

# Users Guide for DualSPHysics code



DualSPHysics v4.0

April 2016

[dualsphysics@gmail.com](mailto:dualsphysics@gmail.com)



## **Abstract**

This guide documents the DualSPHysics code based on the Smoothed Particle Hydrodynamics model named SPHysics. This manuscript describes how to compile and run the DualSPHysics code (a set of C++ and CUDA files). New pre-processing tools are implemented to create more complex geometries and new post-processing tools are developed to analyse easily numerical results. Several working examples are documented to enable the user to use the codes and understand how they work.



## CONTENTS

<b>1. Introduction</b>	<b>7</b>
<b>2. Developers and institutions</b>	<b>9</b>
<b>3. SPH formulation</b>	<b>11</b>
<b>4. CPU and GPU implementation</b>	<b>31</b>
<b>5. Running DualSPHysics</b>	<b>35</b>
<b>6. DualSPHysics open-source code</b>	<b>39</b>
6.1 CPU source files	45
6.2 GPU source files	48
<b>7. Compiling DualSPHysics</b>	<b>51</b>
7.1 Windows compilation	51
7.2 Linux compilation	52
7.3 Cmake	54
<b>8. Format Files</b>	<b>57</b>
<b>9. Pre-processing</b>	<b>59</b>
<b>10. Processing</b>	<b>65</b>
<b>11. Post-processing</b>	<b>71</b>
11.1 Visualization of particle output data	71
11.2 Visualization of boundaries	76
11.3 Analysis of numerical measurements	79
11.4 Force computation	83
11.5 Analysis of floating data	85
11.6 Surface representation	87
<b>12. Testcases</b>	<b>91</b>
12.1 CaseDambreak	93
12.2 CasePeriodicity	97
12.3 CaseMovingSquare	98
12.4 CaseForces	99
12.5 CaseSloshing	100
12.6 CaseWavemaker	102
12.7 CaseWaveGeneration	106
12.8 CaseFloating	108
12.9 CasePump	113
12.10 CaseDEM	114
12.11 CaseMultiphase	116
<b>13. How to modify DualSPHysics for your application</b>	<b>117</b>
<b>14. FAQ: Frequently asked questions about DualSPHysics</b>	<b>119</b>
<b>15. New on DualSPHysics v4.0</b>	<b>125</b>
<b>16. DualSPHysics future</b>	<b>129</b>
<b>17. References</b>	<b>131</b>
<b>18. Licenses</b>	<b>137</b>



## 1. Introduction

Smoothed Particle Hydrodynamics is a Lagrangian meshless method that has been used in an expanding range of applications within the field of Computation Fluid Dynamics (CFD) [Gómez-Gesteira et al., 2010] where particles represent the flow, interact with structures, and exhibit large deformation with moving boundaries. The SPH model is approaching a mature stage for CFD with continuing improvements and modifications such that the accuracy, stability and reliability of the model are reaching an acceptable level for practical engineering applications.

The DualSPHysics code originates from SPHysics, which is an open-source SPH model developed by researchers at the Johns Hopkins University (US), the University of Vigo (Spain), the University of Manchester (UK) and the University of Rome, La Sapienza. The software is available to free download at [www.sphysics.org](http://www.sphysics.org). A complete guide of the FORTRAN code is found in [Gómez-Gesteira et al., 2012a; 2012b].

The SPHysics FORTRAN code was validated for different problems of wave breaking [Dalrymple and Rogers, 2006], dam-break behaviour [Crespo et al., 2008], interaction with coastal structures [Gómez-Gesteira and Dalrymple, 2004] or with a moving breakwater [Rogers et al., 2010].

Although SPHysics allows problems to be simulated using high resolution and a wide range of formulations, the main problem for its application to real engineering problems is the excessively long computational runtimes, meaning that SPHysics is rarely applied to large domains. Hardware acceleration and parallel computing are required to make SPHysics more useful and versatile for engineering application.

Originating from the computer games industry, Graphics Processing Units (GPUs) have now established themselves as a cheap alternative to High Performance Computing (HPC) for scientific computing and numerical modelling. GPUs are designed to manage huge amounts of data and their computing power has developed in recent years much faster than conventional central processing units (CPUs). Compute Unified Device Architecture (CUDA) is a parallel programming framework and language for GPU computing using some extensions to the C/C++ language. Researchers and engineers of different fields are achieving high speedups implementing their codes with the CUDA language. Thus, the parallel power computing of GPUs can be also applied for SPH methods where the same loops for each particle during the simulation can be parallelised.

The FORTRAN SPHysics code is robust and reliable but is not properly optimised for huge simulations. DualSPHysics is implemented in C++ and CUDA language to carry out simulations on either the CPU or GPU respectively. The new CPU code presents some advantages, such as more optimised use of the memory. The object-oriented programming paradigm provides a code that is easy to understand, maintain and modify with a sophisticated control of errors available. Furthermore, better optimisations are implemented, for example particles are reordered to give faster access to memory, and the best approach to create the neighbour list is implemented [Domínguez et al., 2011]. The CUDA language manages the parallel execution of threads on the GPUs. The best approaches were considered to be implemented as an extension of the C++ code, so the most appropriate optimizations to parallelise particle interaction on GPU were implemented [Domínguez et al., 2013a; 2013b]. The first rigorous validations were presented in [Crespo et al., 2011]. The version 3.0 of the code is fully documented in [Crespo et al., 2015].

Version 4 of the code has been developed to include the latest developments including coupling with the Discrete Element Method (DEM) and multi-phase developments as detailed in Section 3.

In the following sections we will describe the SPH formulation available in DualSPHysics, the implementation and optimization techniques, how to compile and run the different codes of the DualSPHysics package and future developments.

## **2. Developers and institutions**

Different countries and institutions collaborate in the development of DualSPHysics. The project is mainly led by the Environmental Physics Laboratory (EPHYSLAB) from Universidade de Vigo (Spain) and the School of Mechanical, Aerospace and Civil Engineering (MACE) from The University of Manchester (UK).

The following list includes the researchers that have collaborated in the current version of the code or are working in functionalities to be updated in future releases.

### **Developers:**

#### **Universidade de Vigo, Spain**

Dr José M. Domínguez  
Dr Alejandro J.C. Crespo  
Dr Anxo Barreiro  
Professor Moncho Gómez Gesteira

#### **EPHYTECH SL, Spain**

Orlando G. Feal

#### **Instituto Superior Técnico, Lisbon, Portugal**

Dr Ricardo Canelas

#### **Universiteit Gent - Flanders Hydraulics Research, Belgium**

Dr Corrado Altomare

#### **The University of Manchester, UK**

Dr Benedict D. Rogers  
Dr Georgios Fourtakas  
Dr Athanasios Mokos

#### **Science & Technology Facilities Council, UK**

Dr Stephen Longshaw

#### **Università degli studi di Parma, Italy**

Dr Renato Vacondio

### **Contributors:**

Imperial College London, UK. Mashy D Green

Universidad Politécnica de Madrid, Spain. Jose Luis Cercós Pita.

Universidad de Guanajuato, Mexico. Carlos Enrique Alvarado Rodríguez.



### 3. SPH formulation

First the **SPH formulation available** on the new DualSPHysics code is summarised.

Users are referred to the relevant publications below:

- Time integration scheme:
  - Verlet [[Verlet, 1967](#)].
  - Symplectic [[Leimkhuler, 1996](#)].
- Variable time step [[Monaghan and Kos, 1999](#)].
- Kernel functions:
  - Cubic Spline kernel [[Monaghan and Lattanzio, 1985](#)].
  - Quintic Wendland kernel [[Wendland, 1995](#)].
- Density treatment:
  - Delta-SPH formulation [[Molteni and Colagrossi, 2009](#)].
- Viscosity treatments:
  - Artificial viscosity [[Monaghan, 1992](#)].
  - Laminar viscosity + SPS turbulence model [[Dalrymple and Rogers, 2006](#)].
- Weakly compressible approach using Tait's equation of state.
- Shifting algorithm [[Lind et al., 2012](#)].
- Dynamic boundary conditions [[Crespo et al., 2007](#)].
- Floating objects [[Monaghan et al., 2003](#)].
- Periodic open boundaries [[Gómez-Gesteira et al., 2012a](#)]
- Coupling with Discrete Element Method [[Canelas et al., 2016](#)].
- External body forces [[Longshaw and Rogers, 2015](#)].
- Double precision [[Domínguez et al., 2013c](#)].
- Multi-phase (soil-water) [[Fourtakas and Rogers, 2016](#)] – executable only.

Features that will be integrated soon on the CPU-GPU solver as **future improvements**:

- Multi-GPU implementation [[Domínguez et al., 2013b](#)].
- Variable particle resolution [[Vacondio et al., 2013](#)].
- Multi-phase (gas-liquid) [[Mokos et al., 2015](#)].
- Inlet/outlet flow conditions.
- Local Uniform Stencil (LUST) boundary conditions [[Fourtakas et al., 2014](#)].
- Boundary Integral conditions [[Domínguez et al., 2015](#)].
- Active Wave Absorption System [[Altomare et al., 2015a](#)]
- Coupling with SWASH Wave Propagation Model [[Altomare et al., 2015b](#)].

Smoothed Particle Hydrodynamics (SPH) is a Lagrangian meshless method. The technique discretises a continuum using a set of material points or *particles*. When used for the simulation of fluid dynamics, the discretised Navier-Stokes equations are locally integrated at the location of each of these particles, according to the physical properties of surrounding particles. The set of neighbouring particles is determined by a distance based function, either circular (two-dimensional) or spherical (three-dimensional), with an associated characteristic length or *smoothing length* often denoted as  $h$ . At each time-step new physical quantities are calculated for each particle, and they then move according to the updated values.

The conservation laws of continuum fluid dynamics are transformed from their partial differential form to a form suitable for particle based simulation using integral equations based on an interpolation function, which gives an estimate of values at a specific point. Typically this interpolation or weighting function is referred to as the kernel function ( $W$ ) and can take different forms, with the most common being cubic or quintic. In all cases however, it is designed to represent a function  $F(\mathbf{r})$  defined in  $\mathbf{r}'$  by the integral approximation

$$F(\mathbf{r}) = \int F(\mathbf{r}') W(\mathbf{r} - \mathbf{r}', h) d\mathbf{r}' \quad (1)$$

The smoothing kernel must fulfil several properties [Monaghan, 1992; Liu, 2003], such as positivity inside a defined zone of interaction, compact support, normalization and monotonically decreasing value with distance and differentiability. For a more complete description of SPH, the reader is referred to [Monaghan, 2005; Violeau, 2013].

The function  $F$  in Eq. (1) can be approximated in a non-continuous, discrete form based on the set of particles. In this case the function is interpolated at a particle ( $a$ ) where a summation is performed over all the particles that fall within its region of compact support, as defined by the smoothing length  $h$

$$F(\mathbf{r}_a) \approx \sum_b F(\mathbf{r}_b) W(\mathbf{r}_a - \mathbf{r}_b, h) \Delta v_b \quad (2)$$

where the subscript denotes an individual particle,  $\Delta v_b$  is the volume of a neighbouring particle ( $b$ ). If  $\Delta v_b = m_b / \rho_b$ , with  $m$  and  $\rho$  being the mass and the density of particle  $b$  respectively then Eq. (2) becomes

$$F(\mathbf{r}_a) \approx \sum_b F(\mathbf{r}_b) \frac{m_b}{\rho_b} W(\mathbf{r}_a - \mathbf{r}_b, h) \quad (3)$$

### 3.1 The Smoothing Kernel

The performance of an SPH model depends heavily on the choice of the smoothing kernel. Kernels are expressed as a function of the non-dimensional distance between particles ( $q$ ), given by  $q = r/h$ , where  $r$  is the distance between any two given particles  $a$  and  $b$  and the parameter  $h$  (the smoothing length) controls the size of the area around

particle  $a$  in which neighbouring particles are considered. Within DualSPHysics, the user is able to choose from one of the following kernel definitions:

a) Cubic spline [Monaghan and Lattanzio, 1985]

$$W(r,h) = \alpha_D \begin{cases} 1 - \frac{3}{2}q^2 + \frac{3}{4}q^3 & 0 \leq q \leq 1 \\ \frac{1}{4}(2-q)^3 & 1 \leq q \leq 2 \\ 0 & q \geq 2 \end{cases} \quad (4)$$

where  $\alpha_D$  is equal to  $10/7\pi h^2$  in 2-D and  $1/\pi h^3$  in 3-D.

