vertex’ tags must be provided with the x, y coordinates attributes.

Tag	Attribute	Values
PressModel	value	ASM
		SSM
		Takeda
NormalUModel	value	ASM
		SSM
		Takeda
		U0M
TangentUModel	value	ASM
		SSM
		Takeda
		U0M

Table 4.10: Ghost particles options.

Subgroup	Description
Wall	Wall to mirror the fluid particles using set models.

Table 4.11: Ghost particles options.

4.3 Particles distribution file

In the section 4.2.6 fluids XML definition files have been described, introducing that the particles distribution must be provided in a external file. Particles distribution input files can have 3 formats:

1. ASCII
2. GiD
3. XML

In this chapter each format will be described in order to allow users to know the best way to introduce particles in SPH.

4.3.1 ASCII formatted file

It is probably the best option in the most cases, and has been the option selected for the examples presented in the chapter 8. Is a plain text file where all particles are defined each one in a file line.

Comments can be included in the file with the symbol ’#’. All the line contents after the ’#’ symbol will be ignored.

AQUAgpusph expects that following fields per particle are present on the provided files:

1. x (Mandatory): X coordinate [m].
2. y (Mandatory): Y coordinate [m].

3. z (Mandatory, only for 3D): Z coordinate [m].
4. n_x (Mandatory): X normal component. Fluid particles can have null normal.
5. n_y (Mandatory): Y normal component. Fluid particles can have null normal.
6. n_z (Mandatory, only for 3D): Z normal component. Fluid particles can have null normal.
7. v_x (Mandatory): X velocity component [m/s].
8. v_y (Mandatory): Y velocity component [m/s].
9. v_z (Mandatory, only for 3D): Z velocity component [m/s].
10. m (Mandatory): Mass if it is a fluid or fix particle [kg], area if it is a wall element [m^2].
11. *imove* (Optional): Moving flag. $imove > 0$ for all fluid particles, $imove < 0$ for fixed particles or vertexes ($imove = 0$ is reserved for sensors).
12. ρ (Optional): Density [kg/m^3].
13. c_S (Optional): Sound speed [m/s]. Sound speed must be selected, at least, 10 times greater than maximum expected fluid velocity.
14. h (Optional): Kernel characteristic height [m].

Field values must be set sorted, and can be separated by comma, semicolon, parenthesis, spaces or tabulator symbols. If several separators are concatenated between the field values will be joined as a space separator.

If one mandatory field is missing AQUAgpusph will report an error on the initialization stage. Regarding optional fields, it's strongly recommended set *imove* flag and density, that will be set as $imove = 1$ and $\rho = \rho_0$ by default. Be mindful that the state equation relates pressure and density fields, so if you want to perform a simulation with some initial pressure field you need to setup the density field according to the EOS 2.4.

Mass fields is a bit particular variable since if the particle is a boundary area element, this field must be set as the area assigned to it.

Take care also with the 2D simulations since the field values are described per meter of depth (3D missing direction). For instance area of the wall elements will be the line length [m]. You can see the provided examples in order to know more about this.

When a file with an unknown extension is provided in order to load the particles, it will be treated as ASCII file type.

4.3.2 GiD formatted file

AQUAgpusph accepts particles distribution in GiD mesh format. Mesh points will be used as the particles positions, with the provided fields. GiD format must provide following fields:

1. x (Mandatory): X coordinate [m].
2. y (Mandatory): Y coordinate [m].
3. z (Mandatory, only for 3D): Z coordinate [m].

4. w (Mandatory, only for 3D): Usually $w = 1$[m].
5. n_x (Mandatory): X normal component. Fluid particles can have null normal.
6. n_y (Mandatory): Y normal component. Fluid particles can have null normal.
7. n_z (Mandatory, only for 3D): Z normal component. Fluid particles can have null normal.
8. n_w (Mandatory, only for 3D): Usually $n_w = 0$.
9. v_x (Mandatory): X velocity component [m/s].
10. v_y (Mandatory): Y velocity component [m/s].
11. v_z (Mandatory, only for 3D): Z velocity component [m/s].
12. v_z (Mandatory, only for 3D): Usually $v_w = 0$[m/s].
13. m (Mandatory): Mass if it is a fluid or fix particle [kg], area if it is a wall element [m^2].
14. *imove* (Optional): Moving flag. $imove > 0$ for all fluid particles, $imove < 0$ for fixed particles or vertexes ($imove = 0$ is reserved for sensors).
15. ρ (Optional): Density [kg/m^3].
16. c_S (Optional): Sound speed [m/s]. Sound speed must be selected, at least, 10 times greater than maximum expected fluid velocity.
17. h (Optional): Kernel characteristic height [m].

Fields are really similar to the introduced at 4.3.1 section about the ASCII formatted input.

4.3.3 XML formatted file

This is an auxiliar method that is only recommended for a really low number of particles. In this method an XML file is requested, similar to the explained in chapter 4, but expecting **Particle** group instances (one per particle).

In table 4.12 a list with the valid tags and their attributes for each particle are related. The configurable fields are similar to the introduced at 4.3.1 section about the ASCII formatted input; revisit this section in order to know the possibilities of initial configuration file.

Tag	Attributes	Description
Position	x	(Mandatory) X coordinate [m].
	y	(Mandatory) Y coordinate [m].
	z	(Mandatory, only for 3D) Z coordinate [m].
Normal	x	(Mandatory) X normal component.
	y	(Mandatory) Y normal component.
	z	(Mandatory, only for 3D) Z normal component.
Velocity	x	(Mandatory) X velocity component [m/s].
	y	(Mandatory) Y velocity component [m/s].
	z	(Mandatory, only for 3D) Z velocity component [m/s].
Mass	value	(Mandatory) Mass for fluid or fix particles [kg], area for vertexes [m^2].
Imove	value	(Optional) Moving flag.
Density	value	(Optional) Density [kg/m^3].
Cs	value	(Optional) Sound speed [m/s].
KernelH	value	(Optional) Kernel characteristic height [m].

Table 4.12: Particles valid fields tags.

Chapter 5

Running AQUAgpusph

5.1 Launching simulation

In order to launch AQUAgpusph simulation you need to execute the program specifying the XML definition file in a command line. If you have installed AQUAgpusph in a folder present on the PATH environment variable you can launch a 3D simulation typing:

```
AQUAgpusph -i PathToXML
```

or

```
AQUAgpusph2D -i PathToXML
```

for 2D simulations.

AQUAgpusph accepts several command line options. You can get a complete list and description of them in the help page (‘-h/–help’ help option). If the ‘-i/–input’ option is not provided, Input.xml file will be selected as the XML case definition file, so must exist in the execution folder. You can consider use ’-n/–no-reassembly’ since output files reassembly can be a really time and hard-disk consuming operation, and simply unuseful if the simulation has been ran without interruptions.

You can start a simulation from previous output file. If you plan to perform the simulation in several runs, consider not perform the reassembly before the last execution. Also you will need to set options according (see section 4.2.2 to learn more about this).

AQUAgpusph execution can be stopped using ‘c’ key. When ‘c’ key event is detected simulation will be stopped at the end of the active time step, and output saved and closed correctly.

Eventually you can stop the simulation pressing ‘Ctrl’+‘c’, but then the simulation will stop immediately, and can result in output unreadable files[1].

5.2 Tracking simulation

5.2.1 General

AQUAgpusph prints the log information in 2 different ways, on the screen and in a log HTML file. Both of them have similar content, and can be controlled independently.

[1]H5Part output will be surely broken

5.2.2 Screen log information

The screen real time output can be controlled in the settings configuration section, see chapter 4.2.2 for details.

The screen output will report some data organized in groups:

- Simulation progress
  - Simulation time [s].
  - Simulation time step [s].
  - Percentage of the simulation accomplished.
  - Last output frame printed.
  - Estimated Time to Arrive [s].
- Profile data (Only Debug version)
- Calculation server memory
  - Number of particles.
  - Number of cells.
  - Allocated memory [bytes].
- Energy data
  - Momentums [$N \cdot s$]
  - Angular momentums [$N \cdot m\ s$]
  - Kinetic energy [J]
  - Potential energy [J]
  - Total energy [J]
- Events registry

Depending on your settings the “Energy data” group will be printed more often, or eventually not printed. Actually energy data is not computed on GPU, which implies a significant time consuming process and it’s strongly recommended to decrease print rate as much as possible.

In the other hand, when AQUAgpusph has computed the energy data in a certain time step, it does not recompute it until new step is called, so you can consider to set that all energy data outputs to be performed on the same instant (see the sections 5.2.3 and 6.2 to learn more about this).

AQUAgpusph also shows in real time an events registry, where some relevant messages are shown. Messages are classified in 4 categories[2]:

1. **Terminal output**: Initialization and closing information.
2. **White**: Relevant information.
3. **Orange**: Warning messages.
4. **Red**: Error messages.

In section 5.2.4 you can find a list of the messages that you can expect from the simulation, and the possible causes of them.

[2]Colors are only actives if AQUAgpusph has been built with NCurses support, see chapter 3 for details.

5.2.3 HTML log file

Depending on the settings discussed on the section 4.2.4, AQUAgpusph can be configured to print a log HTML file "Log.html"[3]. The log file contains useful information to to track it remotely and in order to know and preserve all incidents and remarks associated with the simulation.

HTML log file contains information classified in 3 categories:

1. **Black**: Relevant information.
2. **Orange**: Warning messages.
3. **Red**: Error messages.

In section 5.2.4 you can find a list with some relevant messages that you can get on the log HTML file. Also Information about date and time, and printed output files will be included on this file.

5.2.4 Messages glossary

AQUAgpusph reports several messages on real time. In this section we show all the messages that can be shown, with description, source, and possible solution if proceed.

The messages have the following structure:

```
[TYPE (T=TIMEs, Step=STEP)] (METHOD): MESSAGE
```

TYPE is the type of message, that can be INFO, WARNING, or ERROR; *TIME* is the simulation time where the event has been registered; *STEP* is the simulation time step where the event has been registered; *METHOD* is the AQUAgpusph routine that registered the event; *MESSAGE* is the description of the event.

The list of events that you can obtain during the AQUAgpusph execution are:

[ERROR] VTK output called, but not supported:

If AQUAgpusph has been built without VTK support, but VTK files output have been requested, this error will be reported each time that output is called. See chapter 3.3.2 in order to know how to activate support for VTK files.

[ERROR] H5Part output called, but not supported:

If AQUAgpusph has been built without H5Part support, but H5Part files output have been requested, this error will be reported each time that output is called. See chapter 3.3.2 in order to know how to activate support for H5Part files.

[ERROR] Can't send variable to kernel:

Error produced when AQUAgpusph can't send a variable to an OpenCL kernel. If you modified an OpenCL kernel and/or the source code to customize AQUAgpusph functionality, ensure that sent variables from AQUAgpusph and declared ones on OpenCL kernel matchs.

This error will stop AQUAgpusph execution.

[3]If prefix is not set as command line option. You can set a prefix that will be inserted at the start of all the output file names

[ERROR] Can't execute the kernel:

OpenCL kernel can't be launched. If the problem is specifically documented, it will be added to the log too. Common causes of this problem is a custom OpenCL kernel, or unallocatable resources.

This error will stop the AQUAgpusph execution.

[ERROR] Can't wait to kernels end:

If AQUAgpusph has been built in Debug mode, an execution profile will be activated in order to report time consumed by each stage of the code. This error is reported when program can't wait for kernel execution, usually caused by a kernel execution error.

This error will stop the AQUAgpusph execution.

[ERROR] Can't profile kernel execution:

Similar to previous one, in this case program can wait to kernel finish, but can not be profiled (execution time can not be extracted).

This error will stop the AQUAgpusph execution.

[ERROR] Resultant keys overflows unsigned int type:

Radix sort stage related error. This error is usually caused by a too large number of cells, that can happen if a domain is not bounded (see section 4.2.5 in order t know how to set it) and one or more particles go out of physical boundaries.

This error will stop the AQUAgpusph execution.

[ERROR] perform() execution fail:

Error associated to simulations with Python script motion control. The error is caused by a Python runtime execution error detected at perform() required method. Motion control Python script must be fixed.

Check chapter 4.2.8 in order to learn more about how to set a Python script controlled motion, and 7.8.3 to learn more about the Python scripting.

This error will stop the AQUAgpusph execution.

[ERROR] perform() returned quaternion is not valid:

Error associated to simulations with Python script motion control. The error is caused by a wrong returned value by Python script perform() method. Motion control Python script must be fixed.

You can see chapter 4.2.8 in order to learn more about how to set a Python script controlled motion, and 7.8.3 to learn more about the Python scripting.

This error will stop the AQUAgpusph execution.

[ERROR] Failure retrieving memory from server:

OpenCL memory can not be recovered on host. AQUAgpusph eventually provide details of this error into the log.

This error will stop the AQUAgpusph execution.

[ERROR] Failure sending memory to server:

Data can not be sent from host to OpenCL platform. AQUAgpusph eventually provide details of this error into the log.

This error will stop the AQUAgpusph execution.

[ERROR] Fail allocating memory for *array* (*m* bytes):

Produced by an unacceptable number of cells number. Requested memory on OpenCL platform can't be allocated.

This error will stop the AQUAgpusph execution.

[ERROR] timestep has dramaticaly decreased! [dt_t -> dt_{t+dt}]:

In AQUAgpusph a variable time step can be set, see chapter 4.2.4 to learn more about the time step alternatives. Variable time step is a good way to walk around some transitional instabilities, decreasing instantaneously the time step, and trying to avoid that the particles pass trough walls therefore.

This error is reported when the time step is reduced in one or more orders of magnitude. If this error is reported the simulation can continue, but long computational times with wrong results can be expected, and you may consider the possibility of stop and fix simulation in order to relaunch it.

[WARNING] timestep changed [dt_t -> dt_{t+dt}]:

In AQUAgpusph a variable time step can be set, see chapter 4.2.4 to learn more about the time step alternatives. Variable time step is a good way to walk around some transitional instabilities, decreasing instantaneously the time step, and trying to avoid that the particles pass trough walls therefore.

This message indicates that the time step has been adapted. The time step variation is an indicative that the simulation is not running right, so if you receive too much time step change reports, or the time step change is large, maybe you must consider repeat the simulation increasing the sound speed or the time step divisor.

[WARNING] Number of cells increased [n_t -> n_{t+dt}]:

Like other SPH codes, AQUAgpusph neighbour particles localization is based on a PIC algorithm, where each particle is marked by a grid cell where is situated. The number of cells where the particles can be located depends on the extension of fluid domain. Therefore, it can be modified along the simulation. You can learn more about cells usage on AQUAgpusph at chapter 7.5.

In most simulations fluid bounds are not constant along the simulation, so some reports about cell numbers increase can occur. If this increase is too large the reallocation of memory can be impossible, breaking the simulation, and possibly indicating that some particles has abandoned the physical domain.

[**INFO**] **List too small, local memory usage will be avoided:**

When the number of particles is too small, local memory can't be used in the transpose stage of the Radix sort process (See the chapter 7). AQUAgpusph will run more slowly therefore, but since the number of particles is not large probably you must can ignore this notification.

Sometimes you can try to increase sightly the number of particles in order to recover the possibility of using local memory, increasing the resolution and the performance.

[**INFO**] **Interrumption detected:**

Notice that 'c' key has been pressed, and the simulation will stop.

[**INFO**] **Allocated memory =** m **bytes:**

Reports allocated memory on OpenCL platform.

Chapter 6

PostProcessing simulations

6.1 General

AQUAgpusph output files can be organized in 4 groups:

1. Log.
2. Data files.
3. Custom data files.
4. Visualization files.

Log files are commented on chapter 5.2.

Data files output are selected by the user. In chapters 6.2 to 6.4 the content of data files are discussed.

If you select a Python scripted motion, you can perform custom output data files. In chapter 7.8.3 you can learn more about the motion controlled by a Python script.

Ending files designed to visualization processes can be printed too, as explained into 4.2.4 section. In this user manual, the visualization using paraview will be explained.

6.2 Energy data file

Energy file is a standard output file of AQUAgpusph program, you can set printing rate of this file as explained on 4.2.4 section. Energy data file is called “Energy.dat”[1].

Energy data file printed by AQUAgpusph has a header that explains the fields printed into it. Following columns are printed on energy data file:

1. **Time**: Print time [s].
2. **X_Mom**: Momentum at X direction [N · s].
3. **Y_Mom**: Momentum at Y direction [N · s].
4. **Z_Mom** (Only for 3D): Momentum at Z direction [N · s].

[1]If a prefix is set on command line arguments, will inserted on the start of the file name

5. **X_AngMom** (Only for 3D): Angular momentum respect X axis [N · m s].
6. **Y_AngMom** (Only for 3D): Angular momentum respect Y axis [N · m s].
7. **Z_AngMom**: Angular momentum respect Z axis [N · m s].
8. **Kinetic_Energy**: Kinetic energy [J].
9. **Potential_Energy**: Potential energy [J].
10. **Total_Energy**: Total energy [J].

Each line represents a time step of the simulation, characterised by simulation time instant. Energy data (including momentums and angular momentums) only considers fluid particles, but not vertexes, fixed particles or sensors.

This data output file can easily plotted with gnuplot for instance.

```
#########################################################
#                                                       #
#    #    ##   #  #   #                           #     #
#   # #  #  #  #  #  # #                          #     #
#  ##### #  #  #  # #####  ##  ###  #  #  ## ###  ###   #
#  #   # #  #  #  # #   # #  # #  # #  # #   #  # #  #  #
#  #   # #  #  #  # #   # #  # #  # #  #   # #  # #  #  #
#  #   #  ## #  ##  #   #  ### ###   ### ##  ###  #  #  #
#                            # #             #          #
#                          ##  #             #          #
#                                                       #
#########################################################
#
# Another QUAlity GPU-SPH, by CEHINAV.
# http://canal.etsin.upm.es/
# Authors:
# Jose Luis Cercos-Pita
# Leo Miguel Gonzalez
# Antonio Souto-Iglesias
#
#########################################################
#
# GnuPlot script to plot energy global variables in real time.
#
#########################################################
#
set multiplot
set key
set grid
# Line styles
set style line 1 lt 1 lw 2
set style line 2 lt 2 lw 2
set style line 3 lt -1 lw 2
# Plot Kinetic & gravity energy
```

```
set xlabel "Time [s]"
set xtic
set ylabel "Kinetic Energy [J]"
set ytic
set y2label "Gravity Energy [J]"
set y2tic
set size square 0.5,1.0
set origin 0.0,0.0
plot "Energy.dat" using 1:5 title 'Kinetic energy' axes x1y1 with lines ls 1, \
     "" using 1:6 title 'Gravity energy' axes x1y2 with lines ls 2

# Plot global energy
set origin 0.5,0.0
set ylabel "Energy [J]"
unset y2label
unset y2tic
plot "Energy.dat" using 1:7 title 'Energy' axes x1y1 with lines ls 3
set nomultiplot

# Replot in 5 seconds
pause 5
replot
reread
```

Attached script can be used to plot kinetic, potential, and total energies during the simulation is running. The plot will be autoupdated (5 seconds refresh period), showing Kinetic energy on the left plot, refereed to left y axis, potential energy on the left plot as well, but refereed to right y axis, and total energy on the right plot.

Similar scripts can be easily make to plot momentums or angular momentums, or in order to create images (without showing and refreshing) as well.

6.3 Bounds data file

Bounds data file is so quite similar to the previous one. You can find how to set printing rate on section 4.2.4 as well. Bounds data file is called "Bounds.dat"[2].

This file contains following data fields:

1. **Time**: Print time [s].
2. **X_Min**: Minimum X coordinate [m].
3. **Y_Min**: Minimum Y coordinate [m].
4. **Z_Min** (Only for 3D): Minimum Z coordinate [m].
5. **X_Max**: Maximum X coordinate [m].
6. **Y_Max**: Maximum Y coordinate [m].
7. **Z_Max** (Only for 3D): Maximum Z coordinate [m].

[2]If a prefix is set on command line arguments, will inserted on the start of the file name

8. **V_Min**: Minimum velocity [m/s].

9. **V_Max**: Maximum velocity [m/s].

Like energy data, in bounds data only fluid particles are considered, discarding fixed particles, vertexes and sensors therefore.

On previous section you have a script that can be easily fit to plot bounds data.

6.4 Sensors data file

Sensors data is a little bit different. Sensors file will printed only if you add at least one sensor as explained on chapter 4.2.7, where you can set printing rate too. Sensors data file is called "Sensors.dat"[3].

Sensors file contains following fields (explained in a header):

1. **Time**: Print time [s].

2. **X** (One per sensor): X coordinate [m].

3. **Y** (One per sensor): Y coordinate [m].

4. **Z** (One per sensor, only for 3D): Z coordinate [m].

5. **press** (One per sensor): Pressure [Pa].

6. **density** (One per sensor): Density [kg/m^3].

7. **sumW** (One per sensor): Kernel completeness factor (formerly Shepard term).

This file has the singularity that 6 last described fields are repeated as many times as sensors have been added. Pressure and density written in file are ever renormalized using Shepard correction, if you want to know the traditional SPH interpolated value you must multiply the values by the Shepard term (last field of each sensor).

Sensors data file can be easily plotted with gnuplot as well.

6.5 Visualization files

6.5.1 General

AQUAgpusph actually offers 2 types of visualization output files, Tecplot and H5Part ones. Tecplot output files are outdated, and since is a plain text output format, is created and loaded slowly and hight hard disk consuming, so is strongly recommended to don't use this format. However H5Part has been spicifically designed to perform particles output in a binary form, being computationally efficient compared with other formats, and is the format that will be documented here. AQUAgpusph VTK format support is incoming, but not implemented yet.

[3]If a prefix is set on command line arguments, will inserted on the start of the file name

6.5.2 H5Part

H5Part is an high efficient implementation of hdf5 file template oriented to particles based output. Main advantages are:

1. Is a binary format that reduce significantly the Hard Disk resources consuming.
2. Is a free library.
3. All time steps can be packed on one file.

And main disadvantages are:

1. Is not generalized format, so is not commonly introduced on standard package repository of most usually used Linux distributions.
2. If program fails before file is right closed, will result on unrecoverable data.

On actual AQUAgpusph implementation this is the main output format. H5Part files can be postprocessed with ParaView[4]. This chapter is not oriented to describe the work with ParaView, please refer to ParaView's web page to find tutorials, in this chapter we will describe some specific operations that can helps you processing data.

In AQUAgpusph particles have 2 relevant flags, ***imove*** and ***ifluid***. The first one marks if the particle is:

1. $imove < 0$ a fix particle or a wall vertex.
2. $imove = 0$ a sensor.
3. $imove > 0$ a fluid particle.

So you can use **Clip** tool over this scalar in order to split particles in this 3 categories that can be visualized independently. Regarding ***ifluid*** flag, marks the fluid of each particle, allowing you extract different fluids in order to process it independently of the other ones, with the **Clip** filter as well.

All the fields exported by AQUAgpusph are scalars, including vectors components. You can build vectorial fields for some specific purposes with the **Calculator** tool, simply set the sum of each vector component multiplied by *iHat, jHat or kHat* associated variable.

You can get a large number of additional plugins specifically oriented to SPH visualization works from pv-meshless.

On actual AQUAgpusph package is not possible to select what fields will be printed, printing all relevant particles data. Future releases will offer the possibility of discard some fields of output in order to decrease the output file size.

[4]ParaView H5Part Reader plugin must be loaded

Chapter 7

AQUAgpusph in detail

7.1 General

AQUAgpusph functionality can be deeply analyzed using Doxygen documentation. In section 3 you can find how to build Doxygen developers documentation during AQUAgpusph compile process. When AQUAgpusph have been already built you can open Doxygen documentation web page with your internet browser, loading doc/Doxygen/html/index.html web page file.

The objective of this chapter is provide some key notes about how AQUAgpusph works in order to allow users understand well what can do or not with this software.

In figure 7.1 you can see the general flux diagram. AQUAgpusph is based on a host-server structure. Host is designed to prepare the case, and to receive and process results, since server is designed to receive data from server, and compute as many time steps as possible until no new output is needed.

This structure does not only simplify the process and allows more comprehensive code, also is designed for tolerate several calculation servers working simultaneously on future releases, trying to preserve the actual simplicity.

7.2 Arguments manager

Arguments manager is the piece of code that analyse input command line options/arguments. You can get a list of valid options and arguments executing:

```
AQUAgpusph --help
```

Valid options are:

- **-i, --input=INPUT**: Set the definition XML file of the simulation.
- **-o, --output-prefix=PREFIX**: Set a prefix for output files.
- **-n, --no-reassembly**: Visualization output files will not reassembled.
- **-v, --version**: Show AQUAgpusph version and stop the execution.
- **-h, --help**: Show AQUAgpusph helps page and stop the execution.

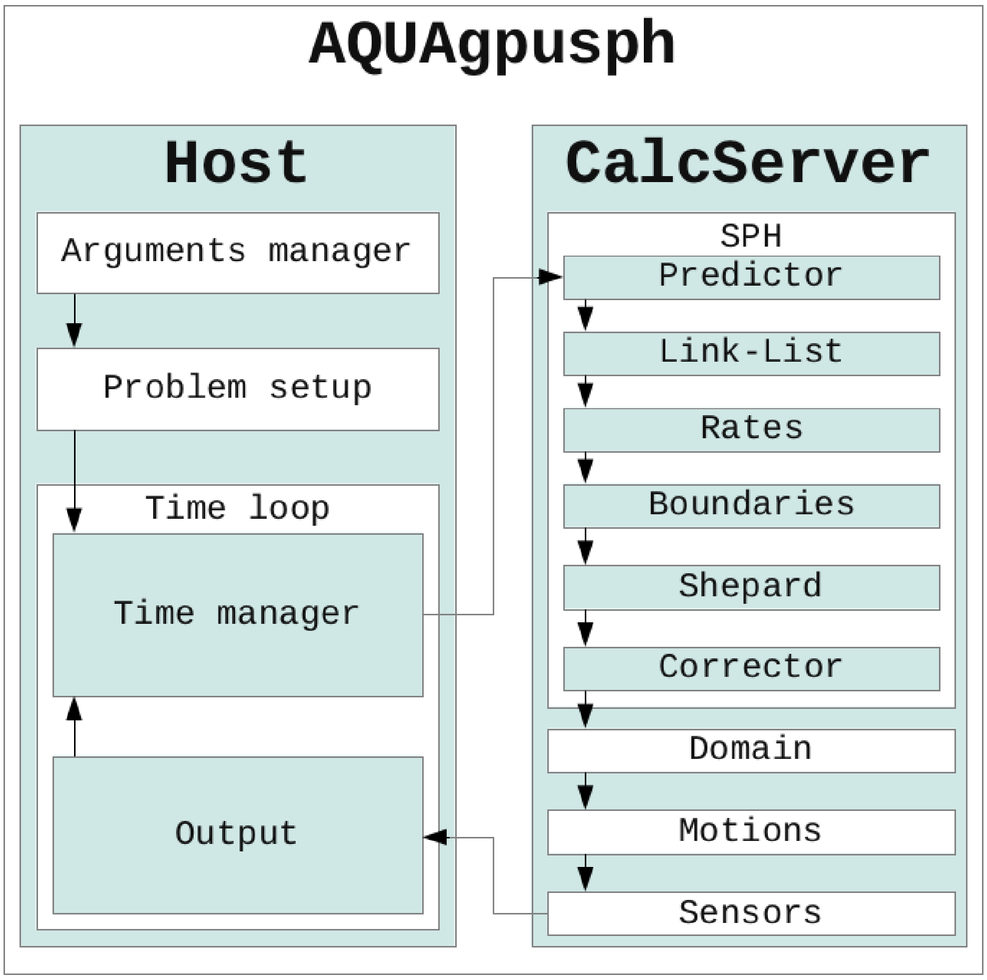


Figure 7.1: General AQUAgpusph flux diagram

Options can be freely used simultaneously until incompatibilities are not notified here, or at the help page. Some combinations of command line options can result on undesired behaviour, like for instance mixing --version with --help will discard one of them.

In almost cases --no-reassembly is a good option, due to reassembly output files can be a really hard-disk & CPU consuming process.

7.3 Problem setup

The problem setup stage basically consist on loads all settings established as explained on chapter 4, and load the particles/vertexes data from described files on chapter 4.3.

Problem setup will report work done, and the eventually incidences, on the initialization screen output. The screen output during initialization and close stages of AQUAgpusph execution will be preserved if NCurses support has been activated, but if NCurses support is not active you can lost the initialization process reported messages due to the large amount of these printed during the simulation. In this case you can redirect the screen output to a file typing:

```
AQUAgpusph COMMANDLINE_OPTIONS > screen.out
```

You can also detach the execution from terminal, for instance using:

```
nohup AQUAgpusph COMMANDLINE_OPTIONS > screen.out &
```

Allowing you to run AQUAgpusph remotely, but in this case you will lost the possibility of stop simulation using 'c' key[1], that as described on chapter 5.1 allows you to right stop the simulation closing files.

In both cases you will preserve a file called "screen.out" that saves the messages reported during initialization and close stages, and along the simulation as well.

7.4 Leap-Frog integration scheme

Weakly compressible SPH method is able to return the instantaneous acceleration and density rate for each particle in a Lagrangian point of view, so 3 variables must be integrated along the time, velocity, position and density of each particle. AQUAgpusph time integration is based on a Leap-Frog predictor-corrector scheme:

1. Predictor:

$$\dot{x}_{t+dt}^{\mathrm{pred}} = \dot{x}_t + dt\left(\ddot{x}_t + g\right)$$

$$x_{t+dt}^{\mathrm{pred}} = x_t + dt\dot{x}_t + \frac{dt^2}{2}\left(\ddot{x}_t + g\right)$$

$$\rho_{t+dt}^{\mathrm{pred}} = \rho_t + dt\dot{\rho}_t$$

2. SPH interactions:

$$\ddot{x}_{t+dt} \leftarrow \mathrm{SPH}$$

$$\dot{\rho}_{t+dt} \leftarrow \mathrm{SPH}$$

3. Corrector:

$$\dot{x}_{t+dt} = \dot{x}_{t+dt}^{\mathrm{pred}} + \frac{dt}{2}\left(\ddot{x}_{t+dt} - \ddot{x}_t\right)$$

$$x_{t+dt} = x_{t+dt}^{\mathrm{pred}}$$

$$\rho_{t+dt} = \rho_{t+dt}^{\mathrm{pred}} + \frac{dt}{2}\left(\dot{\rho}_{t+dt} - \dot{\rho}_t\right)$$

7.5 Link-List

SPH codes commonly uses a PIC (Particle In Cell) based algorithm to easily locate particle's neighbours, called Link-List. The main difference with typical PIC algorithm is that on SPH a chain is built from the cells data, linking each particle with the next one on the same cell, allowing that cell only saves the index of the first particle of the chain.

In figure 7.2 a simplified scheme of Link-List stored data is shown. Particles are located in cells that have a length of the maximum interaction distance, then each cell allocates a particle index called "head of chain" (*ihoc*). Each particle stores the next one of the cell chain (*ll*) until last particle is reached, where some invalid index is set (-1 for instance).

In GPU oriented code this behaviour can be conveniently modified looking for sorted memory address reading, that is really more efficient, sorting particles by cell index. The problem is that traditional method implies storing Link-List chain, and more over, implies a chaotic memory access at particles interaction stage, with a heavy performance penalty. So, as many other GPU based SPH codes, AQUAgpusph sorts the particles,

[1]Advanced Linux users can get the process PID and then reinject keys to program

transforming the Link-List array ll[i] in i+1 variables, so is not anymore needed to store it therefore.

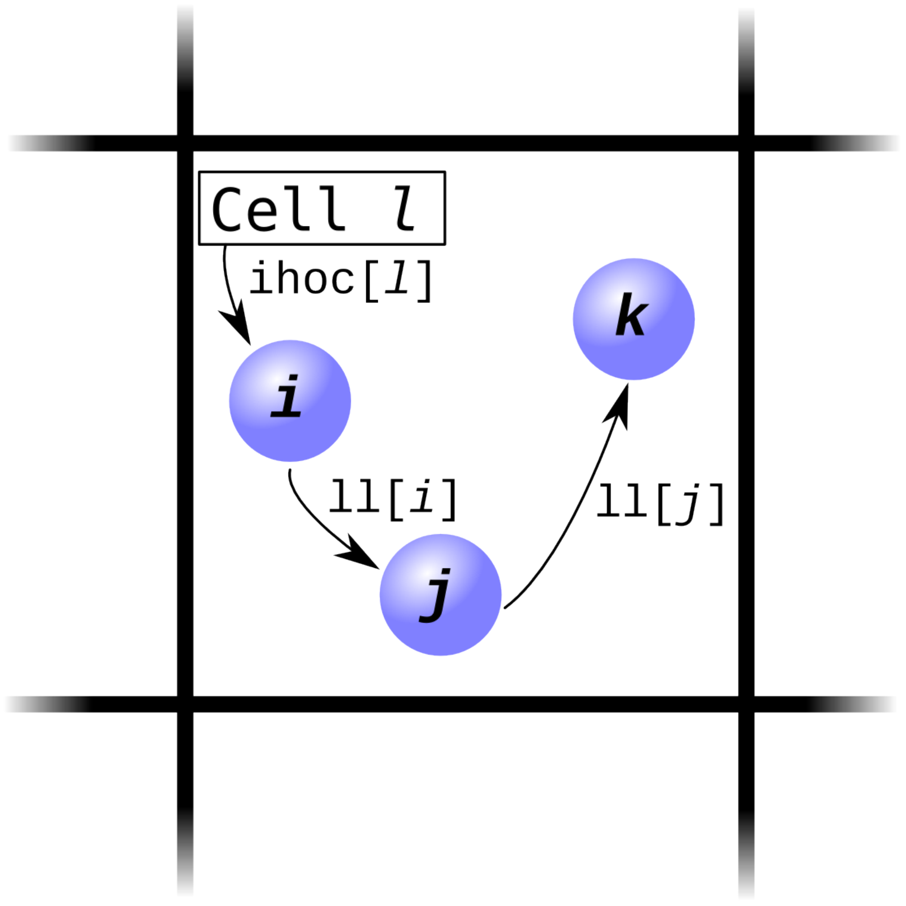


Figure 7.2: Conventional Link-List scheme

7.6 Particles interactions

Particles interactions is the core of every SPH method, and is the most complex stage too, being the most time consuming one. AQUAgpusph can perform several particles interaction stages by time step depending on simulation settings, for instance, if 'DeLeffe' boundary condition is set at least 3 interactions stages will be performed, the usual one, the boundary integrals, and 'ElasticBounce' non wall trespassing condition, but since only the usual one needs to compute all possible particles interactions, the others may take less time.