The tensile correction method, proposed by [Monaghan, 2000], is only actively used in the cases of a kernel whose first derivative goes to zero with the particle distance  $q$ .

b) Quintic [Wendland, 1995]

$$W(r,h) = \alpha_D \left(1 - \frac{q}{2}\right)^4 (2q+1) \quad 0 \leq q \leq 2 \quad (5)$$

where  $\alpha_D$  is equal to  $7/4\pi h^2$  in 2-D and  $21/16\pi h^3$  in 3-D.

In the text that follows, only kernels with an influence domain of  $2h$  ( $q \leq 2$ ) are considered.

### 3.2. Momentum Equation

The momentum conservation equation in a continuum is

$$\frac{d\mathbf{v}}{dt} = -\frac{1}{\rho} \nabla P + \mathbf{g} + \boldsymbol{\Gamma} \quad (6)$$

where  $\boldsymbol{\Gamma}$  refers to dissipative terms and  $\mathbf{g}$  is gravitational acceleration. DualSPHysics offers different options for including the effects of dissipation.

#### 3.2.1. Artificial Viscosity

The artificial viscosity scheme, proposed by [Monaghan, 1992], is a common method within fluid simulation using SPH due primarily to its simplicity. In SPH notation, Eq. 6 can be written as

$$\frac{d\mathbf{v}_a}{dt} = -\sum_b m_b \left( \frac{P_b + P_a}{\rho_b \cdot \rho_a} + \Pi_{ab} \right) \nabla_a W_{ab} + \mathbf{g} \quad (7)$$

where  $P_k$  and  $\rho_k$  are the pressure and density that correspond to particle  $k$  (as evaluated at  $a$  or  $b$ ). The viscosity term  $\Pi_{ab}$  is given by

$$\Pi_{ab} = \begin{cases} -\alpha \overline{c}_{ab} \mu_{ab} & \mathbf{v}_{ab} \cdot \mathbf{r}_{ab} < 0 \\ 0 & \mathbf{v}_{ab} \cdot \mathbf{r}_{ab} > 0 \end{cases} \quad (8)$$

where  $\mathbf{r}_{ab} = \mathbf{r}_a - \mathbf{r}_b$  and  $\mathbf{v}_{ab} = \mathbf{v}_a - \mathbf{v}_b$  with  $\mathbf{r}_k$  and  $\mathbf{v}_k$  being the particle position and velocity respectively.  $\mu_{ab} = h\mathbf{v}_{ab} \cdot \mathbf{r}_{ab} / (r_{ab}^2 + \eta^2)$ ,  $\bar{c}_{ab} = 0.5(c_a + c_b)$  is the mean speed of sound,  $\eta^2 = 0.01h^2$  and  $\alpha$  is a coefficient that needs to be tuned in order to introduce the proper dissipation. The value of  $\alpha=0.01$  has proven to give the best results in the validation of wave flumes to study wave propagation and wave loadings exerted onto coastal structures [Altomare et al., 2015a; 2015c].

### 3.2.2. Laminar viscosity and Sub-Particle Scale (SPS) Turbulence

Laminar viscous stresses in the momentum equation can be expressed as [Lo and Shao, 2002]

$$(v_0 \nabla^2 \mathbf{v})_a = \sum_b m_b \left( \frac{4v_0 \mathbf{r}_{ab} \cdot \nabla_a W_{ab}}{(\rho_a + \rho_b)(r_{ab}^2 + \eta^2)} \right) \mathbf{v}_{ab} \quad (9)$$

where  $v_0$  is kinematic viscosity (typically  $10^{-6}$  m<sup>2</sup>s for water). In SPH discrete notation this can be expressed as

$$\frac{d\mathbf{v}_a}{dt} = -\sum_b m_b \left( \frac{P_b + P_a}{\rho_b \cdot \rho_a} \right) \nabla_a W_{ab} + \mathbf{g} + \sum_b m_b \left( \frac{4v_0 \mathbf{r}_{ab} \cdot \nabla_a W_{ab}}{(\rho_a + \rho_b)(r_{ab}^2 + \eta^2)} \right) \mathbf{v}_{ab} \quad (10)$$

The concept of the Sub-Particle Scale (SPS) was first described by [Gotoh et al., 2001] to represent the effects of turbulence in their Moving Particle Semi-implicit (MPS) model. The momentum conservation equation is defined as

$$\frac{d\mathbf{v}}{dt} = -\frac{1}{\rho} \nabla P + \mathbf{g} + v_0 \nabla^2 \mathbf{v} + \frac{1}{\rho} \nabla \cdot \vec{\tau} \quad (11)$$

where the laminar term is treated as per Eq. 9 and  $\vec{\tau}$  represents the SPS stress tensor. Favre-averaging is needed to account for compressibility in weakly compressible SPH [Dalrymple and Rogers, 2006] where eddy viscosity assumption is used to model the SPS stress tensor with Einstein notation for the shear stress component in coordinate directions  $i$  and  $j$   $\frac{\vec{\tau}_{ij}}{\rho} = v_t \left( 2S_{ij} - \frac{2}{3} k \delta_{ij} \right) - \frac{2}{3} C_s \Delta l^2 |S| S_{ij}$ , where  $\vec{\tau}_{ij}$  is the sub-particle stress tensor,  $v_t = [(C_s \Delta l)^2 |S|]$  the turbulent eddy viscosity,  $k$  the SPS turbulence kinetic energy,  $C_s$  the Smagorinsky constant (0.12),  $C_s=0.0066$ ,  $\Delta l$  the particle to particle spacing and  $|S|=0.5(2S_{ij}S_{ij})$  where  $S_{ij}$  is an element of the SPS strain tensor. [Dalrymple and Rogers, 2006] introduced SPS into weakly compressible SPH using Favre averaging, Eq.11 can be re-written as

$$\begin{aligned}
\frac{d\mathbf{v}_a}{dt} = & - \sum_b m_b \left( \frac{P_b + P_a}{\rho_b \cdot \rho_a} \right) \nabla_a W_{ab} + \mathbf{g} \\
& + \sum_b m_b \left( \frac{4v_0 r_{ab} \cdot \nabla_a W_{ab}}{(\rho_a + \rho_b)(r_{ab}^2 + \eta^2)} \right) \mathbf{v}_{ab} + \\
& + \sum_b m_b \left( \frac{\bar{\tau}_{ij}^b}{\rho_b^2} + \frac{\bar{\tau}_{ij}^a}{\rho_a^2} \right) \nabla_a W_{ab}
\end{aligned} \tag{12}$$

where the superscripts refer to particles  $a$  and  $b$ .

### 3.3. Continuity Equation

Throughout the duration of a weakly-compressible SPH simulation (as presented herein) the mass of each particle remains constant and only their associated density fluctuates. These density changes are computed by solving the conservation of mass, or continuity equation, in SPH form:

$$\frac{d\rho_a}{dt} = \sum_b m_b \mathbf{v}_{ab} \cdot \nabla_a W_{ab} \tag{13}$$

### 3.4. Equation of State

Following the work of [Monaghan, 1994], the fluid in the SPH formalism defined in DualSPHysics is treated as weakly compressible and an equation of state is used to determine fluid pressure based on particle density. The compressibility is adjusted so that the speed of sound can be artificially lowered; this means that the size of time step taken at any one moment (which is determined according to a Courant condition, based on the currently calculated speed of sound for all particles) can be maintained at a reasonable value. Such adjustment however, restricts the sound speed to be at least ten times faster than the maximum fluid velocity, keeping density variations to within less than 1%, and therefore not introducing major deviations from an incompressible approach. Following [Monaghan et al., 1999] and [Batchelor, 1974], the relationship between pressure and density follows the expression

$$P = b \left[ \left( \frac{\rho}{\rho_0} \right)^\gamma - 1 \right] \tag{14}$$

where  $\gamma = 7$ ,  $b = c_o^2 \rho_0 / \gamma$  where  $\rho_0 = 1000 \text{ kg/m}^3$  is the reference density and  $c_o = c(\rho_o) = \sqrt{(\partial P / \partial \rho)}|_{\rho_o}$  which is the speed of sound at the reference density.

### 3.5. DeltaSPH

Within DualSPHysics it is also possible to apply a *delta-SPH* formulation, that introduces a diffusive term to reduce density fluctuations. The state equation describes a very stiff density field, and together with the natural disordering of the Lagrangian particles, high-frequency low amplitude oscillations are found to populate the density scalar field [Molteni and Colagrossi, 2009]. DualSPHysics uses a diffusive term in the continuity equation, now written as

$$\frac{d\rho_a}{dt} = \sum_b m_b \mathbf{v}_{ab} \cdot \nabla_a W_{ab} + 2\delta_\Phi h c_0 \sum_b (\rho_b - \rho_a) \frac{\mathbf{r}_{ab} \cdot \nabla_a W_{ab}}{\mathbf{r}_{ab}^2} \frac{m_b}{\rho_b} \quad (15)$$

This represents the original delta-SPH formulation by [Molteni and Colagrossi, 2009], with the free parameter  $\delta_\Phi$  that needs to be attributed a suitable value. This modification can be explained as the addition of the Laplacian of the density field to the continuity equation. [Antuono et al., 2012] has presented a careful analysis of the influence of this term in the system, by decomposing the Laplacian operator, observing the converge of the operators and performing linear stability analysis to inspect the influence of the diffusive coefficient. This equation represents exactly a diffusive term in the domain bulk. The behaviour changes close to open boundaries such as free-surface. Due to truncation of the kernel (there are no particles being sampled outside of an open boundary), the first-order contributions are not null [Antuono et al., 2010], resulting in a net force applied to the particles. This effect is not considered relevant for non-hydrostatic situations, where this force is many orders of magnitude inferior to any other force involved. Corrections to this effect were proposed by [Antuono et al., 2010], but involve the solution of a renormalization problem for the density gradient, with considerable computational cost. A delta-SPH ( $\delta_\Phi$ ) coefficient of 0.1 is recommended for most applications.

### 3.6. Shifting algorithm

Anisotropic particle spacing is an important stability issue in SPH as, especially in violent flows, particles cannot maintain a uniform distribution. The result is the introduction of noise in the velocity and pressure field, as well as the creation of voids within the water flow for certain cases.

To counter the anisotropic particle spacing, [Xu et al., 2009] proposed a particle shifting algorithm to prevent the instabilities. The algorithm was first created for incompressible SPH, but can be extended to the weakly compressible SPH model used in DualSPHysics [Vacondio et al., 2013]. With the shifting algorithm, the particles are moved (“shifted”) towards areas with fewer particles (lower particle concentration) allowing the domain to maintain a uniform particle distribution and eliminating any voids that may occur due to the noise.