As explained in section 7.5, the particles interactions computation performance is heavily dependant on Link-List stage and AQUAgpusph performs a particles sorting based on cell as many other GPU oriented SPH codes, but two main differences must be remarked:

7.6.1 Unsorted output arguments

In section 7.4 time integration scheme is described, and we can find that after predictor stage, variables $\dot{x}_t$, $\mathbf{x}_t$, and ρ_t are not needed anymore until new time step, so AQUAgpusph can use these arrays in order to store sorted data for SPH interactions stage, decreasing memory requirements, but the same process can not be applied to output data $\ddot{x}_t$ and $\dot{\rho}_t$, because is needed on corrector stage, so there are 2 ways to solve it:

1. Create new arrays for the sorted output data, working all time in sorted space.
2. Work on sorted space, but write at unsorted space.

AQUAgpusph use the second way, that have some significant advantages:

1. Less memory requirements that the first way.
2. Since AQUAgpusph is using non needed arrays to store the sorted input data, the unsorted data is preserved.
3. Since the output data is written on unsorted space directly, after SPH interactions stage all the unsorted data is available to continue working, including to print output files (Note that the unsorted space is the original one).
4. No unsort operation is required.

Of course, write into the unsorted space at the interactions stage can be more expensive, but since the writing operations are really less frequent than the reading ones, performance decrease can be assumed.

7.6.2 Coalescence read

Newest hardware not suffers really heavy penalties when a lot of threads try to read from the same memory address, nevertheless AQUAgpusph incorporates an algorithm in order to reduce as much as possible the number of readers over unique address. In figure 7.3 a scheme about the interactions process is shown. Interactions stage is divided in 3 substages:

1. Interacts with next particles of the home cell[2] chain.
2. Interacts with the particles on the neighbour cells.
3. Interacts with previous particles of the home cell chain.

As you can see in figure 7.3 the particles does not reads from the same memory address in any interactions. In practice, due to the particles interactions can take different times, due to the neighbour cells particles can try to read from same memory address, several simultaneous reads can happens, but in much less number.

7.7 Boundary conditions

7.7.1 General

AQUAgpusph provides most commonly used boundary conditions in order to allow users to can choose the one than better fit to their problem. The generally suggested method is Boundary integrals (formerly 'DeLeffe') that is flexible and powerful, but have the disadvantage that requires 'Shepard' correction for consistency.

Along this chapter all boundary conditions will be described, and advantages discussed. Also the mixing possibilities will be introduced.

[2]Home cell is the cell where the computing particle is located

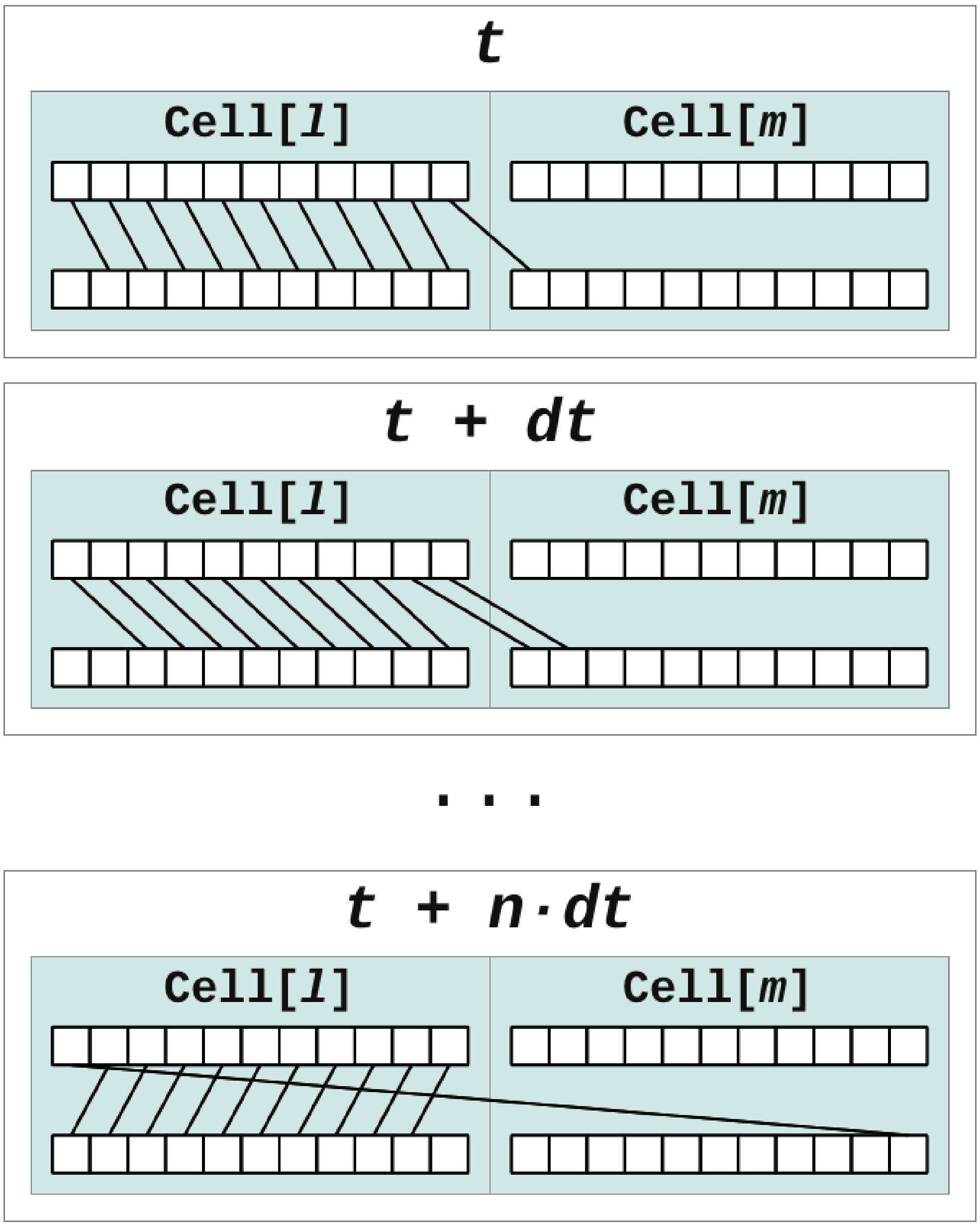


Figure 7.3: Coalescence reading flow

7.7.2 Elastic bounce

This is the simplest way to impose the boundary conditions, based on particle elastic bound within the wall, where a elastic factor is applied in order to set plasticity.
In figure 7.4 a scheme of the elastic bound effect is shown. In AQUAgpusph walls are defined by a set of vertexes (black balls), that have a normal[3] and area associated (area is stored on mass array). In order to AQUAgpusph identify particles as vertexes, *imove* flag must be set lower than 0.
Elastic bounce is applied when a particle will moved nearer to a vertex than ‘BoundDist’ parameter introduced on chapter 4.2.5, then the normal component of velocity is modified in order to get an elastic bound multiplied by elastic factor ‘BoundElasticFactor’. If the elastic factor is 1, a fully elastic bound will performed, however if elastic factor is 0, particle will stopped (normal component of the velocity will affected only), so with values between 0 and 1 a semi-elastic bound will performed. You can set elastic factors out of [0,1] bounds if you know what are you doing.
Main advantage of this boundary is the simplicity that reverts on a good performance, but using the Elastic

[3]Ensure yourself that the provided normals are normalized, AQUAgpusph will accept it as set by user

bounce boundary condition alone will return poor physics results, The Elastic bounce boundary condition is then aimed to work with other boundary conditions, serving a second barrier for particles that can try to trespass the walls.

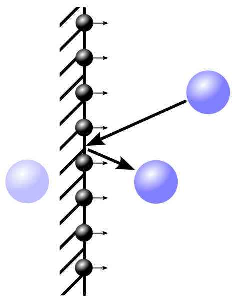


Figure 7.4: Elastic bounce boundary scheme

7.7.3 Fix particles

'Fix particles', also called 'dummy particles' sometimes, is a extension of the fluid domain along walls with a little bit especial fluid particles linked to them.
In figure 7.5 an scheme of the fixed particles distribution is shown. Must be noticed that first row of fixed particles is placed $dr/2$ inner on wall, and the several rows are placed every dr. The fixed particles are similar to the fluid ones, but a normal must be provided and *imove* flag must be lower than 0.
The fixed particles will use continuity equation like the fluid particles, but not momentum equation, because fixed particles velocity is linked to the wall motion. Related to this, if no motions are set, the initial fixed particles velocity will be preserved along the simulation[4].
SPH equations when 'Fix particles' boundary condition is used are therefore

$$\left\langle \frac{\nabla p}{\rho} \right\rangle_a \Bigg|_{a \in \text{Fluid}} = \frac{1}{\gamma_a} \sum_{b \in \text{Fluid} \cup \text{Boundary}} \left(\frac{p_a}{\rho_a^2} + \frac{p_b}{\rho_b^2} \right) \nabla W_{ab} m_b$$

$$\langle \rho \ \text{div}(v) \rangle_a \Big|_{a \in \text{Fluid} \cup \text{Boundary}} = -\frac{1}{\gamma_a} \sum_{b \in \text{Fluid} \cup \text{Boundary}} (v_a - v_b) \nabla W_{ab} m_b$$

$$v_{a \in \text{Boundary}} = \text{f}\left(x_{\text{walls}}(t)\right)$$

Where γ_a can be forced to be 1 ever setting Shepard correction to "None" as decribed on the chapter 4.2.5, that have the advantage of getting a fully conservative notation.
'Fix particles' uses internally 'ElasticBounce boundary' condition described on section 7.7.2. It's strongly recommended let 'Fix particles' work with 'ElasticBounce boundary' in order to warranty that the particles can trespass the walls, but if you want to disable this feature simply set 'BoundDist' parameter to 0, **never pass null normals to the fixed particles**.
'Fix particles' is chronologically the first boundary condition implemented on SPH, has the main advantage that is easy to implement since the fixed particles have a really similar treatment to the usual fluid particles, but have the main disadvantage of the particles velocity set using only walls motion as a 'U0M', that gives an inconsistent Laplacian operator. Also requires increase the number of particles significantly, more as the number of neighbours grows.

[4]you can modify this behaviour with custom predictor and corrector kernels

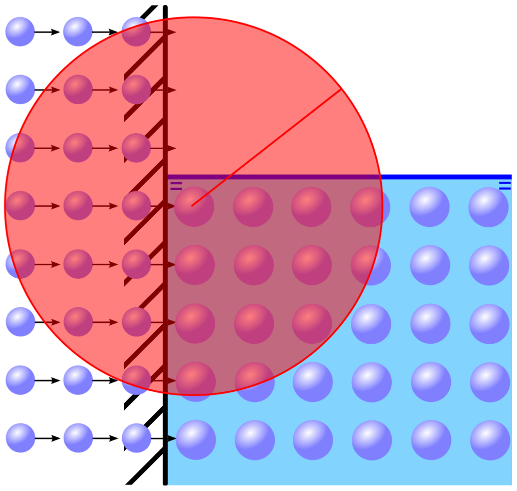


Figure 7.5: Fix particles scheme

7.7.4 Boundary integrals, formerly 'DeLeffe'

Boundary integrals is the newest method to compute boundary conditions, and the implementation method is similar to 'ElasticBounce boundary' described at section 7.7.2.
In figure 7.6 a boundary integrals scheme is shown. In original DeLeffe's boundaries integral development, oriented to 2D applications, an analytic solution of the integral, with a discretized version of fields, is purposed for full integration line (green highlighted on the image). In AQUAgpusph each vertex is associated to a wall area, then not only the field is discretized but also the integral as well, with a little bit worst result, but allowing to use this method in 3D simulations too.
In order to can converge the discretized boundary integral to the analytical one, you can set more vertexes with smaller areas, but be mindful that the values in vertexes along the walls are interpolated from the fluid particles, so really high resolution in walls, compared to fluid resolution, don't make any sense due to bad interpolated fields.
Boundary integrals have advantages in terms of consistency, with consistency of order $O(h)$ for all the operators, including the Laplacian. At the other hand this boundary condition requires Shepard correction, that can be a little bit unstable, and breaks the fully conservative SPH notation. The equations are therefore

$$\left.\left\langle \frac{\nabla p}{\rho} \right\rangle_a\right|_{a\in\text{Fluid}} = \frac{1}{\gamma_a}\left(\sum_{b\in\text{Fluid}} \left(\frac{p_a}{\rho_a^2} + \frac{p_b}{\rho_b^2}\right) \nabla W_{ab} m_b - \sum_{b\in\text{Boundary}} \rho_b \left(\frac{p_a}{\rho_a^2} + \frac{p_b}{\rho_b^2}\right) n_b S_b \right)$$

$$\left.\langle \rho \ \text{div}(v) \rangle_a\right|_{a\in\text{Fluid}} = -\frac{1}{\gamma_a}\left(\sum_{b\in\text{Fluid}} (v_a - v_b) \nabla W_{ab} m_b - \sum_{b\in\text{Boundary}} \rho_b (v_a - v_b) n_b S_b \right)$$

$$p_{a\in\text{Boundary}} = \sum_{b\in\text{Fluid}} \left(\frac{p_b}{\rho_b} + g \cdot r_{ab}\right) W_{ab} m_b$$

$$v_{a\in\text{Boundary}} = \text{f}\left(x_{\text{walls}}(t)\right)$$

Since the integration domain is truncated (this method is not based on a fluid extension), Shepard correction usage is mandatory. In AQUAgpusph Shepard correction over the pressure term will be applied by default, but not over the continuity equation, because compressibility is small in all cases, and Shepard correction can unstabilize this equation. Nevertheless you can activate the Shepard normalization as described in the chapter 4.2.5.
Must be noticed that the vertexes must be placed on the center of their representative area, taking special care on the limits of walls. In figure 7.7 the distribution of vertexes, that is separated $dr/2$ (half of distance between particles dr), in a corner is shown. Since the area (line length because is a 2D example) associated to each

vertex is $dr/2$, the distance of first vertex to corner is $dr/4$.
'DeLeffe' uses internally 'ElasticBounce boundary' condition described on section 7.7.2. It's strongly recommended let 'DeLeffe' work with 'ElasticBounce boundary' in order to warranty that the particles can trespass the walls, but if you want to disable this feature simply set 'BoundDist' parameter to 0, **never pass null normals to the vertexes**.