An improvement on the initial shifting algorithm was proposed by [Lind et al., 2012] who used Fick's first law of diffusion to control the shifting magnitude and direction. Fick's first law connects the diffusion flux to the concentration gradient:

$$\mathbf{J} = -D_F \nabla C \quad (16)$$

where  $\mathbf{J}$  is the flux,  $C$  the particle concentration, and  $D_F$  the Fickian diffusion coefficient.

Assuming that the flux, i.e. the number of particles passing through a unit surface in unit time, is proportional to the velocity of the particles, a particle shifting velocity and subsequently a particle shifting distance can be found. Using the particle concentration, the particle shifting distance  $\delta\mathbf{r}_s$  is given by:

$$\delta\mathbf{r}_s = -D \nabla C_i \quad (17)$$

where  $D$  is a new diffusion coefficient that controls the shifting magnitude and absorbs the constants of proportionality. The gradient of the particle concentration can be found through an SPH gradient operator:

$$\nabla C_i = \sum_j \frac{m_j}{\rho_j} \nabla W_{ij} \quad (18)$$

The proportionality coefficient  $D$  is computed through a form proposed by [Skillen et al., 2013]. It is set to be large enough to provide effective particle shifting, while not introducing significant errors or instabilities. This is achieved by performing a Von Neumann stability analysis of the advection-diffusion equation:

$$D \leq \frac{1}{2} \frac{h^2}{\Delta t_{\max}} \quad (19)$$

where  $\Delta t_{\max}$  is the maximum local time step that is permitted by the CFL condition for a given local velocity and particle spacing. The CFL condition states that:

$$\Delta t_{\max} \leq \frac{h}{\|\mathbf{u}\|_i} \quad (20)$$

Combining Eq. 19 and 20 we can derive an equation to find the shifting coefficient  $D$ :

$$D = Ah \|\mathbf{u}\|_i \quad (21)$$

where  $A$  is a dimensionless constant that is independent of the problem setup and discretisation. Values in the range of [1,6] are proposed with 2 used as default.

The shifting algorithm is heavily dependent on a full kernel support. However, particles at and adjacent to the free surface cannot obtain the full kernel support, which will introduce errors in the free-surface prediction, potentially causing non-physical instabilities. Applying Fick's law directly would result in the rapid diffusion of fluid particles from the fluid bulk, due to the large concentration gradients at the free surface.

To counter this effect, [Lind et al., 2012] proposed a free-surface correction that limits diffusion to the surface normal but allow shifting on the tangent to the free surface.

Therefore, this correction is only used near the free surface, identified by the value of the particle divergence, which is computed through the following equation, first proposed by [Lee et al., 2008]:

$$\nabla \cdot \mathbf{r} = \sum_j \frac{m_j}{\rho_j} \mathbf{r}_{ij} \cdot \nabla_i W_{ij} \quad (22)$$

This idea is applied to the DualSPHysics code by multiplying the shifting distance of Equation (17) with a free-surface correction coefficient  $A_{FSC}$ .

$$A_{FSC} = \frac{\nabla \cdot \mathbf{r} - A_{FST}}{A_{FSM} - A_{FST}} \quad (23)$$

where  $A_{FST}$  is the free-surface threshold and  $A_{FSM}$  is the maximum value of the particle divergence. The latter depends on the domain dimensions:

$$A_{FSM} = \begin{cases} 2 \text{ for 2D} \\ 3 \text{ for 3D} \end{cases}$$

while the free surface threshold is selected for DualSPHysics as:

$$A_{FST} = \begin{cases} 1.5 \text{ for 2D} \\ 2.75 \text{ for 3D} \end{cases}$$

To identify the position of the particle relative to the free surface, the difference of the particle divergence to  $A_{FST}$  is used. Therefore, the full shifting equation (Eq. 17) with the free surface correction is:

$$\delta\mathbf{r}_s = \begin{cases} -A_{FSC} Ah \|\mathbf{u}\|_i \nabla C_i & \text{if } (\nabla \cdot \mathbf{r} - A_{FST}) < 0 \\ -Ah \|\mathbf{u}\|_i \nabla C_i & \text{if } (\nabla \cdot \mathbf{r} - A_{FST}) = 0 \end{cases} \quad (24)$$

More information about the shifting implementation can be found in [Mokos, 2013].

### 3.7. Time stepping

DualSPHysics includes a choice of numerical integration schemes, if the momentum ( $\mathbf{v}_a$ ), density ( $\rho_a$ ) and position ( $\mathbf{r}_a$ ) equations are first written in the form

$$\frac{d\mathbf{v}_a}{dt} = \mathbf{F}_a \quad (25a)$$

$$\frac{d\rho_a}{dt} = D_a \quad (25b)$$

$$\frac{d\mathbf{r}_a}{dt} = \mathbf{v}_a \quad (25c)$$

These equations are integrated in time using a computationally simple Verlet based scheme or a more numerically stable but computationally intensive two-stage Symplectic method.

### 3.7.1. Verlet Scheme

This algorithm, which is based on the common Verlet method [Verlet, 1967] is split into two parts and benefits from providing a low computational overhead compared to some other integration techniques, primarily as it does not require multiple (i.e. predictor and corrector) calculations for each step. The predictor step calculates the variables according to

$$\mathbf{v}_a^{n+1} = \mathbf{v}_a^{n-1} + 2\Delta t \mathbf{F}_a^n; \quad \mathbf{r}_a^{n+1} = \mathbf{r}_a^n + \Delta t \mathbf{V}_a^n + 0.5\Delta t^2 \mathbf{F}_a^n; \quad \rho_a^{n+1} = \rho_a^{n-1} + 2\Delta t D_a^n \quad (26)$$

where the superscript  $n$  denotes the time step,  $\mathbf{F}_a^n$  and  $D_a^n$  are calculated using Eq. 7 (or Eq. 12) and Eq. 13 (or Eq. 14) respectively.

However, once every  $N_s$  time steps (where  $N_s \approx 50$  is suggested), variables are calculated according to

$$\mathbf{v}_a^{n+1} = \mathbf{v}_a^n + \Delta t \mathbf{F}_a^n; \quad \mathbf{r}_a^{n+1} = \mathbf{r}_a^n + \Delta t \mathbf{V}_a^n + 0.5\Delta t^2 \mathbf{F}_a^n; \quad \rho_a^{n+1} = \rho_a^n + \Delta t D_a^n \quad (27)$$

This second part is designed to stop divergence of integrated values through time as the equations are no longer coupled. In cases where the Verlet scheme is used but it is found that numerical stability is an issue, it may be sensible to increase the frequency at which the second part of this scheme is applied, however if it should be necessary to increase this frequency beyond  $N_s = 10$  then this could indicate that the scheme is not able to capture the dynamics of the case in hand suitably and the Symplectic scheme should be used instead.

### 3.7.2. Symplectic Scheme

Symplectic integration algorithms are time reversible in the absence of friction or viscous effects [Leimkuhler, 1996]. They can also preserve geometric features, such as the energy time-reversal symmetry present in the equations of motion, leading to improved resolution of long term solution behaviour. The scheme used here is an explicit second-order Symplectic scheme with an accuracy in time of  $O(\Delta t^2)$  and involves a predictor and corrector stage.

During the predictor stage the values of acceleration and density are estimated at the middle of the time step according to

$$\mathbf{r}_a^{\frac{n+1}{2}} = \mathbf{r}_a^n + \frac{\Delta t}{2} \mathbf{v}_a^n; \quad \rho_a^{\frac{n+1}{2}} = \rho_a^n + \frac{\Delta t}{2} D_a^n. \quad (28)$$

During the corrector stage  $d\mathbf{v}_a^{\frac{n+1}{2}}/dt$  is used to calculate the corrected velocity, and therefore position, of the particles at the end of the time step according to

$$\mathbf{v}_a^{n+1} = \mathbf{v}_a^{\frac{n+1}{2}} + \frac{\Delta t}{2} \mathbf{F}_a^{\frac{n+1}{2}}, \quad (29)$$

$$\mathbf{r}_a^{n+1} = \mathbf{r}_a^{\frac{n+1}{2}} + \frac{\Delta t}{2} \mathbf{v}_a^{n+1}.$$

and finally the corrected value of density  $d\rho_a^{n+1}/dt = D_a^{n+1}$  is calculated using the updated values of  $\mathbf{v}_a^{n+1}$  and  $\mathbf{r}_a^{n+1}$  [Monaghan, 2005].

### 3.7.3. Variable Time Step

With explicit time integration schemes the timestep is dependent on the Courant-Friedrichs-Lowy (CFL) condition, the forcing terms and the viscous diffusion term. A variable time step  $\Delta t$  is calculated according to [Monaghan et al., 1999] using

$$\begin{aligned}\Delta t_f &= CFL \cdot \min(\Delta t_f, \Delta t_{cv}) \\ \Delta t_f &= \min\left(\sqrt{h/|\mathbf{f}_a|}\right)_a \\ \Delta t_{cv} &= \min_a \frac{h}{c_s + \max_b \left| \frac{h \mathbf{v}_{ab} \cdot \mathbf{r}_{ab}}{(r_{ab}^2 + \eta^2)} \right|}\end{aligned}\tag{30}$$

where  $\Delta t_f$  is based on the force per unit mass ( $|\mathbf{f}_a|$ ), and  $\Delta t_{cv}$  combines the Courant and the viscous time step controls.

## 3.8. Boundary conditions

In DualSPHysics, the boundary is described by a set of particles that are considered as a separate set to the fluid particles. The software currently provides functionality for solid impermeable and periodic open boundaries. Methods to allow boundary particles to be moved according to fixed forcing functions are also present.

### 3.8.1. Dynamic Boundary Condition

The Dynamic Boundary Condition (DBC) is the default method provided by DualSPHysics [Crespo et al., 2007]. This method sees boundary particles that satisfy the same equations as fluid particles, however they do not move according to the forces exerted on them. Instead, they remain either fixed in position or move according to an imposed/assigned motion function (i.e. moving objects such as gates, wave-makers or floating objects).

When a fluid particle approaches a boundary and the distance between its particles and the fluid particle becomes smaller than twice the smoothing length ( $h$ ), the density of the affected boundary particles increases, resulting in a pressure increase. In turn this results in a repulsive force being exerted on the fluid particle due to the pressure term in the momentum equation.

Stability of this method relies on the length of time step taken being suitably short in order to handle the highest present velocity of any fluid particles currently interacting

with boundary particles and is therefore an important point when considering how the variable time step is calculated.

Different boundary conditions have been tested in DualSPHysics in the work of [Domínguez et al., 2015]: Dynamic Boundary Condition (DBC), Local Uniform STencil (LUST) and Boundary Integral (INTEGRAL). Validations with dam-break flows and sloshing tanks highlighted the advantages and drawbacks of each method.

### 3.8.2. Periodic Open Boundary Condition

DualSPHysics provides support for open boundaries in the form of a periodic boundary condition. This is achieved by allowing particles that are near an open lateral boundary to interact with the fluid particles near the complimentary open lateral boundary on the other side of the domain.

In effect, the compact support kernel of a particle is clipped by the nearest open boundary and the remainder of its clipped support applied at the complimentary open boundary [Gómez-Gesteira et al., 2012a].

### 3.8.3. Pre-imposed Boundary Motion

Within DualSPHysics it is possible to define a pre-imposed movement for a set of boundary particles. Various predefined movement functions are available as well as the ability to assign a time-dependant input file containing kinematic detail.