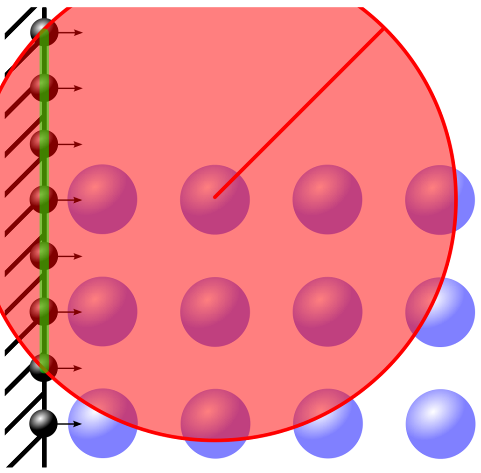


Figure 7.6: Boundary integrals scheme

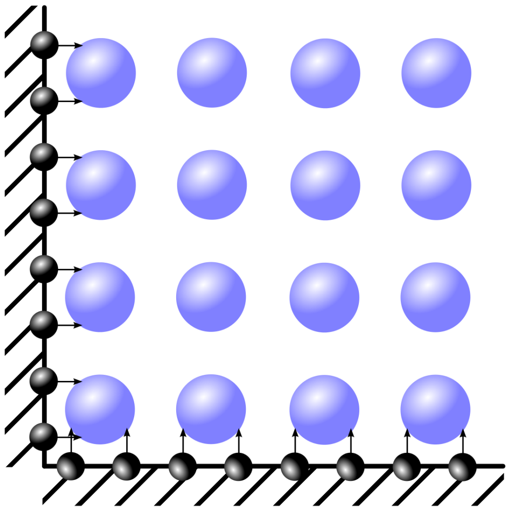


Figure 7.7: Corner particles and vertexes distribution

7.7.5 Ghost particles

'Ghost particles' is an algorithm so quite similar to 'Fix particles' one, but in this case the extension of fluid is not composed by particles linked to wall, that integrates their density using continuity equation, but is composed by a mirroring of the fluid respect to the wall.
In the figure 7.8 a scheme about the fluid particles mirroring is shown.
So the SPH equation are therefore

$$\left\langle \frac{\nabla p}{\rho} \right\rangle_a \Bigg|_{a \in \text{Fluid}} = \frac{1}{\gamma_a} \sum_{b \in \text{Fluid} \cup \text{Boundary}} \left(\frac{p_a}{\rho_a^2} + \frac{p_b}{\rho_b^2} \right) \nabla W_{ab} m_b$$

$$\langle \rho \; \mathrm{div}(v) \rangle_a \Big|_{a \in \mathrm{Fluid}} = -\frac{1}{\gamma_a} \sum_{b \in \mathrm{Fluid} \, \cup \, \mathrm{Boundary}} (v_a - v_b) \, \nabla W_{ab} m_b$$

$$p_{a \in \mathrm{Boundary}} = \mathrm{f}\left(x_a, x_{\mathrm{walls}}\left(t\right)\right)$$

$$v_{a \in \mathrm{Boundary}} = \mathrm{f}\left(x_a, x_{\mathrm{walls}}\left(t\right)\right)$$

The usual method to implement this boundary condition is perform a mirroring stage before the particles interaction, storing the mirrored particles data in order to can use them within the fluid particles integration. In GPU oriented code[5] this approach has the problem of the previously unknown number of ghost particles, so memory reallocation or over-dimensioned arrays are mandatory therefore. This can be walked around creating fix particles, but getting their values from fluid by a mirroring operation and interpolation, with the advantage that the number of particles is ever know and reallocation is not needed therefore, but with the disadvantage that requires to generate the fixed particles.
In AQUAgpusph a new algorithm is drafted, where the ghost particles are not precomputed, called 'Virtual ghost particles'. With this method a particles interaction process is called for each wall defined, so this method is not recommended to high number of walls, nevertheless, when the particles interaction is called respect to a wall, each particle will first test that is enough near to the wall before start working, so each 'Ghost particles' interaction called is significantly cheaper than usual one. The working method is locate all neighbours of the particle as is done on the usual particles interaction method, and for each neighbour located compute the mirrored particle on runtime, avoiding the need to store this data.
Since the number of walls must be relatively small, define the walls using particles with *imove* < 0 flag is not a good option, so AQUAgpusph provides a parallel interface as has been described on chapter 4.2.9. This feature allows you to mix 'Ghost particles' boundary condition with all other previous described, but probably you only wants to mix it with 'ElasticBounce boundary', in order to ensure than the fluid particles will not trespass the walls.
'Ghost particles' boundary has the advantages of be easily defined in simple cases, and has been widely employed along the SPH method history, so results can be easily compared. 'Ghost particles' have some consistency problems and is a condition bad defined in corners. When walls are perpendiculars the problem can be walked around adding diagonal walls as shown in figure 7.9, but for the moment general angles intersection can not be solved in AQUAgpusph with 'Ghost particles' boundary method yet.

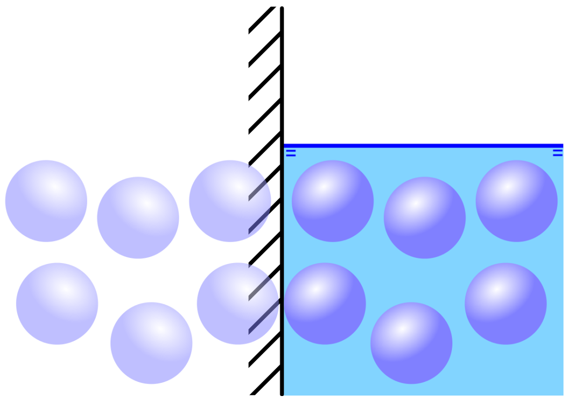


Figure 7.8: Ghost particles scheme

[5]and more generally for any parallel oriented one

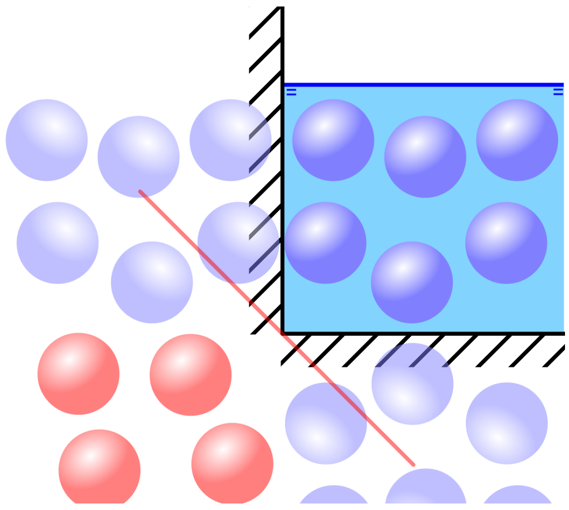


Figure 7.9: Ghost particles corner distribution

7.8 Motions

7.8.1 General

In AQUAgpusph the motions imposed externally can be classified in 2 ways, by the method to impose the motion, and by the method to compute the motion data. In table 7.1 all the available motions in AQUAgpusph are listed, with their classification.

Along this section all the different motion types, and control methods will be described. You can go to the chapter 8 to see practical applications of some number of available motions.

Movement	Motion type	Computation method
LIQuaternion	Quaternion defined by axes	Linear interpolated data table
ScriptQuaternion	Quaternion defined by axes	Python script

Table 7.1: AQUAgpusph available motions.

7.8.2 Motion types

Quaternion based motions

The main method to impose motions in AQUAgpusph is to use Quaternions, that have the key feature that the motion is ever fully described, not depending on any criteria. The problem with Quaternion imposed motions is that will overwrite all the previous motions, so if you want to superpose several motions, Quaternion one must be the first.
When Quaternion motion is imposed in reality you are setting a movable reference coordinates, where object is fixed, So in order to define instantaneous quaternion you must provide enough data to can describe it.

1. 2D Simulations (AQUAgpusph2D)

   (a) **C.x**: x coordinate of the quaternion center.

   (b) **C.y**: y coordinate of the quaternion center.

   (c) **X.x**: x component of the X quaternion axis.

   (d) **X.y**: y component of the X quaternion axis.

2. 3D Simulations (AQUAgpusph)
   (a) **C.x**: x coordinate of the quaternion center.
   (b) **C.y**: y coordinate of the quaternion center.
   (c) **C.z**: z coordinate of the quaternion center.
   (d) **X.x**: x component of the X quaternion axis.
   (e) **X.y**: y component of the X quaternion axis.
   (f) **X.z**: z component of the X quaternion axis.
   (g) **Y.x**: x component of the Y quaternion axis.
   (h) **Y.y**: y component of the Y quaternion axis.
   (i) **Y.z**: z component of the Y quaternion axis.

The undefined axis will be computed considering an orthonormal base. Provided axes will not be normalized, so please ensure to provide right axes data.
Quaternion motions provided in AQUAgpusph will apply the motion over the particles with *imove* ≤ 0 flag (Fixed particles or vertexes) and over defined walls as described on section 4.2.9. You can customize affected motion particles editing the provided Quaternion OpenCL kernel, but not walls for 'Ghost particles' boundary condition, that requires editing the AQUAgpusph source code.
In almost SPH simulations the time step is really small, for this reason in AQUAgpusph the velocity of the boundary points (fixed particles, vertexes, or wall points) is determined as an euler derivation

$$v_{t+dt} = \frac{x_{t+dt} - x_t}{dt}$$

7.8.3 Computation methods

Linear interpolated data table

The simplest method to set the motion data is providing a tabulated file with the required data (see section 7.8.2 to know what data is required by the selected motion). Tabulated file must contain one line per time instant with time at first column, and all the request fields by motion at the next ones.
Comments can be included in the file with the symbol '#'. All the line content after '#' symbol will be ignored. Fields can be separated by comma, semicolon, parenthesis, spaces or tabulator symbols, and if several separators are concatenated between the field values then will be joined as a unique space separator.
When values are requested for a time instant t, such that $t_n < t < t_{n+1}$, values will be linearly interpolated

$$x(t) = \frac{t - t_n}{t_{n+1} - t_n} x(t_{n+1}) + \left(1 - \frac{t - t_n}{t_{n+1} - t_n}\right) x(t_n)$$

You can see the section 8 to see practical application of linear interpolated tabulated motions data.

Python script controlled motion

The most flexible and powerful method to control the motion is using a Python script where 2 methods must be implemented:
init
Method used to give the needed fields, as described on section 7.8.2 for each type of motion, for the time instant $t = 0$s, that can be useful to set initial condition.
perform
Method called each time step in order to get all needed fields, as described on section 7.8.2 for each type of motion. Input and output arguments may vary for both methods depending on the type of motion selected. We really encourage user to go to chapter 8, where practical applications of Python controlled motions can be found.

7.9 Sensors

In AQUAgpusph sensors are points where pressure and density fields will be measured. The method to add sensors into the simulations has been described in the section 4.2.7.

Internally, the sensors are set as particles similar to the fluid ones, but with *imove* = 0 flag (fluid particles have *imove* > 0 flag, and solid boundary elements *imove* < 0), and a null mass. The fields are interpolated in the way described in section 2.3, where the discretized operators can be defined such that

$$\langle p \rangle_a = \frac{1}{\gamma_a} \sum_{b \in \text{Fluid}} \frac{p_b}{\rho_b} W_{ab} m_b \tag{7.1}$$

$$\langle \rho \rangle_a = \frac{1}{\gamma_a} \sum_{b \in \text{Fluid}} W_{ab} m_b \tag{7.2}$$

Sensors measured values are ever renormalized with the Shepard correction (γ_a is not forced to be equal to 1), even though Shepard correction has switched off for the forces and density rate computation, but in order to you can revert the correction γ_a (formerly *sumW*) is included into the output sensors data file.

The sensors output data file is printed in the execution folder (so enough permissions are expected) with the name "**Sensors.dat**", inside the file a header will be printed, where the fields included are documented, and then several rows (one per output time instant) with the following data distributed in columns:

1. Time instant t[s].
2. Sensor data columns (one per sensor).
   (a) X position X[m].
   (b) Y position Y[m].
   (c) Z position Z[m] (only for 3D cases).
   (d) Pressure p[Pa].
   (e) Density ρ[kg/m^3].
   (f) Kernel completeness factor γ.

The fields are separated by tabulators. This file can be directly plot with gnuplot, or loaded easily with almost data sheets.

Chapter 8

Examples

8.1 General

In this section the examples provided with the AQUAgpusph package are documented, introducing the most intelligible way to generate the case, the method to running it, and the post-process applied.

Some examples are based on papers, or has been used in some publications, that are conveniently documented in order to you can refer it if you want to learn more about the cases basis.

8.2 Lateral water impact 1x, with De Leffe boundary condition

8.2.1 General

This case is based on Botia-Vera et al. [2], where experimental data is available. This experiment has been simulated in AQUAgpusph and published in Macià et al. [8]. As menitoned in source paper, you can get all the experiment data from `http://canal.etsin.upm.es/ftp/SPHERIC_BENCHMARKS`. Here we will perform a relaxed version of the simulation published. The topics that will be covered in this example are:

1. How to setup a physical model, with some subtopics:
   (a) How to setup default AQUAgpusph solver for the Navier-Stokes equations (See section 2).
   (b) How to setup a domain.
   (c) How to setup De Leffe boundary condition.
   (d) How to setup a linear interpolated quaternion motion.
2. How to discretize the domain.
3. How to discretize the time.
4. How to run and track a simulation.
5. How to plot output data with gnuplot.
6. How to visualize output with Paraview.

This example is related with the next one, where the same experiment is simulated but ghost particles boundary condition is used instead of boundary integrals one.

In order to get a ready to run instance of this example 2D, version of AQUAgpusph package must be built, and examples must be switched on at CMake configuration, as described in section 3.3.2. You can find the example either on the built package, at the subfolder “examples”, and on the installed package, at “${CMAKE_INSTALL_PREFIX}/${CMAKE_INSTALL_DATADIR}/examples” (see section 3.3.2 to learn more about this folder).

In this example we will build the case from the scratch, creating all the files required manually, aiming to a more illustrative process, but usually you don’t want to follow this way and you may consider to use previously generated cases and edit them. Following examples will be performed in a more practical way.

8.2.2 Case description

The test is focused on a wave impact problem, being a rectangular tank with a roll forced motion, where pressures along the time are registered in specific locations. The main objective is reproduce the wave impact pressure registered.

In the figure 8.1 a schematic view of the tank with the dimensions and sensors placed is shown. The thickness of the tank is 62mm, and is filled with 93mm of water. In the scheme the rotation center is shown as well.

In the figure 8.2 the pressure registered on sensor 1 along the experiment is shown, with the roll forced motion. Must be noticed that the forced motion is almost a sinusoidal motion but for the initialization stage. The most relevant impact is the first one, being the main objective to simulate.

Since the thickness of the tank is small compared with the other dimensions, and Reynolds number high due to the water fluid usage, the effect on direction perpendicular to paper is neglected, considering a 2D case.

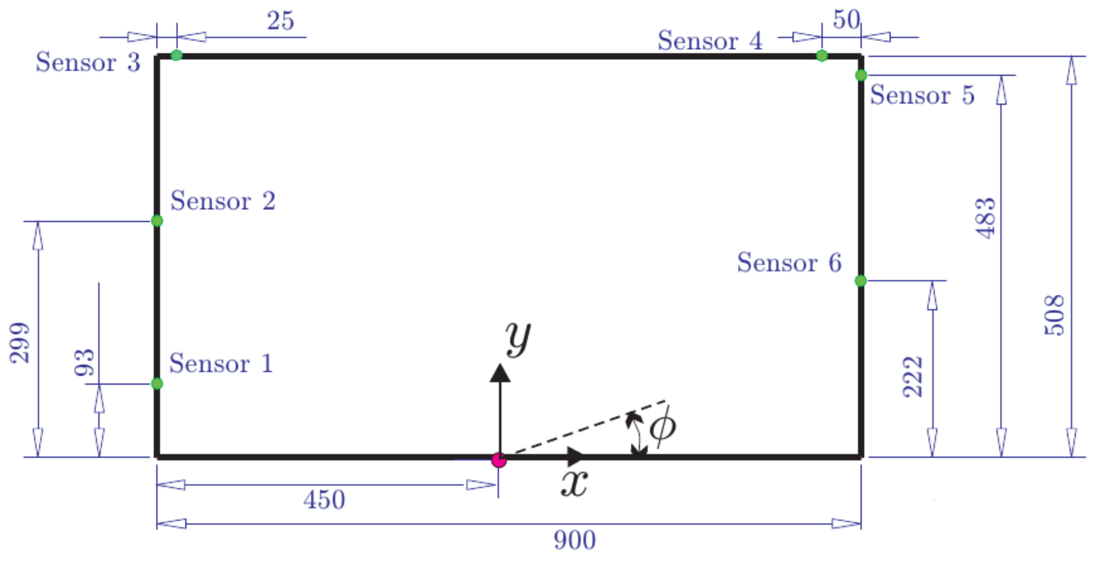


Figure 8.1: Tank dimensions and sensor positions

8.2.3 Creating folder and files

We will use a similar folder and files structure than the used on the provided example with the AQUAgpusph package. First generate the folder when do plain to work (hereinafter $EXAMPLE_PATH), and into the folder generate 2 additional subfolders:

1. **doc**: We will place here some files needed to plot the output data.