These boundary particles behave as a DBC described in Section 3.8.1, however rather than being fixed, they move independently of the forces currently acting upon them. This provides the ability to define complex simulation scenarios (i.e. a wave-making paddle) as the boundaries influence the fluid particles appropriately as they move.

### 3.8.4. Fluid-driven Objects

It is also possible to derive the movement of an object by considering its interaction with fluid particles and using these forces to drive its motion. This can be achieved by summing the force contributions for an entire body. By assuming that the body is rigid, the net force on each boundary particle is computed according to the sum of the contributions of all surrounding fluid particles according to the designated kernel function and smoothing length. Each boundary particle  $k$  therefore experiences a force per unit mass given by

$$\mathbf{f}_k = \sum_{a \in WPs} \mathbf{f}_{ka} \quad (31)$$

where  $\mathbf{f}_{ka}$  is the force per unit mass exerted by the fluid particle  $a$  on the boundary particle  $k$ , which is given by

$$m_k \mathbf{f}_{ka} = -m_a \mathbf{f}_{ak} \quad (32)$$

For the motion of the moving body, the basic equations of rigid body dynamics can then be used

$$M \frac{d\mathbf{V}}{dt} = \sum_{k \in BPs} m_k \mathbf{f}_k \quad (33a)$$

$$I \frac{d\boldsymbol{\Omega}}{dt} = \sum_{k \in BPs} m_k (\mathbf{r}_k - \mathbf{R}_o) \times \mathbf{f}_k \quad (33b)$$

where  $M$  is the mass of the object,  $I$  the moment of inertia,  $\mathbf{V}$  the velocity,  $\boldsymbol{\Omega}$  the rotational velocity and  $\mathbf{R}_o$  the centre of mass. Equations 33a and 33b are integrated in time in order to predict the values of  $\mathbf{V}$  and  $\boldsymbol{\Omega}$  for the beginning of the next time step. Each boundary particle within the body then has a velocity given by

$$\mathbf{u}_k = \mathbf{V} + \boldsymbol{\Omega} \times (\mathbf{r}_k - \mathbf{R}_o) \quad (34)$$

Finally, the boundary particles within the rigid body are moved by integrating Eq. 34 in time. The works of [Monaghan et al., 2003] and [Monaghan, 2005] show that this technique conserves both linear and angular momentum. [Bouscasse et al., 2013] presented successful validations of nonlinear water wave interaction with floating bodies in SPH comparing with experimental data from [Hadžić et al., 2005] that includes deformations in the free-surface due to the presence of floating boxes and the movement of those objects during the experiment (heave, surge and roll displacements). Several validations using DualSPHysics are performed in [Canelas et al., 2015] that analyse the buoyancy-driven motion with solid objects larger than the smallest flow scales and with various densities. They compared SPH numerical results with analytical solutions, with other numerical methods [Fekken, 2004] and with experimental measurements.

### 3.9. Wave generation

Wave generation is included in this version of DualSPHysics, for long-crested waves only. In this way, the numerical model can be used to simulate a physical wave flume. Both regular and random waves can be generated. The following sections refer only to the piston-type wavemaker.

#### 3.9.1. First order wave generation

The Biesel transfer functions express the relation between wave amplitude and wavemaker displacement [Biesel and Suquet, 1951], under the assumption of irrotational and incompressible fluid and constant pressure at the free surface. The transfer function links the displacement of the piston-type wavemaker to the water surface elevation, under the hypothesis of monochromatic sinusoidal waves in one dimension in the  $x$ -direction:

$$\eta(x, t) = \frac{H}{2} \cos(\omega t - kx + \delta) \quad (35)$$

where  $H$  is the wave height,  $d$  the water depth,  $x$  is distance and  $\delta$  is the initial phase. The quantity  $\omega=2\pi/T$  is the angular frequency and  $k=2\pi/L$  is the wave number with  $T$  equal to the wave period and  $L$  the wave length. The initial phase  $\delta$  is given by a random number between 0 and  $2\pi$ .

Eq. 35 expresses the surface elevation at infinity that Biesel defined as the far-field solution. The Biesel function can be derived for the far-field solution and for a piston-type wavemaker as:

$$\frac{H}{S_0} = \frac{2 \sinh^2(kd)}{\sinh(kd) \cosh(kd) + kd} \quad (36)$$

where  $S_0$  is the piston stroke. Once the piston stroke is defined, the time series of the piston movement is given by:

$$e_i(t) = \frac{S_0}{2} \sin(\omega t + \delta) \quad (37)$$

### 3.9.2. Second order wave generation

The implementation of a second order wavemaker theory will prevent the generation of spurious secondary waves. The second order wave generation theory implemented in DualSPHysics is based on [Madsen, 1971] who developed a simple second-order wavemaker theory to generate long second order Stokes waves that would not change shape as they propagated. The theory proposed by [Madsen, 1971] is straightforward, controllable, computationally inexpensive with efficient results, and is accurate for waves of first and second order.

The piston stroke  $S_0$  can be redefined from Eq. 36 as  $S_0=H/m_1$  where:

$$m_1 = \frac{2 \sinh^2(kd)}{\sinh(kd) \cosh(kd) + kd} \quad (38)$$

Following [Madsen, 1971], to generate a wave of second order, an extra term must be added to Eq. 37. This term, for piston-type wavemaker, is equal to:

$$e_2(t) = \left[ \left( \frac{H^2}{32d} \right) \cdot \left( \frac{3 \cosh(kd)}{\sinh^3(kd)} \right) - \frac{2}{m_1} \right] \sin(2\omega t + 2\delta) \quad (39)$$

Therefore, the piston displacement, for regular waves, is the summation of Eq. 37 and Eq. 39:

$$e(t) = \frac{S_0}{2} \sin(\omega t + \delta) + \left[ \left( \frac{H^2}{32d} \right) \cdot \left( \frac{3 \cosh(kd)}{\sinh^3(kd)} \right) - \frac{2}{m_1} \right] \sin(2\omega t + 2\delta) \quad (40)$$

Madsen limited the application of this approximate second order wavemaker theory to waves that complied with the condition given by  $HL^2/d^3 < 8\pi^2/3$ . A specific warning is implemented in DualSPHysics to inform the user whether or not this condition is fulfilled.

### 3.9.3. First order wave generation of irregular waves

Monochromatic waves are not representative of sea states that characterise real wave storm conditions. Sea waves are mostly random or irregular in nature. Irregular wave generation is performed in DualSPHysics based on [Liu and Frigaard, 2001]. Starting

from an assigned wave spectra, the Biesel transfer function (Eq. 36) is applied to each component in which the spectrum is discretised. The procedure for the generation of irregular waves is summarised as follows:

1. Defining the wave spectrum through its characteristic parameters (peak frequency, spectrum shape, etc.).
2. Dividing the spectrum into  $N$  parts ( $N > 50$ ) in the interval  $(f_{start}, f_{stop})$ , where generally the values assumed by the spectrum ( $S_\eta$ ) at the extremes of this interval are smaller than the value assumed for the peak frequency,  $f_p$ :  $S_\eta(f_{start}) \leq 0.01 \cdot S_\eta(f_p)$  and  $S_\eta(f_{stop}) \leq 0.01 \cdot S_\eta(f_p)$ .
3. The frequency band width is so-defined as  $\Delta f = (f_{stop} - f_{start})/N$ . The irregular wave is decomposed into  $N$  linear waves.
4. Determining the angular frequency  $\omega_i$ , amplitude  $a_i$  and initial phase  $\delta_i$  (random number between 0 and  $2\pi$ ) of each i-th linear wave. The angular frequency  $\omega_i$  and amplitude  $a_i$  can therefore be expressed as follows:

$$\omega_i = 2\pi f_i \quad (41)$$

$$a_i = \sqrt{2S_\eta(f_i)\Delta f} = \frac{H_i}{2} \quad (42)$$

5. Converting the time series of surface elevation into the time series of piston movement with the help of Biesel transfer function:

$$\frac{H_i}{S_{0,i}} = \frac{2 \sinh^2(k_i d)}{\sinh(k_i d) \cosh(k_i d) + k_i d} \quad (43)$$

6. Composing all the i-th components derived from the previous equation into the time series of the piston displacement as:

$$e(t) = \sum_{i=1}^N \frac{S_{0,i}}{2} \sin(\omega_i t + \delta_i) \quad (44)$$

In DualSPHysics two standard wave spectra have been implemented and used to generate irregular waves: JONSWAP and Pierson-Moskowitz spectra. The characteristic parameters of each spectrum can be assigned by the user together with the value of  $N$  (number of parts in which the spectrum is divided).

The user can choose among four different ways to define the angular frequency. It can be determined assuming an equidistant discretization of the wave spectrum ( $f_i = f_{start} + i\Delta f - \Delta f/2$ ), or chosen as unevenly distributed between  $(i-0.5)\Delta f$  and  $(i+0.5)\Delta f$ . An unevenly distributed band width should be preferred: in fact, depending on  $N$ , an equidistant splitting can lead to the repetition of the same wave group in the time series that can be easily avoided using an unevenly distributed band width. The last two ways to determine the angular frequency of each component and its band width consist of the application of a stretched or cosine stretched function. The use of a stretched or cosine stretched function has been proven to lead the most accurate results in terms of wave height distribution and groupiness, even when the number of spectrum components  $N$ , is relatively low. If there is a certain wave group that is repeating, finally the full range of wave heights and wave periods is not reproduced and statistically the wave train is not representing a real sea state of random waves.

A phase seed is also used and can be changed in DualSPHysics to obtain different random series of  $\delta_i$ . Changing the phase seed allows generating different irregular wave time series both with the same significant wave height ( $H_{m0}$ ) and peak period ( $T_p$ ).

### 3.10. Coupling with Discrete Element Method (DEM)

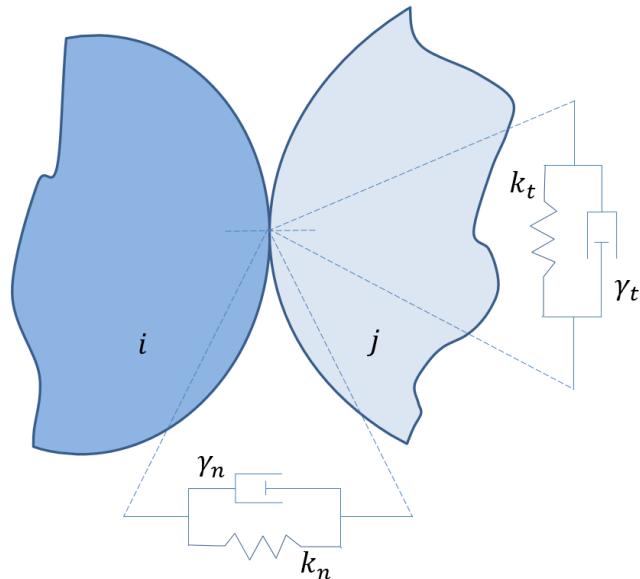
The discrete element method (DEM) allows for the computation of rigid particle dynamics, by considering contact laws to account for interaction forces. The coupled numerical solution, based on SPH and DEM discretisations, resolves solid-solid and solid-fluid interactions over a broad range of scales.

Forces arise whenever a particle of a solid object interacts with another. In the particular case of a solid-solid collision, the contact force is decomposed into  $\mathbf{F}_n$  and  $\mathbf{F}_t$ , normal and tangential components respectively. Both of these forces include viscous dissipation effects. This is because two colliding bodies undergo a deformation which will be somewhere between perfectly inelastic and perfectly elastic, usually quantified by the normal restitution coefficient

$$e_n = -\frac{\left.v_n\right|_{t=t^n} - \left.v_n\right|_{t=0}}{\left.v_n\right|_{t=0}}, e \in [0,1] \quad (45)$$

The total forces are decomposed into a repulsion force,  $\mathbf{F}_r$ , arising from the elastic deformation of the material, and a damping force,  $\mathbf{F}_d$ , for the viscous dissipation of energy during the deformation.