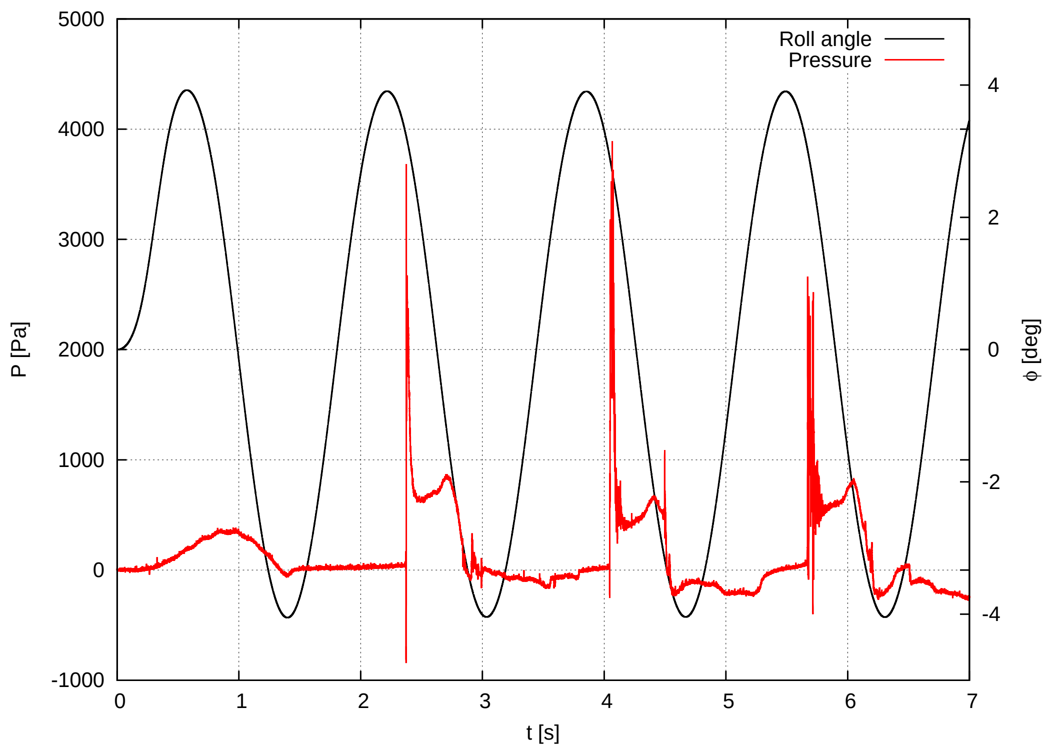


Figure 8.2: Pressure registered on sensor 1, and the motion registry

2. **Move**: We will place here all the motions data, including the description XML file.

Then go to the benchmark web page in order to download the file "lateral_water_1x.txt"[1]. This file contains the experimental data, so place it in the "doc" subfolder.

Now we are ready to start setting up the case data. For practical purposes hereinafter we will consider that AQUAgpusph has been installed using following options (almost of them are the default options):

```
AQUAGPUSPH_3D              = OFF
AQUAGPUSPH_BUILD_DOC       = OFF
AQUAGPUSPH_BUILD_EXAMPLES  = ON
CMAKE_INSTALL_PREFIX       = /usr
CMAKE_INSTALL_BINDIR       = bin
CMAKE_INSTALL_DATADIR      = share/aquagpusph
CMAKE_INSTALL_DOCDIR       = share/doc/aquagpusph
CMAKE_INSTALL_INCLUDEDIR   = include/AQUAgpusph
CMAKE_INSTALL_LIBDIR       = lib
```

So resources have been installed into "/usr/share/aquagpusph/resources" folder. Also you can find the examples resources (sources and scripts) in the folder "/usr/share/aquagpusph/examples", please refer to it in order to see the final resulting configuration files.

[1]You can get this from the example provided with the package too

8.2.4 Setting physical model

General

Several XML files will be created now, so hereinafter, if not any other indication is present, we will assume that when a new XML file is created, this content will be included by default:

```
<?xml version="1.0" ?>
<sphInput>
</sphInput>
```

Where we will insert data between "sphInput" tags.

Physical model will be distributed in several sections:

1. **Equations to solve**: AQUAgpusph allows you to set the OpenCL codes that will be used to perform the simulations, allowing to change the equations to solve as well.
2. **Domain**: Domain related data, i.e. computational domain, gravity acceleration, fluids properties, corrections, ...
3. **Boundary conditions**: Type of boundary condition used, no-slip/free-slip condition, ...
4. **Movements**: Motions description.

The data of these physical model sections are distributed in several files. You can refer to chapter 4 to know in detail the place where data must be allocated.

Equations to solve

We will set the standard AQUAgpusph solver that is described in chapter 2, so we will set the OpenCL codes provided with AQUAgpusph package. Create a XML file called "**OpenCL.xml**", with a section called "OpenCL", and add into the default OpenCL code files. Finally you may get a file similar to this one:

```
<?xml version="1.0" ?>
<sphInput>
<OpenCL>
<Predictor file="/usr/share/aquagpusph/resources/OpenCL/Predictor" />
<LinkList file="/usr/share/aquagpusph/resources/OpenCL/LinkList" />
<Rates file="/usr/share/aquagpusph/resources/OpenCL/Rates" />
<Corrector file="/usr/share/aquagpusph/resources/OpenCL/Corrector" />
<TimeStep file="/usr/share/aquagpusph/resources/OpenCL/TimeStep" />
<Reductions1D file="/usr/share/aquagpusph/resources/OpenCL/Reductions1D" />
<Reductions file="/usr/share/aquagpusph/resources/OpenCL/Reductions" />
<RadixSort file="/usr/share/aquagpusph/resources/OpenCL/RadixSort" />
<Shepard file="/usr/share/aquagpusph/resources/OpenCL/Shepard" />
<Domain file="/usr/share/aquagpusph/resources/OpenCL/Domain" />
<ElasticBounce file="/usr/share/aquagpusph/resources/OpenCL/Boundary/ElasticBounce" />
<DeLeffe file="/usr/share/aquagpusph/resources/OpenCL/Boundary/DeLeffe" />
</OpenCL>
</sphInput>
```

Note that OpenCL code file extension is automatically append. In the provided example this file has been replace by a general version provided with AQUAgpusph package that you can find in "/usr/share/aquagpusph/resources/O
path.

Domain

First for all we will set the domain fields, to do that we will generate a XML file called "**SPH.xml**" with a section called "SPH". The fields that we want to set are:

1. ***g***: Is the gravity acceleration. if you want to define other volumetric forces you can do it, as an acceleration unless you set you own mod AQUAgpusph version (see section 8.2.4). Since this is a 2D simulation this field must have 2 components.

2. c_s: Sound speed. Is a specific fluid property, but in weakly compressible SPH this value is really relevant on global properties, like the time step, AQUAgpusph needs a global value set. Usually this value is significantly lower than the real one.

3. γ: Batchelor' 67 state equation exponent. Is a specific fluid property, but if some fluid has this property undefined, this value will used.

To do it add following *Options*:

```
<Option name="g" x="0.0" y="-9.81" />
<Option name="gamma" value="1.0" />
<Option name="cs" value="45.0" />
```

Also you can set what corrections will be applied on the simulation:

1. **Steps between Link-List steps**: Link-List is a little bit time consuming stage, so don't preforming this operation each time step you can win some speed-up, but increasing the time steps between Link-List implies increasing the cells heigh, so more neighbours per particle will computed, and computational time increased, so take care with this value. In this example we will set that Link-List will performed each time step.

2. **Density reinitialization**: Due to the compressibility the density and pressure fields can turn too noisy along the simulation, but you can interpolate the density field from the particles positions data. Be mindful that density reinitialization can carry some instabilities. In this example we will unset the density reinitialization (setting 0 as the steps between reinitializations).

3. **Shepard**: See section 2.3.2 to learn more about the renormalization factor. Renormalization factor is a correction that can improve some mathematics properties of the interpolation, however, using it the fully conservation formulation is broken, that is a missed feature. At the other hand Shepard correction is mandatory when boundary integrals is imposed as the boundary condition, so in this case we may activate it, but in order to set the simulation as many robust as possible, we will only apply Shepard correction for forces computation, but not for density rate (Shepard at density rate increase the the compressibility and the noisy density field therefore).

Additionally, we will set a computable domain. The particles that go out the computable domain will be replaced by a particle with null mass, and the motion flag conveniently changed to $imove = 0$ (a sensor). If you don't set a computable domain, infinite domain will considered, but in this case, if one or more particles goes too far (go out from tank in this example) the simulation may crash because the number of cells can be greater than the allocatable one.

After the changes your file may looks like this (later we will append more data to this file):

```
<?xml version="1.0" ?>
<sphInput>
<SPH>
```

```
<Option name="g" x="0.0" y="-9.81" />
<Option name="gamma" value="1.0" />
<Option name="cs" value="45.0" />
<Option name="LLSteps" value="1" />
<Option name="DensSteps" value="0" />
<Option name="Shepard" value="Force" />
<Option name="Domain" x="-0.60" y="-0.15" l="1.2" h="0.85" />
</SPH>
</sphInput>
```

Regarding the fluid properties, we will create only one phase (the water one) that we will store in a new XML file called “**Fluids.xml**”, with a section called “Fluid”. Each “Fluid” section is considered as a new fluid, with different properties.

We can add the physical properties of the fluid:

```
<Option name="gamma" value="1.0" />
<Option name="refd" value="998.0" />
<Option name="Viscdyn" value="0.000894" />
```

Also we can set the artificial viscosity α parameter. Since weakly compressible SPH method is a purely explicit formulation, in order to get an stable evolution process you may guarantee a minimum dissipation, that in SPH is carried increasing the viscosity using the α parameter, that is defined such that

$$\alpha = \frac{K\,\nu}{\rho\,c_s\,h}$$

where $K = 6, 8, 15$ for $1D$, $2D$ and $3D$ cases respectively, ν is the dynamic viscosity, ρ is the density, c_s is the sound speed, and h is the kernel characteristic height. If the alpha value specified is not reached then the dynamic viscosity is artificially increased in order to accomplish this requirement, decreasing the Reynolds number therefore, but improving the stability. Usually, to get a stable simulation $\alpha \geq 0$, in this case we will use $\alpha = 0.03$.

```
<Option name="alpha" value="0.03" />
```

Boundary conditions

In this example boundary integrals (Formerly De Leffe [5]) will applied. Boundary integrals implies internally the simple boundary condition usage, so the options relative to this boundary condition must be set too. De Leffe boundary condition is set adding to the “SPH” section of the “SPH.xml” file the following option:

```
<Option name="Boundary" value="DeLeffe" />
```

When Fixed particles, De Leffe, or simple boundary conditions are set, the program expects to get a set of particles with motion flag such that $imove < 0$, that in the case of De Leffe or simple, represents the area elements of the walls. Walls area elements distribution will be discussed on the section 8.2.5, about the spatial discretization. With almost boundary conditions (with only simple boundary condition is not possible) a no-slip and free-slip approaches can be considered. Usually, if you can’t reach an enough big Reynolds number probably you can consider use free-slip approach in order to don’t penalty the simulation with an additional dissipation, in other case no-slip approach is the right way. In this case we have set an artificial viscosity factor of $\alpha = 0.03$, so the dynamic viscosity will be corrected from a real value of 0.000894 Pa/s to 0.389894 Pa/s, so the Reynolds number has been artificially increased 400 times, so we may use free-slip boundary condition since we are really far to the real Reynolds number. Add then the following option:

```
<Option name="SlipCondition" value="FreeSlip" />
```

No-slip boundary condition can be considered some times even thought the Reynolds number has not been reached because can result in slightly more stable simulations due to the increased dissipation on walls, but as many far you will for the real Reynolds number, many more instabilities can be experienced.

Regarding the simple boundary condition used internally by De Leffe's one, 2 options should be configured:

1. **Effect distance**: Value relative to the kernel height h, establishes the distance to wall area element where a particle is enough near to apply the elastic bound. If this value is set to zero the simple boundary condition will be switched off. In this example we use 0.1 h.

2. **Elastic factor**: Set the amount of energy conserved on the interaction, meaning that particle velocity will be mirrored against the wall, and the multiplied by this factor. 1 means a fully conservative interaction while 0 is for a fully dissipative interaction, intermediate values imply partial energy conservation. In this example fully dissipative approach is used.

So we add 2 more options to "SPH":

```
<Option name="BoundDist" value="0.1" />
<Option name="BoundElasticFactor" value="0.0" />
```

Movements

In this case a forced roll motion must be applied, where we have a good angle registry along the time. Since the motion data is known, we can use "Linear interpolated" motion control, explained in section 7.8.3. Regarding the method to define the position in this example the quaternion based one is used, described in section 7.8.2.

First we need to convert the angular data to quaternion data. In $2D$ quaternion is defined by the center of rotation and the x axis, being the y axis automatically computed.

In the example provided with AQUAgpusph package you can see an example on how to perform this operation using LibreOffice, but in this example we will create an small Python script that performs this operation easily.

We starts generating a file called "**CreateMotion.py**". This script may starts reading the data from "lateral_water_1x.txt" file, and converting angles to radians:

```
import math
f     = open('doc/lateral_water_1x.txt', 'r')
lines = f.readlines()
lines = lines[1:]     # Discard header line
data  = []
for l in lines:
    fields = l.split()
    time   = float(fields[0])
    angle  = math.radians(float(fields[2]))
    data.append((time, angle))
f.close()
```

And now we can write the data file for AQUAgpusph motion. You can find how this file must be formatted in the section 7.8.3.

```
f     = open('Move/6DOF.dat', 'w')
for d in data:
    # Write time
    f.write('(%g)\t' % (d[0]))
    # Write CoR
    f.write('(0.0,0.0)\t')
    # Write x axis
    f.write('(%g,%g)\n' % (math.cos(d[1]), math.sin(d[1])))
f.close()
```

In order to execute this script and generate de data file “Move/6DOF.dat”, open a terminal in the example folder and type

```
python CreateMotions.py
```

Having the file in the right format, now we can generate a “LIQuaternion” motion XML description file. In order to do it create the file “**Move/6DOF.xml**”, with the following content:

```
<Movement>
<DataFile file="Move/6DOF.dat"/>
<Script file="/usr/share/aquagpusph/resources/OpenCL/Movements/Quaternion.cl" />
</Movement>
```

That set as the data file the recently generated, and as the OpenCL code the standard provided with AQUAgpusph package, that moves only the sensors and the boundary particles ($imove \leq 0$).

Finally we need to generate the motion XML definition file, to do it create a file called “**Movements.xml**” with the following content:

```
<?xml version="1.0" ?>
<sphInput>
<Movements>
<Movement type="LIQuaternion" file="Move/6DOF.xml" />
</Movements>
</sphInput>
```

That adds only one motion, of the type “LIQuaternion”, with the description file that we have generated previously.

Each type of motion has different specific motion description XML file, for this reason you may reference it in the general movements description file “Movements.xml”.

8.2.5 Spatial discretization

Probably the key feature of SPH is that is a meshfree, so really complex geometries can be managed due to you only need to create a set of particles.

In this case the geometry is quite simple, so no complex operations to set the particles is needed, but a simple Python script will be enough for our interest, that we will call “CreateParticles.py”.

In order to know more about the format of the input particles distribution file see the section 4.3.1. In the same script the fluid particles and DeLeffe boundary elements must be written, see section 7.7.4 to learn more about this. In the figure 8.3 the particles Cartesian distribution scheme is shown. While the tank is defined by the dimensions L, H, the fluid is defined by the dimensions L, h, so the number of particles in each direction will be different for the fluid particles and the boundary elements (formerly vertexes), let nx, ny as the number of fluid particles in each direction and Nx, Ny as the number of vertexes, known that

$$Nx = nx \tag{8.1}$$

So the first thing that we need to do is compute the number of particles and vertexes to generate.

```
# General settings
cs     = 45.0
refd   = 998.0
gamma  = 1.0
g      = 9.81
# Tank dimensions
L      = 0.9
H      = 0.508
# Fluid
h      = 0.093
# Stimated required number of fluid particles
n      = 250000
# Calculate the number of particles at X/Y
ny     = int(round(math.sqrt(n*h/L)))
nx     = int(round(math.sqrt(n*L/h)))
n      = int(nx*ny)
# Calculate distance between particles
dr     = L/nx
hFluid = (ny-0.5)*dr
# Calculate solid particles
Nx     = nx
Ny     = int(round(H/dr))+1
N      = 2*Nx + 2*Ny
# Correct tank dimensions
L      = Nx*dr
H      = Ny*dr
```

Since in almost cases is not possible to get a distance between particles that can fit with L, H, h dimensions, L dimensions is selected as driver, correcting the other dimensions in order to fit it to the distance between particles, assuming that since $dr \ll 1$ the error will be small.

Then we can write the fluid particles, that have $imove > 0$ flag. Fluid initial state is rest, so indications from the section 2.3.6. Regarding the positions we have placed our coordinates origin at the tank center of rotation presented on the figure 8.1.

```
output = open("Particles.dat","w")
for i in range(0,n):
    j      = i
    idd    = j/ny
    idx    = idd / nz
    idy    = j%ny
    idz    = idd % nz
    imove  = 1
    pos    = (idx*dr - 0.5*(D-dr), idy*dr - 0.5*(L-dr), idz*dr + 0.5*dr)
    press  = refd*g*(hFluid - pos[2]);
    dens   = pow( press/prb + 1.0, 1.0/gamma )*refd;
    mass   = dens*dr**3.0
    string = "%g %g %g 0.0 0.0 0.0 0.0 0.0 0.0 %g %d %g %g\n" % \
    (pos[0], pos[1], pos[2], mass, imove, dens, cs)
    output.write(string)
```

And finally the vertexes, where the mass is in reality the area associated to the element, in $2D$ cases like this the line length. The vertexes may be marked with $imove < 0$ flag[2].

```
for i in range(0,N):
    if(i < Nx):                 # Bottom
        j = i
        idy = 0.0
        idx = j+0.5
        normal = [0.0,-1.0]
    elif(i < Nx + Ny):          # Right
        j = i-Nx
        idx = Nx
        idy = j+0.5
        normal = [1.0,0.0]
    elif(i < 2*Nx + Ny):        # Top
        j = i-Nx-Ny
        idy = Ny
        idx = Nx-j-0.5
        normal = [0.0,1.0]
    elif(i < 2*Nx + 2*Ny):      # Left
        j = i-2*Nx-Ny
        idx = 0.0
        idy = Ny-j-0.5
        normal = [-1.0,0.0]
    pos = (idx*dr - 0.5*L, idy*dr)
    if pos[1] <= h:
        press = refd*g*(hFluid-pos[1]);
        dens  = pow( press/prb + 1.0, 1.0/gamma )*refd;
    else:
        dens  = refd;
        press = prb*pow( dens/refd , gamma - 1.0 );
    imove = -1
    mass  = dr    # DeLeffe boundary condition store face areas into the vertex mass
    string = """%g %g %g %g 0.0 0.0 %g %d %g %g\n""" % \
    (pos[0], pos[1], normal[0], normal[1], mass, imove, dens, cs)
    output.write(string)
print("Particles written: %d" % n+N)
print('Particle distance: %g' % (dr))
```

All the times that you create a script in order to generate the particles and vertexes you may have a method to know the number of particles generated, and the distance between particles, or to be capable to set it automatically in the proper definition XML file. In this script the number of particles and vertexes, and the distance

[2]You can use several flags for different walls, that can help you in order to post-process the case, or if you want to perform custom OpenCL scripts in order to can differentiate them

between particles, is shown in the terminal when the script ends the work.

To launch the script open a terminal in the root example folder (the same path of the python script that we have created), and type

```
CreateParticles.py
```

The script will generate the particle into a file called "Particles.dat", and will answer

```
Particles written: 255223
Particle distance: 0.000578778
```

So the file "**Fluids.xml**" must be edited (you have been generated this file previously) in order to set the number of particles as attribute "*n*" in the Fluid tag, and in order to set the file to load the particles from that we have been generated with the Python script. Your final "**Fluids.xml**" file must be like this one:

```
<?xml version="1.0" ?>
<sphInput>
<Fluid n="255223">
<Option name="gamma" value="1.0" />
<Option name="refd" value="998.0" />
<Option name="Viscdyn" value="0.000894" />
<Option name="alpha" value="0.03" />
<Load file="Particles.dat" />
</Fluid>
</sphInput>
```

And the "**SPH.xml**" definition file must be edited as well adding the distance between particles in the section "SPH":

```
<Option name="deltar" x="0.000578778" y="0.000578778" />
```

8.2.6 Time discretization

In AQUAgpusph the time is discretized in order to forward integrate it using the Leap-Frog scheme documented in section 7.4. To perform the time discretization 2 main magnitudes must be controlled, the simulation total time, and the time step. Also we can setup a stabilization time.

First for all create the XML definition file "**Time.xml**" with the section called "Timing". We want to run a simulation of 5 seconds without a stabilization stage and with a time step such that is adapted in real time to fit to the criteria described in section 2.3.7, that in this type of cases can be really useful due to in the sloshing instant some particles can significantly accelerate getting high velocities, but reducing the time step we can manage the situation. To set these options we add following tags into the "Timing" section:

```
<Option name="SimulationStop" type="Time" value="5.0" />
<Option name="TimeStep" value="Variable"/>
<Option name="Stabilization" value="0.0"/>
```

In order to improve the Courant number a division factor can be set as described in section 4.2.5, in our case we will set a factor of 4 adding following option to the "SPH" section of the definition file "SPH.xml":

```
<Option name="DivDt" value="4.0" />
```

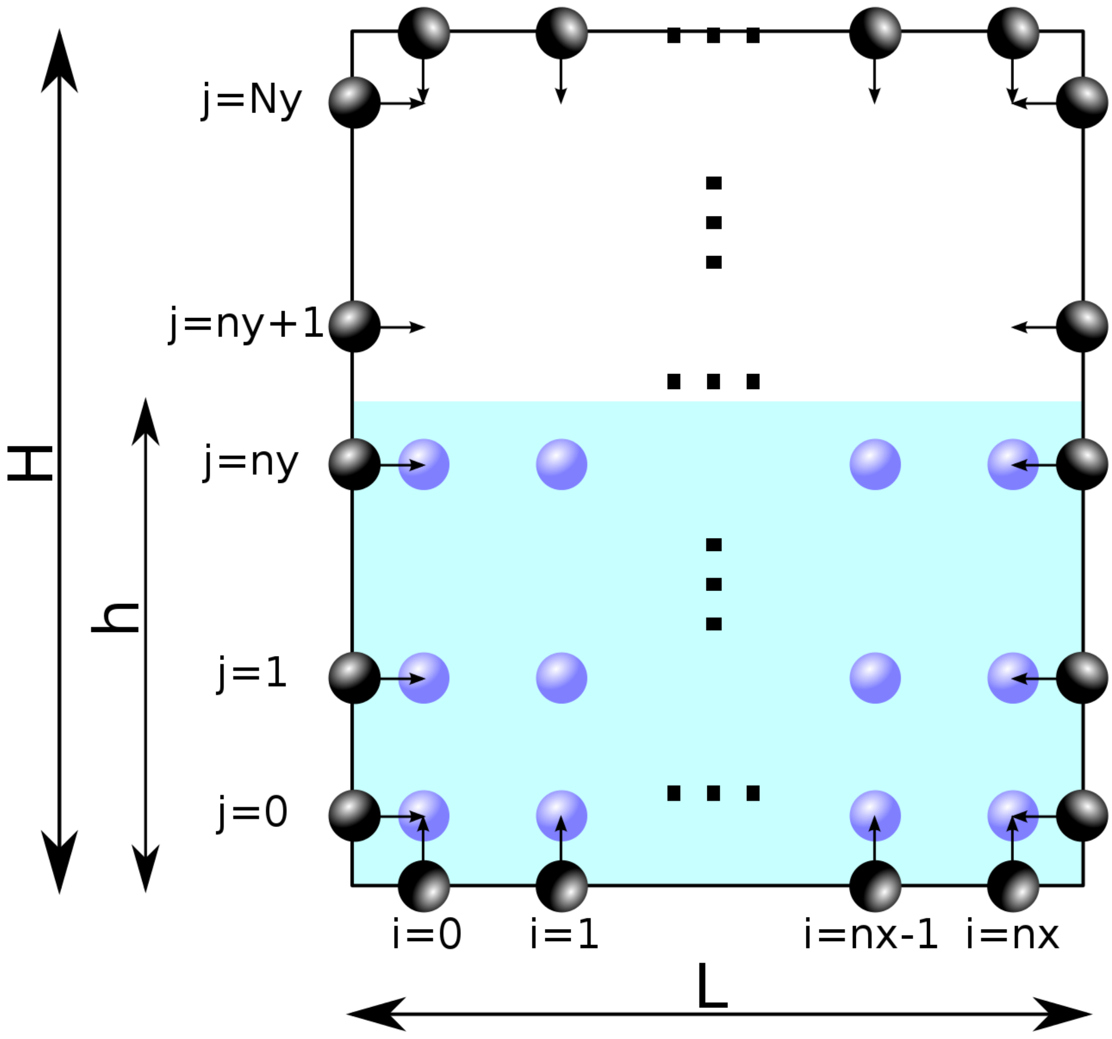


Figure 8.3: Particles packing scheme

8.2.7 Output configuration

In order to post-process the simulation we want several output files:

1. Sensor pressure: In the main output, and will require to set some sensors in the walls.
2. Visualization: H5Part output files will be written in order to can perform visualizations with Paraview.
3. Log file: In order to get a general view of the simulation process with relevant incidents, notifications and errors.
4. Energy: An energy log along the simulation will be written too.

In order to create the sensor 1 shown in the figure 8.1 we will add several measuring points in order to can cover all the real sensor area[3]. To do it create the XML definition file "Sensors.xml" with the following content:

```
<?xml version="1.0" ?>
<sphInput>
<Sensors>
<FPS value="10000.0" />
```

[3]Must be noticed that in the case that kernel height h is greater than the sensor length, adding several points will not make effect since you don't have resolution, but you can consider adding it in order to future resolution increasing.

```
<Script file="/usr/share/aquagpusph/resources/OpenCL/Sensors" />
<Sensor x="-0.45" y="0.09300" type="Interpolated" />
<Sensor x="-0.45" y="0.09275" type="Interpolated" />
<Sensor x="-0.45" y="0.09250" type="Interpolated" />
<Sensor x="-0.45" y="0.09225" type="Interpolated" />
<Sensor x="-0.45" y="0.09200" type="Interpolated" />
<Sensor x="-0.45" y="0.09325" type="Interpolated" />
<Sensor x="-0.45" y="0.09350" type="Interpolated" />
<Sensor x="-0.45" y="0.09375" type="Interpolated" />
<Sensor x="-0.45" y="0.09400" type="Interpolated" />
</Sensors>
</sphInput>
```

See the section 4.2.7 for further details about the options introduced. To set the rest of output files edit the file "**Time.xml**" adding following options into section "Timing":

```
<Option name="Output" format="H5Part" type="FPS" value="30" />
<Option name="LogFiles" type="FPS" value="120" />
<Option name="EnFiles" type="FPS" value="120" />
<Option name="BoundsFiles" type="No" />
```

Take care to don't set too high frequency of the visualization file output because huge hard disk demand can be expected.

8.2.8 Other options

We may set some general AQUAgpusph settings, see section 4.2.2 to learn more about the general options. To set this options simply generate a file called "Settings.xml" with the following content:

```
<?xml version="1.0" ?>
<sphInput>
<Settings>
<Verbose level="1" />
<Start mode="0" />
</Settings>
</sphInput>
```

8.2.9 Main XML definition file

We have been generated following files:

- Fluids.xml
- Movements.xml
- OpenCL.xml
- Sensors.xml
- Settings.xml
- SPH.xml
- Time.xml

- CreateMotions.py
- CreateParticles.py
- Particles.dat
- Move/6DOF.xml
- Move/6DOF.dat
- doc/lateral_water_1x.txt

The data files with the motions or the spatial discretization have been referenced into the corresponding XML files, and the specific motion definition file "Move/6DOF.xml" has been referenced from the general motions definition file "Movements.xml", but all the separately XML definition files must be grouped into a general definition file that will load AQUAgpusph in order to setup the case. Create a file called "**Main.xml**" with the following content:

```
<?xml version="1.0" ?>
<sphInput>
<Include file="Fluids.xml" />
<Include file="Movements.xml" />
<Include file="OpenCL.xml" />
<Include file="Sensors.xml" />
<Include file="Settings.xml" />
<Include file="SPH.xml" />
<Include file="Time.xml" />
</sphInput>
```

This file don't contains any relevant data, only redirects AQUAgpusph to read all the other previously generated files.

8.2.10 Run the simulation

This case has been generated using relative paths[4], so must be executed in the root folder where we have been generated it. Running the case is so quite easy, simply open a terminal in the folder where you have the file "**Main.xml**" and execute

```
AQUAgpusph2D --input Main.xml --no-reassembly
```

See section 5.1 to learn more about the cases launch process and the messages that you will receive while AQUAgpusph is running, and it's stored in the log file "Log.html". Depending on hardware used the simulation can take from 2 hours to more than 10 hours, so consider reduce the resolution or the number of neighbours in this case.

8.2.11 Post-process the simulation

General

Post-process can be divided in 2 main categories:

1. Run time post-process
2. Finished simulation post-process

The first one has the advantage that can be performed while the simulation is running, and consist mainly on plots of output data. The second one is mainly the visualization with the H5Part output.

[4]Relative paths provided into the configuration files are ever relatives to the execution path, not to the XML path

Run rime post-process

In this case the main objective is to plot the pressure obtained on sensor 1 (see figure 8.1), that is included in the data that can be processed in run time. We will use intensively **gnuplot** and **awk** along this section.

You can find more data about how is sensors file formatted in the section 7.9. Basically we want to read the pressure data on each of the 9 measuring points placed around sensor 1 position, and plot the average value along the time. With **awk**, get this pressure for each time instant written can be performed typing in a terminal on the execution folder

```
awk '{p=$4+$9+$14+$19+$24+$29+$34+$39+$44;p=p/9.0; print $1,p}' Sensors.dat
```

That loads the sensors output file “Sensors.dat”, compute the required value for each time step, and return 2 columns with the time instants and the pressure obtained. Regarding the experimental data, pressure has been measured in mbar, so we need to move it to Pa:

```
awk '{p=$2; print $1,100.0*p}' lateral_water_1x.txt
```

So we can obtain, using **awk**, 2 files that contains the experimental and simulated pressures, so we only need to create a **gnuplot** layout, that we can call “**press.gnuplot**” (place it in the root folder), that uses internally **awk** to plot the desired data.

```
# Set plot options
set key
set grid
# Set x parameters
set xlabel "Time [s]"
set xtic
set xrange[0:5]
# Set left y parameters
set ylabel "Pressure [Pa]"
set ytic
set yrange[-1000:5000]
# Line styles
set style line 1 lt -1 lw 1
set style line 2 lt -1 lc rgb "red" lw 1

# Plot
plot \
"<awk '{p=$2; print $1,100.0*p}' doc/lateral_water_1x.txt" \
using 1:2 title 'Experimental' axes x1y1 with lines ls 2, \
"<awk '{p=$4+$9+$14+$19+$24+$29+$34+$39+$44;p=p/9.0; print $1,p}' Sensors.dat" \
using 1:2 title 'SPH' axes x1y1 with lines ls 1

pause 5
replot
reread
```

The script shown above will plot the pressure data while it is available, reploting with in an interval of 5 seconds. In the figure 8.4 you can see a pressure plot example that you can expect using **gnuplot**.

As you can see the results can be significantly improved as shown in the publication [8] where this simulation example was introduced, but is enough for our purposes. Feel free to play with the parameters and see the effects.

In the same way described you can plot the energy graphs.

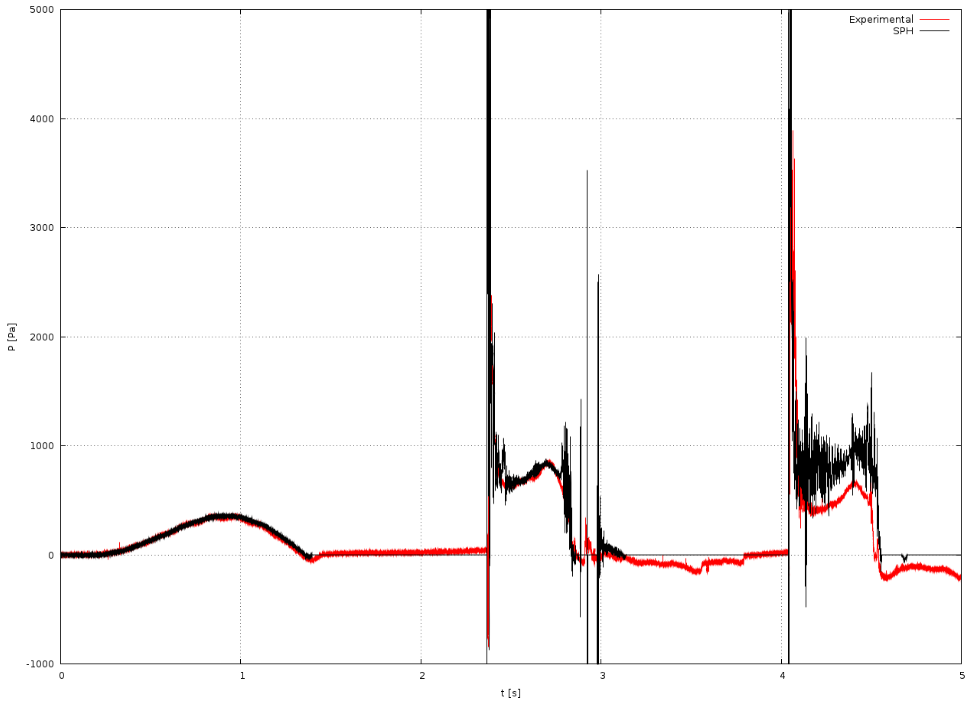


Figure 8.4: Pressure register comparison between experimental and simulation data

Finished simulation post-process

When the simulation has finished we can work over the H5Part output file. In this section we will use **Paraview**. Remember that your **Paraview** needs the H5Part files loader plugin to can run.