Figure 3-1 generally illustrates the proposed viscoelastic DEM mechanism between two interacting particles.



**Figure 3-1.** Schematic interaction between particles with viscoelastic DEM mechanism.

The normal force is given by

$$\mathbf{F}_{n,ij} = \mathbf{F}_n^r + \mathbf{F}_n^d = k_{n,ij} \delta_{ij}^{3/2} \mathbf{e}_{ij}^n - \gamma_{n,ij} \delta_{ij}^{1/2} \dot{\delta}_{ij} \mathbf{e}_{ij}^n \quad (46)$$

where the stiffness is given by

$$k_{n,ij} = \frac{4}{3} E^* \sqrt{R^*} \quad (47)$$

and the damping coefficient is

$$\gamma_{n,ij} = -\frac{\log e_{ij}}{\sqrt{\pi^2 + \log^2 e_{ij}}} \quad (48)$$

where  $\mathbf{e}_{ij}^n$  is the unit vector between the centers of particles  $i$  and  $j$ .

The restitution coefficient  $e_{ij}$  is taken as the average of the two materials coefficients, in what is the only calibration parameter of the model. It is not a physical parameter, but since the current method does not account for internal deformations and other energy losses during contact, the user is given a choice to change this parameter freely in order to control the dissipation of each contact. The reduced radius and reduced elasticity are given by

$$R^* = \left( \frac{1}{R_1} + \frac{1}{R_2} \right)^{-1}; E^* = \left( \frac{1 - v_p^2}{E_1} + \frac{1 - v_p^2}{E_2} \right)^{-1} \quad (49)$$

where  $R_i$  is simply the particle radius,  $E_i$  is the Young modulus and  $v_p$  is the Poisson coefficient of material  $i$ , as specified in the *Floating\_Materials.xml*.

This results in another restriction to the time-step, adding

$$\Delta t_{c,ij} = \frac{3.21}{50} \left( -\frac{M^*}{k_{n,if}} \right)^{2/5} v_{n,ij}^{-1/5} \quad (50)$$

to the existing CFL restrictions (Eq. 30), where  $v_n$  is the normal relative velocity and  $M^*$  is the reduced mass of the system where there is a contact.

Regarding tangential contacts, friction is modelled using the same model:

$$\mathbf{F}_{t,ij} = \mathbf{F}_t^r + \mathbf{F}_t^d = k_{t,ij} \delta_{ij}' \mathbf{e}_{ij}^t - \gamma_{t,ij} \delta_{ij}' \dot{\delta}_{ij} \mathbf{e}_{ij}^t \quad (51)$$

where the stiffness and damping constants are derived to be

$$k_{t,ij} = 2/7 k_{n,ij}; \gamma_{t,ij} = 2/7 \gamma_{n,ij} \quad (52)$$

as to insure internal consistency of the time scales between normal and tangential components. This mechanism models the static and dynamic friction mechanisms by a penalty method. The body does not statically stick at the point of contact, but is constrained by the spring-damper system. This force must be bounded above by the Coulomb friction law, modified with a sigmoidal function in order to make it continuous around the origin regarding the tangential velocity:

$$\mathbf{F}_{t,ij} = \min(\mu_{II} \mathbf{F}_{n,ij} \tanh(8\delta_{ij}') \mathbf{e}_{ij}', \mathbf{F}_{t,ij}) \quad (53)$$

where  $\mu_{IJ}$  is the friction coefficient at the contact of object  $I$  and object  $J$  and is simply taken as the average of the two friction coefficients of the distinct materials, indicated in the *Floating\_Materials.xml*.

More information about DEM implementation can be found in [[Canelas, 2015; Canelas et al., 2016](#)].

### **3.11. Multi-phase: Two-phase liquid-sediment implementation in DualSPHysics**

This guide provides a concise depiction of the multi-phase liquid-sediment model implemented in DualSPHysics solver. The model is capable of simulating problems involving liquid-sediment phases with the addition of highly non-linear deformations and free-surface flows which are frequently encountered in applied hydrodynamics. More specifically, the two-phase liquid-solid model is aimed at flow-induced erosion of fully saturated sediment. Applications include scouring in industrial tanks, port hydrodynamics, wave breaking in coastal applications and scour around structures in civil and environmental engineering flows among others.

#### ***3.11.1. Description of the physical problem***

A typical saturated sediment scour induced by rapid liquid flow at the interface undergoes a number of different behavioural regime changes mostly governed by the characteristics of the sediment and liquid phase rheology at the interface. These sediment regimes are distinguishable as an un-yielded region of sediment, a yielded non-Newtonian region and a pseudo Newtonian sediment suspension region where the sediment is entrained by the liquid flow. These physical processes can be described by the Coulomb shear stress  $\tau_{mc}$ , the cohesive yield strength  $\tau_c$  which accounts for the cohesive nature of fine sediment, the viscous shear stress  $\tau_v$  which accounts for the fluid particle viscosity, the turbulent shear stress of the sediment particle  $\tau_t$  and the dispersive stress  $\tau_d$  which accounts for the collision of larger fraction granulate. The total shear stress can be expressed as

$$\tau^{\alpha\beta} = \tau_{mc} + \tau_c + \tau_v^{\alpha\beta} + \tau_t^{\alpha\beta} + \tau_d^{\alpha\beta} \quad (54)$$

The first two parameters on the right-hand side of the equation define the yield strength of the material and thus can be used to differentiate the un-yielded or yielded region of the sediment state according to the induced stress by the liquid phase in the interface. The model implemented in DualSPHysics uses the Drucker-Prager yield criterion to evaluate yield strength of the sediment phase and the sediment failure surface.

When the material yields the sediment behaves as a non-Newtonian rate dependent Bingham fluid that accounts for the viscous and turbulent effects of the total shear stress of Eq. 54. Typically, sediment behaves as a shear thinning material with a low and high shear stress state of a pseudo-Newtonian and plastic viscous regime respectively. Herein, the Herschel-Buckley-Papanastasiou model is employed as a power law

Bingham model. This combines the yielded and un-yielded region using an exponential stress growth parameter and a power law Bingham model for the shear thinning or thickening plastic region.

Finally, the characteristics of the low concentration suspended sediment that has been entrained by the liquid are modelled using a volumetric concentration based viscosity in a pseudo-Newtonian approach by employing the Vand equation.

### 3.11.2. Sediment phase

The yield surface prediction is modelled using the Drucker-Prager (DP) model. The DP can be written in a general form as [Fourtakas and Rogers, 2016]

$$f(I_1, J_2) = \sqrt{J_2} + (ap - \kappa) = 0 \quad (55)$$

The parameters  $a$  and  $\kappa$  can be determined by projecting the Drucker-Prager onto the Mohr-Coulomb yield criterion in a deviatoric plane

$$a = -\frac{2\sqrt{3}\sin(\phi)}{3-\sin(\phi)} \quad \kappa = \frac{2\sqrt{3}\cos(\phi)}{3-\sin(\phi)} \quad (56)$$

where  $\phi$  is the internal friction and  $c$  the cohesion of the material. Finally, yielding will occur when the following equation is satisfied

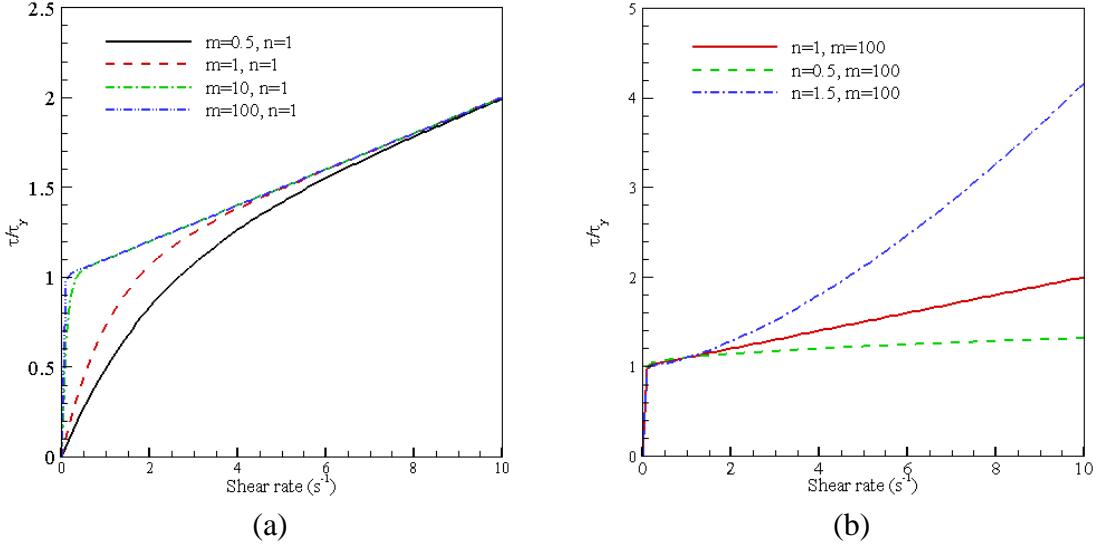
$$-ap + \kappa < 2\mu_d \sqrt{II_D} \quad (57)$$

The multi-phase model uses the Herschel-Bulkley-Papanastasiou (HBP) [Papanastasiou, 1987] rheological characteristics to model the yielded region. The HBP model reads

$$\phi_l = \frac{|\tau_y|}{\sqrt{II_D}} \left[ 1 - e^{-m\sqrt{II_D}} \right] + 2\mu |4II_D|^{\frac{n-1}{2}} \quad (58)$$

where  $m$  controls the exponential growth of stress,  $n$  is the power law index and  $\mu$  is the apparent dynamic viscosity (or consistency index for sediment flows). Figure 3-2(a) shows the initial rapid growth of stress by varying  $m$  whereas Figure 3-2(b) shows the effect of the power law index  $n$ .

Note that as  $m \rightarrow \infty$  the HBP model reduces to the original Herschel-Bulkley model and when  $n=1$  the model reduces to a simple Bingham model. Consequently, when  $n=1$  and  $m=0$  the model reduces to a Newtonian constitutive equation. Therefore, both phases can be modelled using the same constitutive equation. Most importantly, since the HBP parameters can be adjusted independently for each phase the current model is not restricted to Newtonian/Non-Newtonian formulation but can simulate a variety of combinations of flows (i.e. Newtonian/Newtonian, Non-Newtonian/Non-Newtonian with or without a yield strength, etc.).



**Figure 3-2.** Initial rapid growth of stress by varying  $m$  and effect of the power law index  $n$  for the HBP model.

The rheological characteristics of the sediment entrainment by the fluid can be controlled through the volume fraction of the mixture by using a concentration volume fraction in the form of

$$c_{v,i} = \frac{\sum_{j_{sat} \in 2h}^N \frac{m_j}{\rho_j}}{\sum_{j \in 2h}^N \frac{m_j}{\rho_j}} \quad (59)$$

where the summation is defined within the support of the kernel and  $j_{sat}$  refers to the yielded saturated sediment particles only.