To start working executes Paraview, and go to “File/Open”. A new window opens in order to select a file to open, so select the output file “Particles0.h5part” and accept. At this point you still not seeing nothing on the 3D view, but a new filter has been added to the pipeline, as you can see in the figure 8.5. Below the pipeline you can find the options of this filter, ensure that *Xarray*, *Yarray* and *zArray* are right, and press **Apply**. In the figure 8.6 you can see how the options must be set.

After that something is shown in the 3D view, our first objective will be separate in 3 different instances the walls elements, the fluid particles, and the sensors, to do it we need to apply 4 filters, one to extract solid that has $imove < 0$ flag, another one to extract fluid that has $imove > 0$ flag, and 2 more filters for sensors that has $imove = 0$ flag.

Go to **Filters/Common/Clip**, and a new filter depending on our original one is added. You can change the filter name selecting it in the pipeline browser, and pressing “F2”, set the name “**Solid**”. Now set the options in the object inspector as shown in figure 8.7, and press Apply. In 3D view only solid elements are shown now.

Before to get fluid particles we will format solid elements style. Change to the “Display” tab, and modify “Color by” option to **Solid Color**. Change the “point size” to **1.00** too in the “Style” section. You may see something similar to the presented on figure 8.8.

Selecting the H5Part file instnace apply now another **Clip** filter, but in this case select the “Value” **0.5**, and uncheck “Inside Out”, naming to this new instance “**Fluid**”.

The fluid can be styled selecting “Color by” as **press**, and pressing “Edit Color Map” we can edit the color mapping behaviour. In figure 8.9 the options selected for the color map, in order to get the visualization shown in the figure 8.10 is shown. You can show the legend with the options of the “Color Legend” tab of the same window.

Finally add 2 new **Clip** filters, apply the first one to the H5Part loaded instance, with the “Value” **-0.5**, but with “Inside Out” unchecked, and apply the another filter to the first one with theç “Value” **0.5** and “Inside Out” checked. You can rename both filters as “**Sensors**”.

Sensors must be coloured with **press** too, but in this case the “point size” can be increased to and hight value, for instance **3.0**.

Finally you can change the background color with **Edit/View Settings...**. In this window you can also set/unset the parallel projection[5] and the annotations.

Now you have your visualization formatted, and you can start animating and saving images/videos.In the figure 8.11 you can see an example of an animation performed with ParaView.

You can additionally save a ParaView state that when you loads it apply automatically all the purposed filters.

Figure 8.5: Filter added to Paraview

8.2.12 Conclusions

In this example is shown how to setup a case from scratch, that even though this is the the most complex way to do it, is too easy to perform it. The example has been covered in detail, including the case setup, the simulation launch, and the post-process.

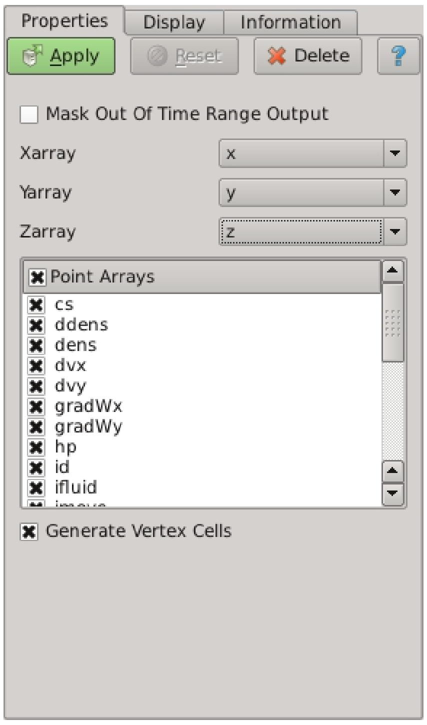
The unique tool that has been managed without text files has been Paraview, but if Paraview has been installed with Python support, the work that we have been performed in the section 8.2.11 can be placed in a Python script, that is a text file, so the entirely simulation could be easily scripted[6].

In the results a typical Weakly compressible SPH (W-SPH) pressure noise can be noticed, but results have a good quality.

During the case generation we have set free-slip boundary condition along the walls because the Reynolds number reached is so quite far from the real one (Dynamic viscosity has been artificially increased 400 times), but we could considered that our Reynolds number is enough to set a no-slip boundary condition, with the hope that we will receive a more stable simulation, and maybe with some better results. In the figure 8.12 the

[5]Is strongly recommended use parallel projection

[6]AQUAgpusph package has internally scripted the case with the CMake tool, that you can test it in comparing the built and installed versions, where all the paths into the definition files and scripts are different



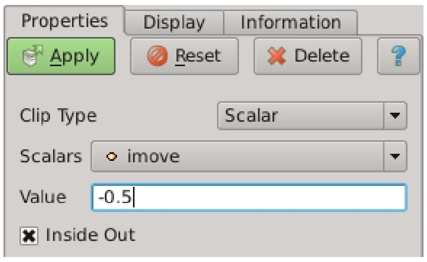
☐ Mask Out Of Time Range Output

Xarray: x
Yarray: y
Zarray: z

☒ Point Arrays
☒ cs
☒ ddens
☒ dens
☒ dvx
☒ dvy
☒ gradWx
☒ gradWy
☒ hp
☒ id
☒ ifluid

☒ Generate Vertex Cells

Figure 8.6: H5Part loaded file options



Clip Type: Scalar
Scalars: imove
Value: -0.5
☒ Inside Out

Figure 8.7: Solid Clip filter options

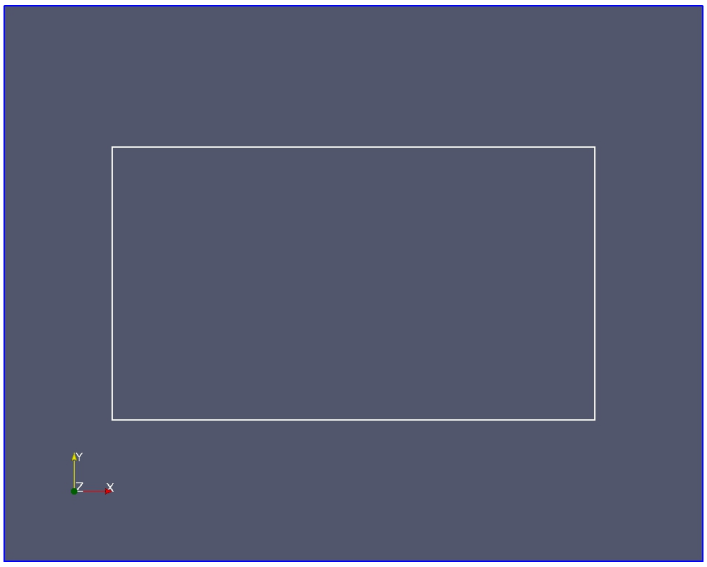
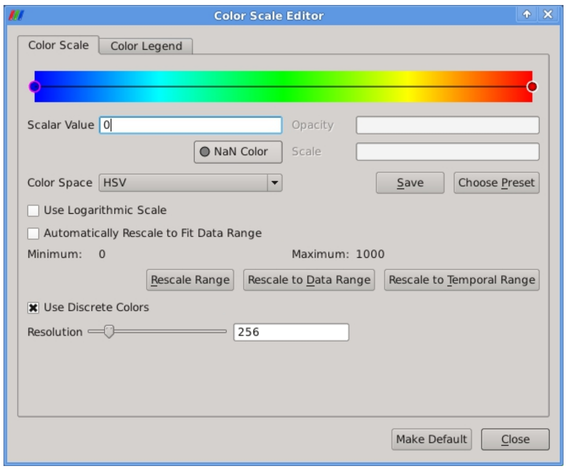


Figure 8.8: 3D view with only the solid elements

pressure measured in the sensor 1 and the simulation result using no-slip boundary condition is shown. You can compare the pressures computed using free-slip, shown in the figure 8.4, and using no-slip, shown in the


Color Scale Editor

Color Scale | Color Legend

Scalar Value: 0

Opacity

NaN Color

Scale

Color Space: HSV

Save | Choose Preset

☐ Use Logarithmic Scale

☐ Automatically Rescale to Fit Data Range

Minimum: 0

Maximum: 1000

Rescale Range | Rescale to Data Range | Rescale to Temporal Range

☒ Use Discrete Colors

Resolution: 256

Make Default | Close

Figure 8.9: Pressure color map options

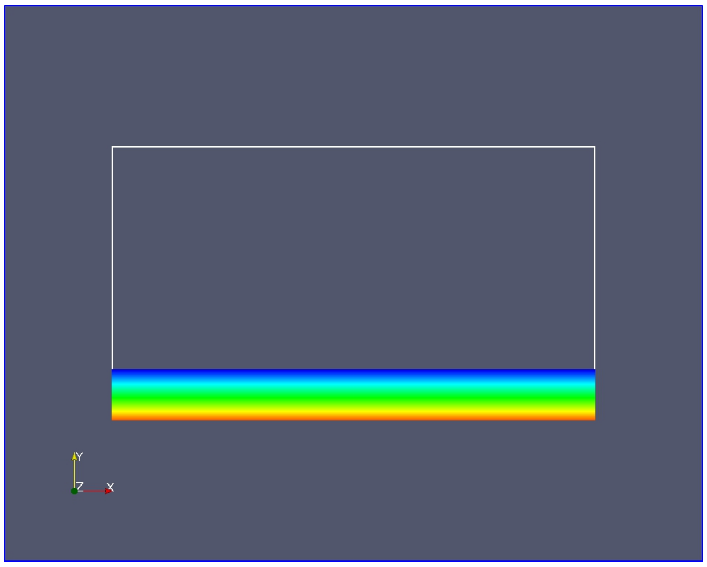


Figure 8.10: 3D view with the solid elements and the fluid particles

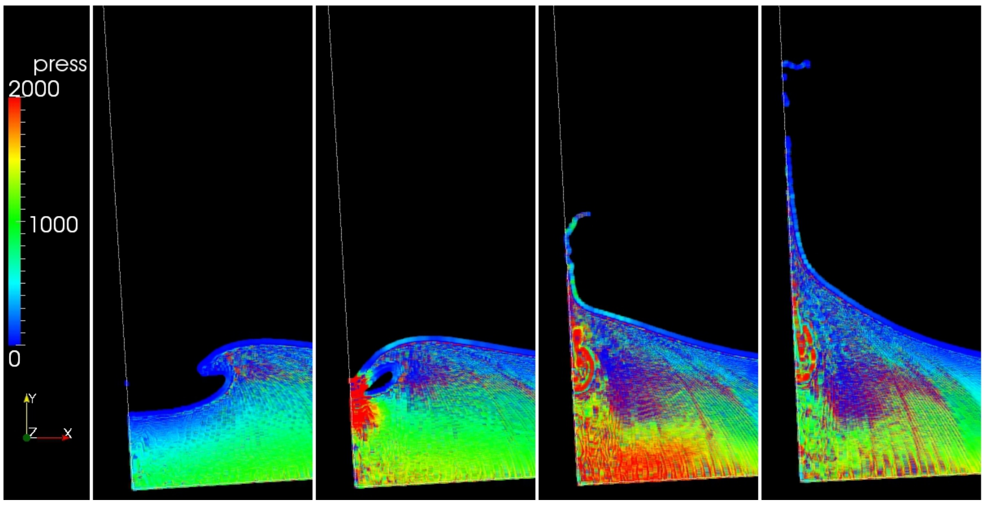


Figure 8.11: Animation of the example for the first impact, frames 70-73, using free-slip boundary condition

figure 8.12, where can be noticed that the best quality of the first one. In the figure 8.13 the same animation shown in the figure 8.11 for the free-slip is presented for the no-slip boundary condition.

In this example we have generated all the needed files in order to perform the simulation, that are ready installed on the system. Since we have been used relative directories to refer the files between them, we can only execute our example from the generated location, but the installed version can be executed in whatever place where we have permission to write. We will use this approach in the next example.

The next example is heavy related with this one, but the boundary condition has been moved from boundary integrals (formerly De Leffe or DeLeffe for the configuration files) to Ghost particles, that is probably the most popular boundary condition generally used.

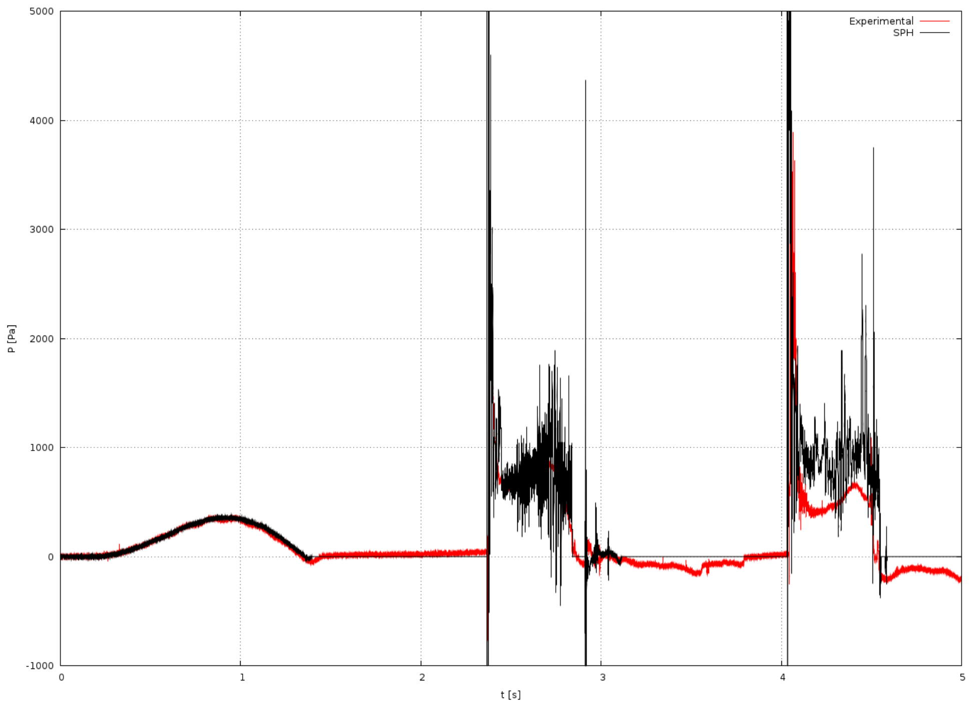


Figure 8.12: Pressure register comparison between experimental and simulation data, using no-slip boundary condition

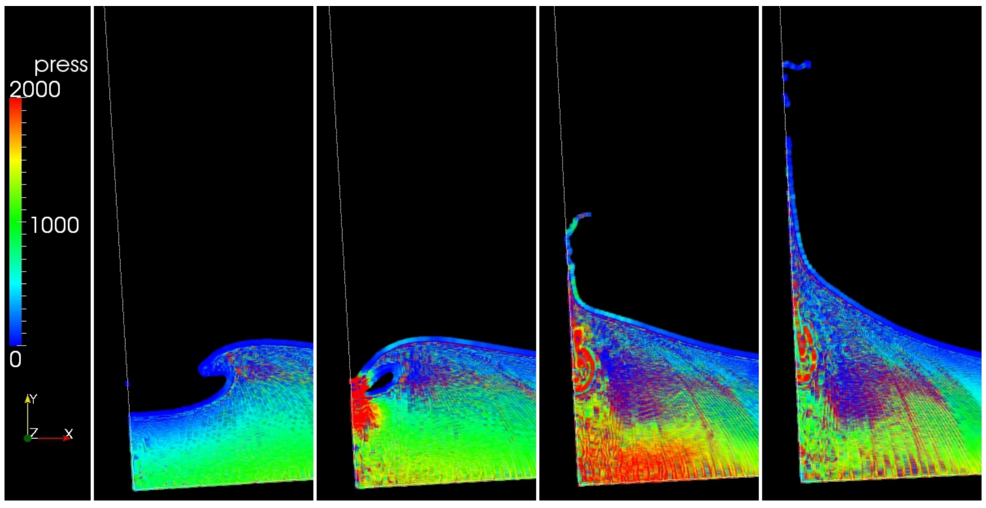


Figure 8.13: Animation of the example for the first impact, frames 70-73, using no-slip boundary condition

8.3 Lateral water impact 1x, with Ghost particles boundary condition

8.3.1 General

This example solves the same case of the previous example described in the section 8.2, but changing the boundary condition from the De Leffe's type to a ghost particles one, discussed on section 7.7.5.

The objective of this example is to show how the ghost particles are used and to get a comparison on the results obtained with each type of boundary condition.

At the other hand, in the previous example the full process to generate a case has been documented, but in this case we will cover only the differences with the previous example, using installed resources to run it.

The topics that will be covered in this example are:

1. Configuration differences due to the boundary condition change.
2. How to run and track a simulation installed on the system.
3. Results comparison using either boundary integrals or ghost particles boundary conditions.

In order to get a ready to run instance of this example, 2D version of AQUAgpusph package must be built, and examples must be switched on at CMake configuration, as described in section 3.3.2. You can find the example either on the built package, at the subfolder "examples", and on the installed package, at "${CMAKE_INSTALL_PREFIX}/${CMAKE_INSTALL_DATADIR}/examples" (see section 3.3.2 to learn more about this folder).

Hereinafter we assume that "${CMAKE_INSTALL_PREFIX}" has been set as "/usr", and the example can be found at "/usr/share/aquagpusph/examples/LateralWater_1x_Ghost" therefore.

8.3.2 Case description

You can find the case description in the section 8.2.2, where the same case has been described.

8.3.3 Settings changes required to modify the boundary condition

During the section 8.2.4 how to set a De Leffe's type of boundary condition has been described. In this case this boundary condition must be deactivated, but we can use the area elements generated to set a "ElasticBounce" boundary condition, described in section 7.7.2, in order to avoid the particles trespass the walls, so the option "Boundary" must be moved to "ElasticBounce":

```
<Option name="Boundary" value="ElasticBounce" />
```

Ghost particles walls are set in a different way of the other boundary conditions, where a set of special particles are used. To set ghost particles driven walls a file called "**GhostParticles.xml**" is created with the following content:

```
<?xml version="1.0" ?>
<sphInput>
<GhostParticles>
<TangentUModel value="SSM" />
<Wall>
<Vertex x="-0.45" y="0.0" />
```

```
<Vertex x="0.45" y="0.0" />
</Wall>
<Wall>
<Vertex x="0.45" y="0.0" />
<Vertex x="0.45" y="0.508" />
</Wall>
<Wall>
<Vertex x="0.45" y="0.508" />
<Vertex x="-0.45" y="0.508" />
</Wall>
<Wall>
<Vertex x="-0.45" y="0.508" />
<Vertex x="-0.45" y="0.0" />
</Wall>
<Wall>
<Vertex x="-0.704" y="0.254" />
<Vertex x="0.0" y="-0.45" />
</Wall>
<Wall>
<Vertex x="0.0" y="-0.45" />
<Vertex x="0.704" y="0.254" />
</Wall>
<Wall>
<Vertex x="0.704" y="0.254" />
<Vertex x="0.0" y="0.958" />
</Wall>
<Wall>
<Vertex x="0.0" y="0.958" />
<Vertex x="-0.704" y="0.254" />
</Wall>
</GhostParticles>
</sphInput>
```

In this case symmetric tangential velocity model is imposed in order to simulate free-slip boundary condition.

Of course this file must be referenced into "**Main.xml**" file too.

Since the "ElasticBounce" boundary condition has been retained, the spatial discretization still being valid.

8.3.4 Running the example installed version

In this example the installed version of the example will be directly executed. To do it, create a folder to work, for instance, a folder called "LateralWater_1x_Ghost" in your home folder, and move into:

```
mkdir ~/LateralWater_1x_Ghost
cd ~/LateralWater_1x_Ghost
```

The example provides a bash script that allows to easily run the case and plot the results during the simulation. To run the simulation type:

```
/usr/share/aquagpusph/examples/LateralWater_1x_Ghost/run.sh --run
```

This command will launch AQUAgpusph with the input file set and reassembly switched off, after cleaning previous execution results. In this case the simulation can take more time due to this boundary condition can be really less computational efficient.

This case will turn unstable and stop running at simulation time of 4.57208 seconds, with a time step too low to can continue working, so when this happens cancel the job pressing '**c**' key (allowing AQUAgpusph to close properly the case).

8.3.5 Post-process the simulation

The case post-process documented along the section 8.2.11 still being valid in this case, but for the plotting process we can use the provided script. In another terminal we can start the plot process executing:

```
/usr/share/aquagpusph/examples/LateralWater_1x_Ghost/run.sh --plot
```

Previous command copy the motion and pressures experimental registry into the execution folder, and launch *gnuplot*. In the figure 8.14 the pressure register computed using Ghost particles and the experimental one are shown. Can be noticed that with the same resolution and number of neighbours, this boundary condition returns really worst results than the shown in the figure 8.4, corresponding to a De Leffe's type free-slip boundary condition.

In the figure 8.15 the first impact visualization is shown.

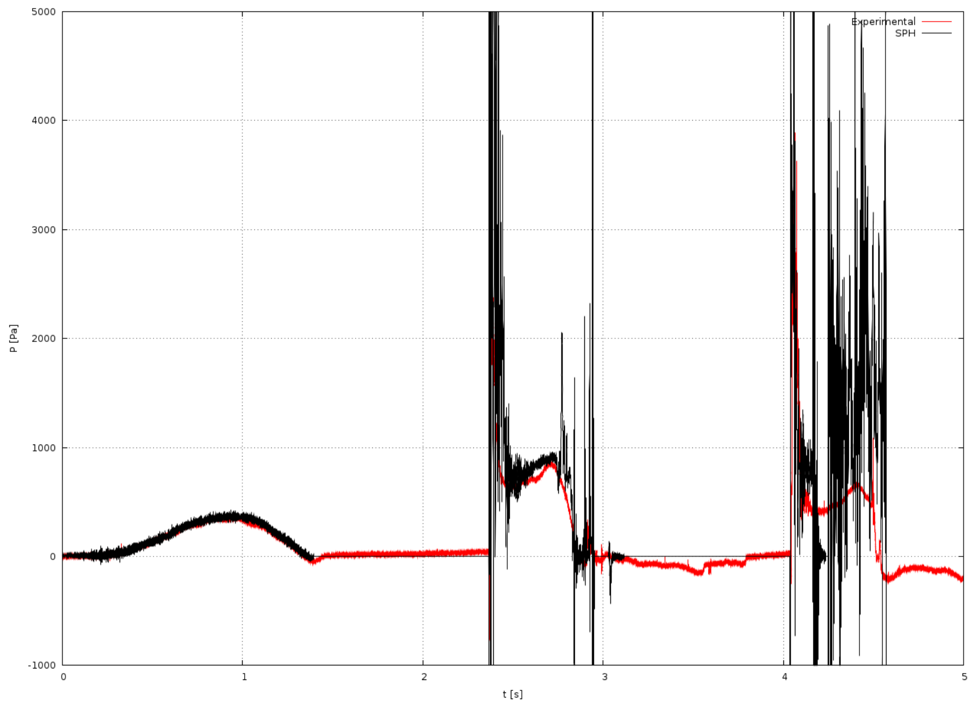


Figure 8.14: Pressure register comparison between experimental and simulation data, using ghost particles and free-slip boundary condition (Tangent velocity SSM model)

8.3.6 Conclusions

In this example two different boundary conditions has been test, the boundary integrals used in the previous example, and the ghost particles documented along this one. Comparing figures 8.4 and 8.14, corresponding to boundary integrals and ghost particles, you can see that the first one is able to reproduce better the experimental

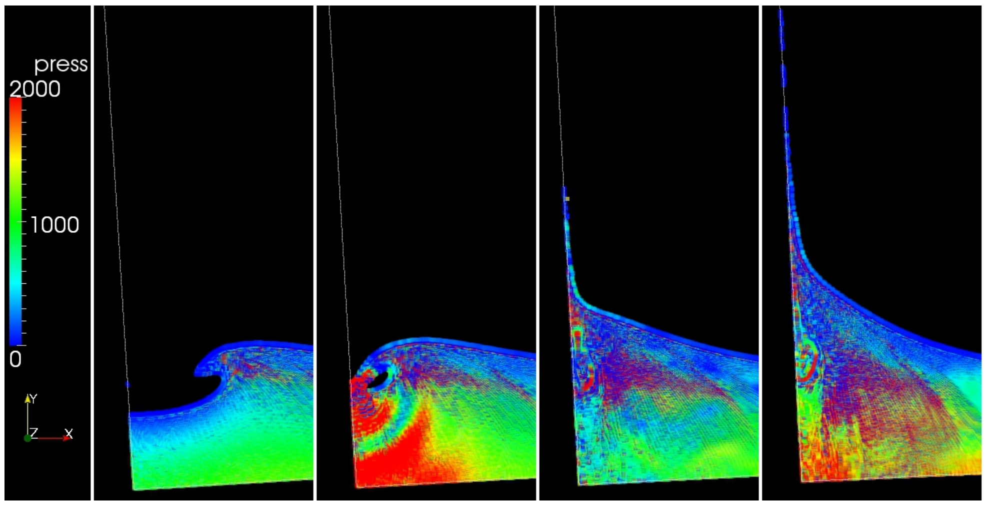


Figure 8.15: Animation of the example for the first impact, frames 70-73, using ghost particles and free-slip boundary condition

wave impact pressure register at sensor 1.

Also a SPH instability has been experienced in this case, causing the program crash due to a too small time step for the used precision.

In order to control the instability we can consider to set a no-slip boundary condition, that in the case of ghost particles is set with the tangential velocity model, changing it from a symmetric model “SSM” to an antisymmetric model “ASM”.

```
<TangentUModel value="ASM" />
```

But “ASM” model is inconsistent for almost general velocity fields when the Laplacian is computed, as **??** shown with polynomial velocity fields, so as $h \to 0$ the accelerations will blows up therefore, that in this case (where the resolution is fine enough) will accelerate the instability.

In figure 8.16 the simulation pressure register computed until $t \leq 0.7328s$, where the first instabilities can be appreciated much earlier than in the free-slip case.

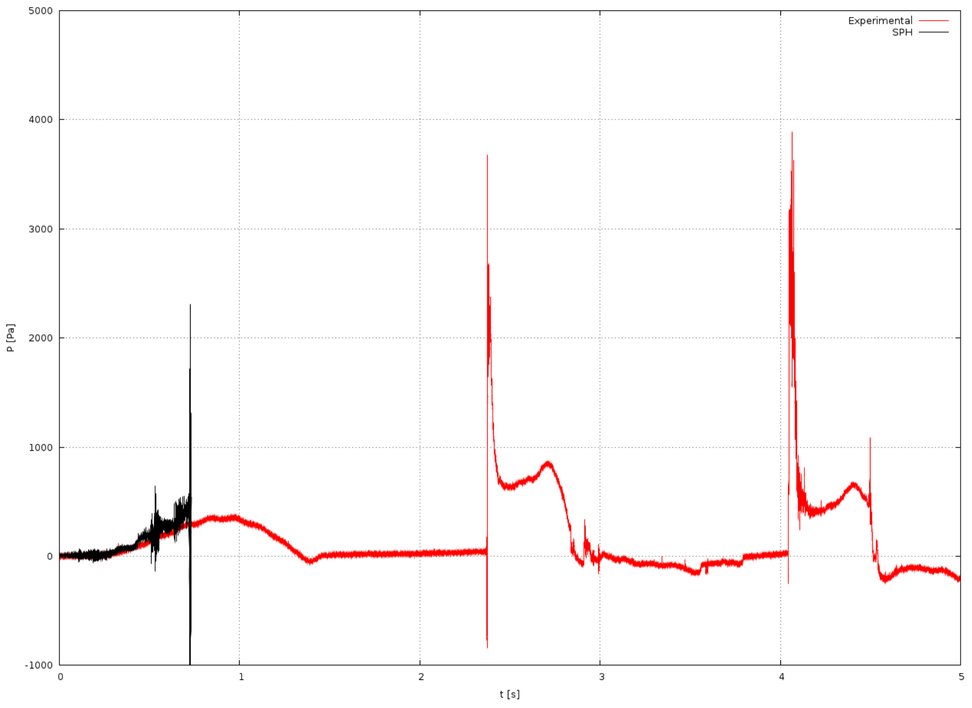


Figure 8.16: Pressure register comparison between experimental and simulation data, using ghost particles and no-slip boundary condition (Tangent velocity ASM model)

Bibliography

[1] Andrea Amicarelli, Jean-Christophe Marongiu, Francis Leboeuf, Julien Leduc, and Joëlle Caro. SPH truncation error in estimating a 3D function. *Computers & Fluids*, 44(1):279 – 296, 2011. ISSN 0045-7930. doi: DOI:10.1016/j.compfluid.2011.01.018. URL `http://www.sciencedirect.com/science/article/B6V26-52079TP-1/2/28aecd9efa85d26137ba55bcaa45daa4`.

[2] E. Botia-Vera, A. Souto-Iglesias, G. Bulian, and L. Lobovský. Three SPH Novel Benchmark Test Cases for free surface flows. In *5th ERCOFTAC SPHERIC workshop on SPH applications*, 2010.

[3] A. Colagrossi and M. Landrini. Numerical simulation of interfacial flows by smoothed particle hydrodynamics. *J. Comp. Phys.*, 191:448–475, 2003.

[4] Andrea Colagrossi, Matteo Antuono, and David Le Touzé. Theoretical considerations on the free-surface role in the Smoothed-particle-hydrodynamics model. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 79(5):056701, 2009. doi: 10.1103/PhysRevE.79.056701.

[5] M. De Leffe, D. Le Touzé, and B. Alessandrini. Normal flux method at the boundary for SPH. In 4^{th} *SPHERIC*, pages 149–156, May 2009.

[6] M. Ferrand, D. R. Laurence, B. D. Rogers, D. Violeau, and C. Kassiotis. Unified semi-analytical wall boundary conditions for inviscid, laminar or turbulent flows in the meshless sph method. *International Journal for Numerical Methods in Fluids*, 71(4):446–472, 2013. ISSN 1097-0363. doi: 10.1002/fld.3666. URL `http://dx.doi.org/10.1002/fld.3666`.

[7] Fabricio Macià, Matteo Antuono, Leo M. González, and Andrea Colagrossi. Theoretical analysis of the no-slip boundary condition enforcement in SPH methods. *Progress of Theoretical Physics*, 125(6): 1091–1121, 2011. doi: 10.1143/PTP.125.1091. URL `http://ptp.ipap.jp/link?PTP/125/1091/`.

[8] Fabricio Macià, Leo M. González, J. L. Cercos-Pita, and A. Souto-Iglesias. A boundary integral SPH formulation. Consistency and applications to ISPH and WCSPH. *Progress of Theoretical Physics*, 128 (3), September 2012.

[9] J. J. Monaghan. Smoothed particle hydrodynamics. *Reports on Progress in Physics*, 68:1703–1759, 2005.

[10] J. J. Monaghan and R. A. Gingold. Shock simulation by the particle method SPH. *Journal of Computational Physics*, 52(2):374–389, 1983.

[11] J.J. Monaghan. Smoothed particle hydrodynamics and its diverse applications. *Annual Review of Fluid Mechanics*, 44(1):323–346, 2012. doi: 10.1146/annurev-fluid-120710-101220. URL `http://www.annualreviews.org/doi/abs/10.1146/annurev-fluid-120710-101220`.

[12] Joseph P. Morris, Patrick J. Fox, and Yi Zhu. Modeling low Reynolds number incompressible flows using SPH. *Journal of Computational Physics*, 136:214–226, 1997. doi: http://dx.doi.org/10.1006/jcph.1997.5776.

[13] L. Pérez-Rojas and J.L. Cercos-Pita. 3D GPU SPH analysis of coupled sloshing and roll motion. In *11th International Conference on Stability of Ships and Ocean Vehicles (STAB 2012)*, September 2012.

[14] Nathan J. Quinlan, Martin Lastiwka, and Mihai Basa. Truncation error in mesh-free particle methods. *International Journal for Numerical Methods in Engineering*, 66(13):2064–2085, 2006. URL `http://dx.doi.org/10.1002/nme.1617`.

[15] A. Souto-Iglesias, L. Delorme, L. Pérez-Rojas, and S. Abril-Pérez. Liquid moment amplitude assessment in sloshing type problems with smooth particle hydrodynamics. *Ocean Engineering*, 33(11-12):1462–1484, 8 2006.

[16] Holger Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Adv. Comput. Math.*, 4(4):389–396, 1995. ISSN 1019-7168. doi: 10.1007/BF02123482. URL `http://dx.doi.org/10.1007/BF02123482`.