We use a suspension viscosity based on the Vand experimental colloidal suspension equation [Vand, 1948] of sediment in a fluid by

$$\mu_{susp} = \mu e^{\frac{2.5c_v}{1 - \frac{39}{64}c_v}} \quad c_v \leq 0.3 \quad (60)$$

assuming an isotropic material with spherically shaped sediment particles. Eq. 60 is applied only when the volumetric concentration of the saturated sediment particle within the SPH kernel is lower than 0.3, which is the upper validity limit of Eq. 60.

More information about this multi-phase implementation can be also found in [Fourtakas, 2014; Fourtakas and Rogers, 2016].



## 4. CPU and GPU implementation

Detailed information about the CPU and GPU implementation can be found in the papers:

Crespo AJC, Domínguez JM, Rogers BD, Gómez-Gesteira M, Longshaw S, Canelas R, Vacondio R, Barreiro A, García-Feal O. 2015. DualSPHysics: open-source parallel CFD solver on Smoothed Particle Hydrodynamics (SPH). Computer Physics Communications, 187: 204-216. doi: 10.1016/j.cpc.2014.10.004

Domínguez JM, Crespo AJC, Valdez-Balderas D, Rogers BD, and Gómez-Gesteira M. 2013. New multi-GPU implementation for Smoothed Particle Hydrodynamics on heterogeneous clusters. Computer Physics Communications, 184: 1848-1860. doi: 10.1016/j.cpc.2013.03.008

Domínguez JM, Crespo AJC and Gómez-Gesteira M. 2013. Optimization strategies for CPU and GPU implementations of a smoothed particle hydrodynamics method. Computer Physics Communications, 184(3): 617-627. doi:10.1016/j.cpc.2012.10.015

Valdez-Balderas D, Domínguez JM, Rogers BD, Crespo AJC. 2012. Towards accelerating smoothed particle hydrodynamics simulations for free-surface flows on multi-GPU clusters. Journal of Parallel and Distributed Computing, doi:10.1016/j.jpdc.2012.07.010

Crespo AJC, Domínguez JM, Barreiro A, Gómez-Gesteira M and Rogers BD. 2011. GPUs, a new tool of acceleration in CFD: Efficiency and reliability on Smoothed Particle Hydrodynamics methods. PLoS ONE, 6(6), e20685. doi:10.1371/journal.pone.0020685

Domínguez JM, Crespo AJC, Gómez-Gesteira M, Marongiu, JC. 2011. Neighbour lists in Smoothed Particle Hydrodynamics. International Journal For Numerical Methods in Fluids, 67(12): 2026-2042. doi: 10.1002/fld.2481

The DualSPHysics code is the result of an optimised implementation using the best approaches for CPU and GPU with the accuracy, robustness and reliability shown by the SPHysics code. SPH simulations such as those in the SPHysics and DualSPHysics codes can be split in three main steps; (i) generation of the neighbour list, (ii) computation of the forces between particles (solving momentum and continuity equations) and (iii) the update of the physical quantities at the next time step. Thus, running a simulation means executing these steps in an iterative manner:

1<sup>st</sup> STEP: Neighbour list (Cell-linked list described in [Domínguez et al., 2011]):

- Domain is divided into square cells of side  $2h$  (or the size of the kernel domain).
- A list of particles, ordered according to the cell to which they belong, is generated.
- All the arrays with the physical variables belonging the particles are reordered according the list of particles.

2<sup>nd</sup> STEP: Force computation:

- Particles of the same cell and adjacent cells are candidates to be neighbours.
- Each particle interacts with all its neighbouring particles (at a distance  $< 2h$ ).

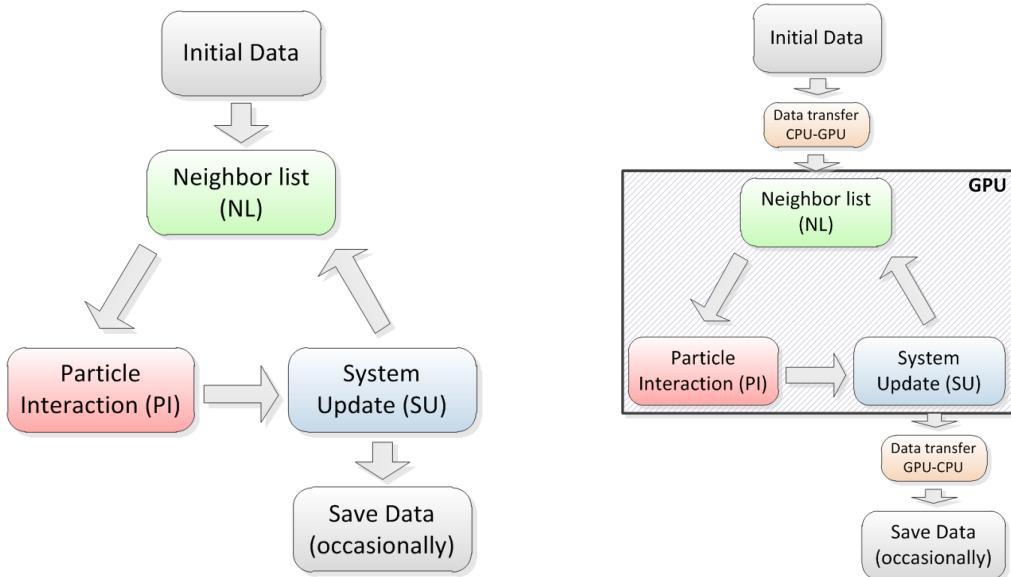
3<sup>rd</sup> STEP: System Update:

- New time step is computed.
- Physical quantities for the next step are updated starting from the values of physical variables at the present or previous time steps using the particle interactions.
- Particle information (velocity and density) are saved on local storage (the hard drive) at defined times.

The GPU implementation is focused on the force computation since following [Domínguez et al., 2011] this is the most consuming part in terms of runtime. However the most efficient technique consists of minimising the communications between the CPU and GPU for the data transfers. If neighbour list and system update are also implemented on the GPU the CPU-GPU memory transfer is needed at the beginning of the simulation while relevant data will be transferred to the CPU when saving output data is required (usually infrequently). [Crespo et al., 2011] used an execution of DualSPHysics performed entirely on the GPU to run a numerical experiment where the results are in close agreement with the experimental results.

The GPU implementation presents some key differences in comparison to the CPU version. The main difference is the parallel execution of all tasks that can be parallelised such as all loops regarding particles. One GPU execution thread computes the resulting force of one particle performing all the interactions with its neighbours. Different to previous versions, in version 4.0 onwards, the symmetry of the particle interaction is not employed on the CPU, the same as in the GPU implementation. On a GPU it is not efficient due to memory coalescence issues. Now, the new CPU structure mimics the GPU threads, which ensures continuity of coding and structure (hence ease of debugging, etc.) – see Section 6.

DualSPHysics is unique where the same application can be run using either the CPU or GPU implementation; this facilitates the use of the code not only on workstations with an Nvidia GPU but also on machines without a CUDA-enabled GPU. The CPU version is parallelised using the OpenMP API. The main code has a common core for both the CPU and GPU implementations with only minor source code differences implemented for the two devices applying the specific optimizations for CPU and GPU. Thus, debugging or maintenance is easier and comparisons of results and computational time are more direct.



**Figure 4-1.** Flow diagram of the CPU (left) and total GPU implementation (right).

### **Double precision**

The parallel computing power of Graphics Processing Units (GPUs) has led to an important increase in the size of the simulations but problems of precision can appear when simulating large domains with high resolution (specially in 2-D simulations). Hence, there are numerous simulations that require a small particle size “ $dp$ ” relative to the computational domain [[Domínguez et al., 2013c](#)], namely fine resolution or long domains. DualSPHysics v4.0 now includes an implementation with double precision where necessary. For example, arrays of position now use double precision and updating state of particles is also implemented with double precision.

### **OpenMP for multi-core executions.**

In the new version, the CPU implementation aims to achieve a higher performance in the current machines that contains many cores. Now, the interactions of a given particles with all its neighbours are carried out by the same execution thread. Symmetry in the force computation is not applied in order to increase the parallelization level of the algorithms. Previous versions of DualSPHysics were fast on CPUs with 4-8 cores but efficiency decreases significantly with number of cores. Furthermore, memory consumption increases with more cores. The new CPU code achieves an efficiency of 86.2% simulating 150,000 particles with 32 cores while the same execution achieved an efficiency of only 59.7% with previous version 3.0 and without memory increase. The current OpenMP implementation is not only fast and efficient with many cores, but also offers more advantages for users that want to modify the code; implementation is now easier since symmetry is removed during force computation such that the CPU code is more similar to the GPU code which facilitates its comprehension and editing.

### **Optimisation of the size of blocks for execution of CUDA kernels.**

The size of the blocks is a key parameter in the execution of CUDA kernels since it can lead to performance differences of 50%. This variation becomes more significant in the kernels that compute particle interactions since take more than 90% of the total execution time.

The version 4.0 includes a new automatic estimation of the optimum block size of CUDA kernels (particle interactions on GPU). This optimum block size depends on: (i) features of the kernel (registers and shared memory), (ii) compilation parameters and the CUDA version, (iii) hardware to be used and GPU specifications and (iv) input data to be processed by the kernel (divergence, memory coalescent access). The CUDA Occupancy Calculator is available from CUDA version 6.5.



## 5. Running DualSPHysics

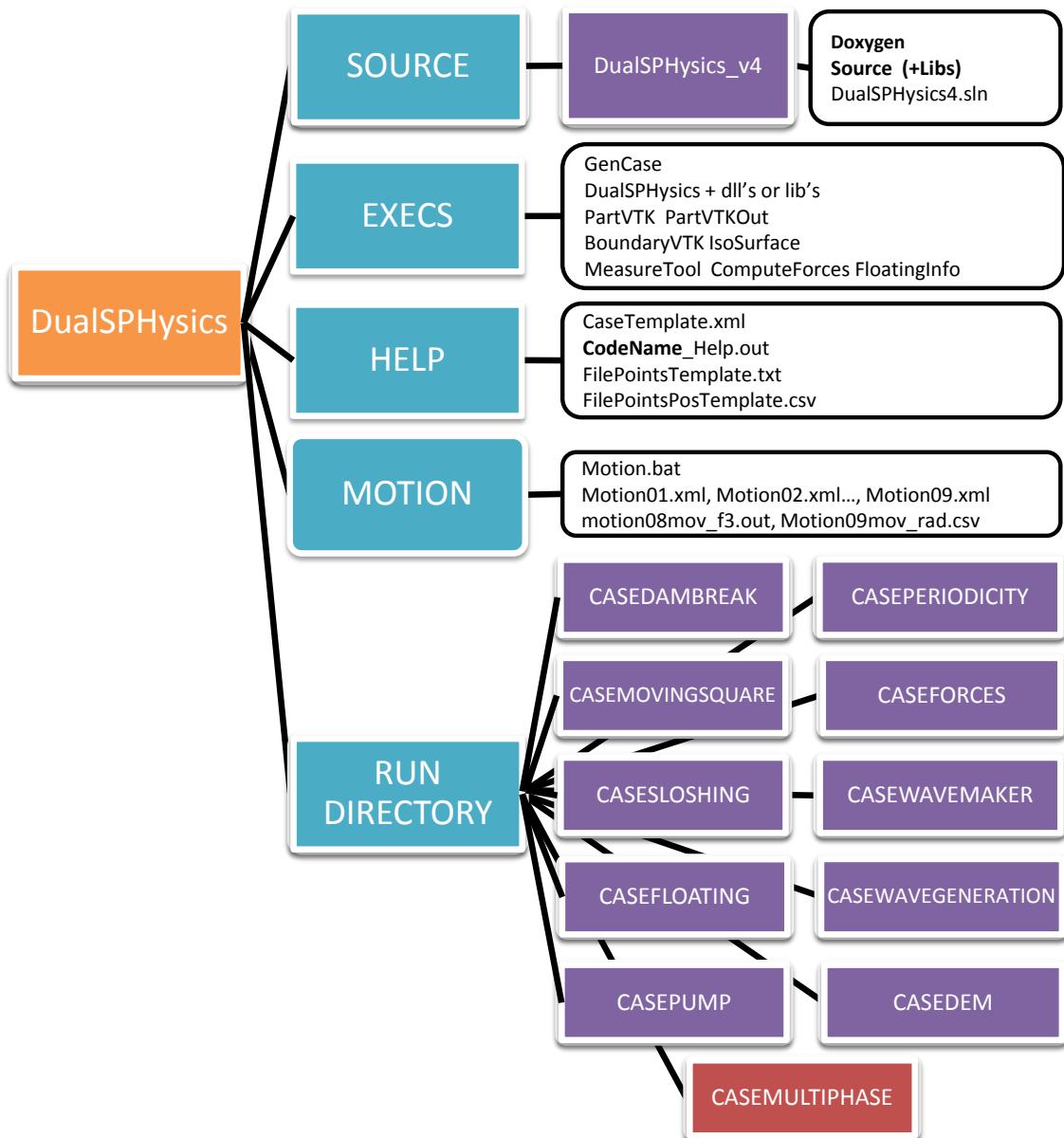
The user can download from <http://dual.sphysics.org/index.php/downloads/> the following files:

- **DUALSPHYSICS DOCUMENTATION:**
  - o **DualSPHysics\_v4.0\_GUIDE.pdf**
  - o **XML\_GUIDE\_v4.0.pdf**
  - o **ExternalModelsConversion\_GUIDE.pdf**
  - o **PostprocessingCalculations.pdf**
- **DUALSPHYSICS PACKAGE:**
  - o **DualSPHysics\_v4.0\_Linux\_x64.zip**
  - o **DualSPHysics\_v4.0\_Windows\_x64.zip**

To start using DualSPHysics, users should follow these instructions:

- 1) First, download and read the **DUALSPHYSICS DOCUMENTATION**:
  - *DualSPHysics\_v4.0\_GUIDE.pdf*: This manuscript.
  - *XML\_GUIDE.pdf*: Helps to create a new case using the input XML file and all XML parameters are explained and related to SPH equations.
  - *ExternalModelsConversion\_GUIDE.pdf*: Describes how to convert the file format of any external geometry of a 3-D model to VTK, PLY or STL using open-source codes.
  - *PostprocessingCalculations.pdf*: Explains how numerical magnitudes are computed.
- 2) Download the **DUALSPHYSICS PACKAGE** (See Figure 5-1):
  - SOURCE
  - EXECS
  - HELP
  - MOTION
  - RUN\_DIRECTORY:
    - o CASEDAMBREAK
    - o CASEPERIODICITY
    - o CASEMOVINGSQUARE
    - o CASEFORCES
    - o CASESLOSHINGCASEWAVEMAKER
    - o CASEWAVEGENERATION
    - o CASEFLOATING
    - o CASEPUMP
    - o CASEDEM
    - o CASEMULTIPHASE

Figure 5-1 shows the structure of the content of **DUALSPHYSICS\_PACKAGE**.



**Figure 5-1.** Directory tree and provided files.

- 3) The appropriate scripts (*.bat* in Windows and *.sh* in Linux) in the CASE directories that reside in the RUN\_DIRECTORY. These scripts must be used depending on the choice of CPU or GPU execution.
- 4) Paraview open-source software ([www.paraview.org](http://www.paraview.org)) is recommended to be used to visualise the results.

**SOURCE:**

- Contains the source files of DualSPHysics\_v4.0.
- Visual studio project for windows and Makefiles for linux.
- Code documentation using *Doxygen*.
- The release includes not only the source files of DualSPHysics v4.0 but also the source files of a code named “ToVtk”. This code is provided to show how to load and interpret particle data, how to read .bi4 files and how to create .vtk files.

**EXECS:**

- Contains the binaries executables.
- Some libraries needed for the codes are also included.

**HELP:**

- Contains *CaseTemplate.xml*, a XML example with all the different labels and formats that can be used in the input XML file.
- A description of the execution parameters of the different codes is presented in *HELP\_NameCode.out*.
- Other TXT and CSV files with template of positions of points to be used with MeasureTool code.

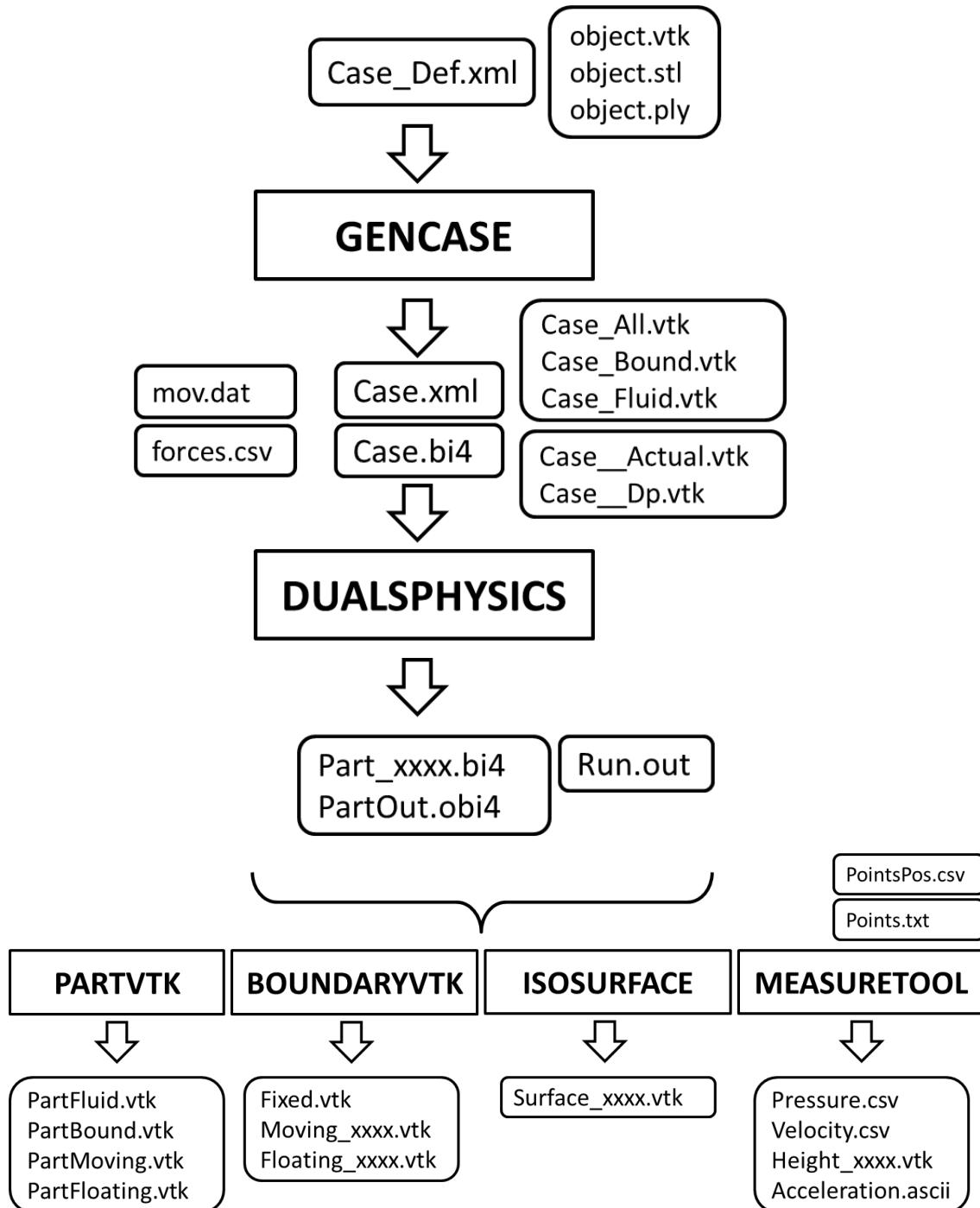
**MOTION:**

- Contains the script *Motion.bat* (*Motion.sh*) to perform the examples with the different type of movements that can be described with DualSPHysics.
- Nine examples can be carried out: *Motion01.xml*..., *Motion09.xml*.
- The text file *Motion08mov\_f3.out* describes the prescribed motion time vs position and the file *Motion09mov\_rad.csv* contains the prescribed rotational motion with time vs radians.

**RUN\_DIRECTORY:**

- This is the directory containing the test CASES. Each CASE directory (or folder) includes input XML, batch scripts to execute the cases and additional files to be used for DualSPHysics code. The output files will be created in the same directory:
  - o CASEDAMBREAK: CaseDambreak & CaseDambreakVal2D.
  - o CASEPERIODICITY: CasePeriodicity.
  - o CASEMOVINGSQUARE: CaseMovingSquare.
  - o CASEFORCES: CaseForces.
  - o CASESLOSHING: CaseSloshingAcc & CaseSloshingMotion.
  - o CASEWAVEMAKER: CaseWavemaker & CaseWavemaker2D.
  - o CASEWAVEGENERATION: CaseWavesREG & CaseWavesIRREG.
  - o CASEFLOATING: CaseFloating & CaseFloatingWavesVal & CaseFloatingSphereVal2D.
  - o CASEPUMP: CasePump.
  - o CASEDEM: CaseBowling & CaseSolids.
  - o CASEMULTIPHASE: CaseTwoPhases.

Figure 5-2 shows the workflow with representative example input and output files of the executable files. This figure will be used later to explain in detail each of the codes and the main tools.



**Figure 5-2.** Workflow of DualSPHysics.

## 6. DualSPHysics open-source code

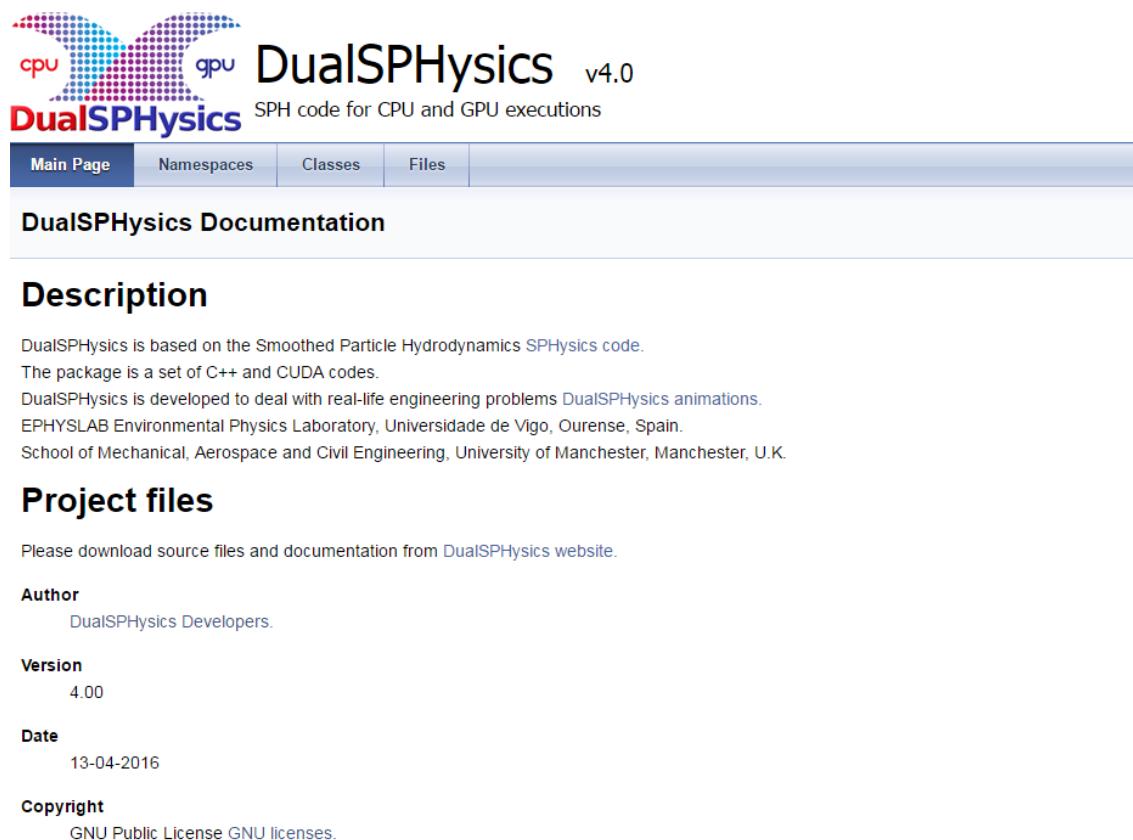
This section provides a brief description of the source files of DualSPHysics v4.0. The source code is freely redistributable under the terms of the GNU General Public License (GPL) as published by the Free Software Foundation ([www.gnu.org/licenses/](http://www.gnu.org/licenses/)). Thus, users can download the files from

### DUALSPHYSICS/SOURCE/DualSPHysics\_4/Source.

First, note that a more complete documentation is provided in directory

### DUALSPHYSICS/SOURCE/DualSPHysics\_4/Doxygen.

This documentation has been created using the documentation system Doxygen ([www.doxygen.org](http://www.doxygen.org)). The user can open the HTML file *index.html* (Figure 6-1) located in the directory mentioned above to navigate through the full documentation.



**Figure 6-1.** Documentation for DualSPHysics code generated with Doxygen.

Open source files are in **DUALSPHYSICS/SOURCE/DualSPHysics\_4/Source**. A complete list of these source files is summarised in Table 6-1. Some files are shared with other codes such as GenCase, BoundaryVTK, PartVTK, PartVTKOut, MeasureTool and IsoSurface. The rest of the files implement the SPH solver, some of them are used both for CPU/GPU executions and others are specific.

**Table 6-1.** List of source files of DualSPHysics code.

Common	SPH on CPU	SPH on GPU
Functions (.h .cpp) FunctionsMath (.h .cpp) JBinaryData (.h .cpp) JException (.h .cpp) JLog2 (.h .cpp) JMatrix4.h JMeanValues (.h .cpp) JObject (.h .cpp) JObjectGpu (.h .cpp) JPartDataBi4 (.h .cpp) JPartFloatBi4 (.h .cpp) JPartOutBi4Save (.h .cpp) JRadixSort (.h .cpp) JRangeFilter (.h .cpp) JReadDatafile (.h .cpp) JSpaceCtes (.h .cpp) JSpaceEParms (.h .cpp) JSpaceParts (.h .cpp) JSpaceProperties (.h .cpp) JTimer.h JTimerCuda.h TypesDef.h		main.cpp JCfgRun (.h .cpp) JPartsLoad4 (.h .cpp) JPartsOut (.h .cpp) JSaveDt.cpp (.h .cpp) JSph (.h .cpp) JSphAccInput (.h .cpp) JSphDtFixed (.h .cpp) JSphVisco (.h .cpp) JTimerClock.h JTimeOut (.h .cpp) Types.h
JFormatFiles2.h <i>JFormatFiles2.lib / libjformatfiles2.a</i>	JSphCpu (.h .cpp)	JSphGpu (.h .cpp) JSphGpu_ker (.h .cu)
JSphMotion.h <i>JSphMotion.lib / libjsphmotion.a</i>	JSphCpuSingle (.h .cpp)	JSphGpuSingle (.h .cpp)
JXml.h <i>JXml.lib / libjxml.a</i> <i>tinystr.h tinyxml.h</i>	JSphTimersCpu.h	JSphTimersGpu.h
JWaveGen.h <i>JWaveGen.lib / libjwavegen.a</i> <i>randomc (.h .cpp)</i>	JSphDivCpu (.h .cpp) JCellDivCpu (.h .cpp) JCellDivCpuSingle (.h .cpp) JArraysCpu (.h .cpp)	JCellDivGpu (.h .cpp) JCellDivGpu_ker (.h .cu) JCellDivGpuSingle (.h .cpp) JCellDivGpuSingle_ker (.h .cu) JArraysGpu (.h .cpp) JBlockSizeAuto (.h .cpp)

## **COMMON FILES:**

### **Functions.h & Functions.cpp**

Declares/implements basic/general functions for the entire application.

### **JBinaryData.h & JBinaryData.cpp**

Declares/implements the class that defines any binary format of a file.

### **JException.h & JException.cpp**

Declares/implements the class that defines exceptions with the information of the class and method.

### **JLog2.h & JLog2.cpp**

Declares/implements the class that manages the output of information in the file Run.out and on screen.

### **JMatrix4.h**

Declares the template for a matrix 4x4 used for geometric transformation of points in space.

### **JMeanValues.h & JMeanValues.cpp**

Declares/implements the class that calculates the average value of a sequence of values.

### **JObject.h & JObject.cpp**

Declares/implements the class that defines objects with methods that throw exceptions.

### **JObjectGpu.h & JObjectGpu.cpp**

Declares/implements the class that defines objects with methods that throw exceptions for tasks on the GPU.

### **JPartDataBi4.h & JPartDataBi4.cpp**

Declares/implements the class that allows reading files with data of particles in format bi4.

### **JPartFloatBi4.h & JPartFloatBi4.cpp**

Declares/implements the class that allows reading information of floating objects saved during simulation.

### **JPartOutBi4Save.h & JPartOutBi4Save.cpp**

Declares/implements the class that allows writing information of excluded particles during simulation.

### **JRadixSort.h & JRadixSort.cpp**

Declares/implements the class that implements the algorithm RadixSort.

### **JRangeFilter.h & JRangeFilter.cpp**

Declares/implements the class that facilitates filtering values within a list.

### **JReadDatafile.h & JReadDatafile.cpp**

Declares/implements the class that allows reading data in ASCII files.

### **JSpaceCtes.h & JSpaceCtes.cpp**

Declares/implements the class that manages the info of constants from the input XML file.

### **JSpaceEParms.h & JSpaceEParms.cpp**

Declares/implements the class that manages the info of execution parameters from the input XML file.

### **JSpaceParts.h & JSpaceParts.cpp**

Declares/implements the class that manages the info of particles from the input XML file.

## **JSpaceProperties.h & JSpaceProperties.cpp**

Declares/implements the class that manages the properties assigned to the particles in the XML file.

## **JTimer.h**

Declares the class that defines a class to measure short time intervals.

## **JTimerCuda.h**

Declares the class that defines a class to measure short time intervals on the GPU using cudaEvent.

## **TypesDef.h**

Declares general types and functions for the entire application.

## **JFormatFiles2.h**

Declares the class that provides functions to store particle data in formats VTK, CSV, ASCII.

## **JFormatFiles2.lib (libjformatfiles2.a)**

Precompiled library that provides functions to store particle data in formats VTK, CSV, ASCII.

## **JSphMotion.h**

Declares the class that provides the displacement of moving objects during a time interval

## **JSphMotion.lib (libjsphmotion.a)**

Precompiled library that provides the displacement of moving objects during a time interval.

## **JXml.h**

Declares the class that helps to manage the XML document using library TinyXML

## **JXml.lib (libjxml.a)**

Precompiled library that helps to manage the XML document using library TinyXML.

## **JWaveGen.h**

Declares the class that implements wave generation for regular and irregular waves.

## **JWaveGen.lib (libjwavegen.a)**

Precompiled library that implements wave generation for regular and irregular waves.

## **SPH SOLVER:**

### **main.cpp**

Main file of the project that executes the code on CPU or GPU.

## **JCfgRun.h & JCfgRun.cpp**

Declares/implements the class that defines the class responsible for collecting the execution parameters by command line.

## **JPartsLoad4.h & JPartsLoad4.cpp**

Declares/implements the class that manages the initial load of particle data.

## **JPartsOut.h & JPartsOut.cpp**

Declares/implements the class that stores excluded particles at each instant until writing the output file.

## **JSaveDt.h & JSaveDt.cpp**

Declares/implements the class that manages the use of prefixed values of DT loaded from an input file.

**JSph.h & JSph.cpp**

Declares/implements the class that defines all the attributes and functions that CPU and GPU simulations share.

**JSphAccInput.h & JSphAccInput .cpp**

Declares/implements the class that manages the application of external forces to different blocks of particles (with the same MK).

**JSphDtFixed.h & JSphDtFixed.cpp**

Declares/implements the class that manages the info of dt.

**JSphVisco.h & JSphVisco.cpp**

Declares/implements the class that manages the use of viscosity values from an input file.

**JTimerClock.h**

Defines a class to measure time intervals with precision of clock().

**JTimeOut.h & JTimeOut.cpp**

Declares/implements the class that manages the use of variable output time to save PARTs.

**Types.h**

Defines specific types for the SPH application.

**SPH SOLVER ONLY FOR CPU EXECUTIONS:****JSphCpu.h & JSphCpu.cpp**

Declares/implements the class that defines the attributes and functions used only in CPU simulations.

**JSphCpuSingle.h & JSphCpuSingle.cpp**

Declares/implements the class that defines the attributes and functions used only in Single-CPU.

**JSphTimersCpu.h**

Measures time intervals during CPU execution.

**JCellDivCpu.h & JCellDivCpu.cpp**

Declares/implements the class responsible of generating the Neighbour List in CPU.

**JCellDivCpuSingle.h & JCellDivCpuSingle.cpp**

Declares/implements the class responsible of generating the Neighbour List in Single-CPU.

**JArraysCpu.h & JArraysCpu.cpp**

Declares/implements the class that manages arrays with memory allocated in CPU.

## **SPH SOLVER ONLY FOR GPU EXECUTIONS:**

### **JSphGpu.h & JSphGpu.cpp**

Declares/implements the class that defines the attributes and functions used only in GPU simulations.

### **JSphGpu\_ker.h & JSphGpu\_ker.cu**

Declares/implements functions and CUDA kernels for the Particle Interaction (PI) and System Update (SU).

### **JSphGpuSingle.h & JSphGpuSingle.cpp**

Declares/implements the class that defines the attributes and functions used only in single-GPU.

### **JSphTimersGpu.h**

Measures time intervals during GPU execution.

### **JCellDivGpu.h & JCellDivGpu.cpp**

Declares/implements the class responsible of generating the Neighbour List in GPU.

### **JCellDivGpu\_ker.h & JCellDivGpu\_ker.cu**

Declares/implements functions and CUDA kernels to generate the Neighbour List in GPU.

### **JCellDivGpuSingle.h & JCellDivGpuSingle.cpp**

Declares/implements the class that defines the class responsible of generating the Neighbour List in Single-GPU.

### **JCellDivGpuSingle\_ker.h & JCellDivGpuSingle\_ker.cu**

Declares/implements functions and CUDA kernels to compute operations of the Neighbour List.

### **JArraysGpu.h & JArraysGpu.cpp**

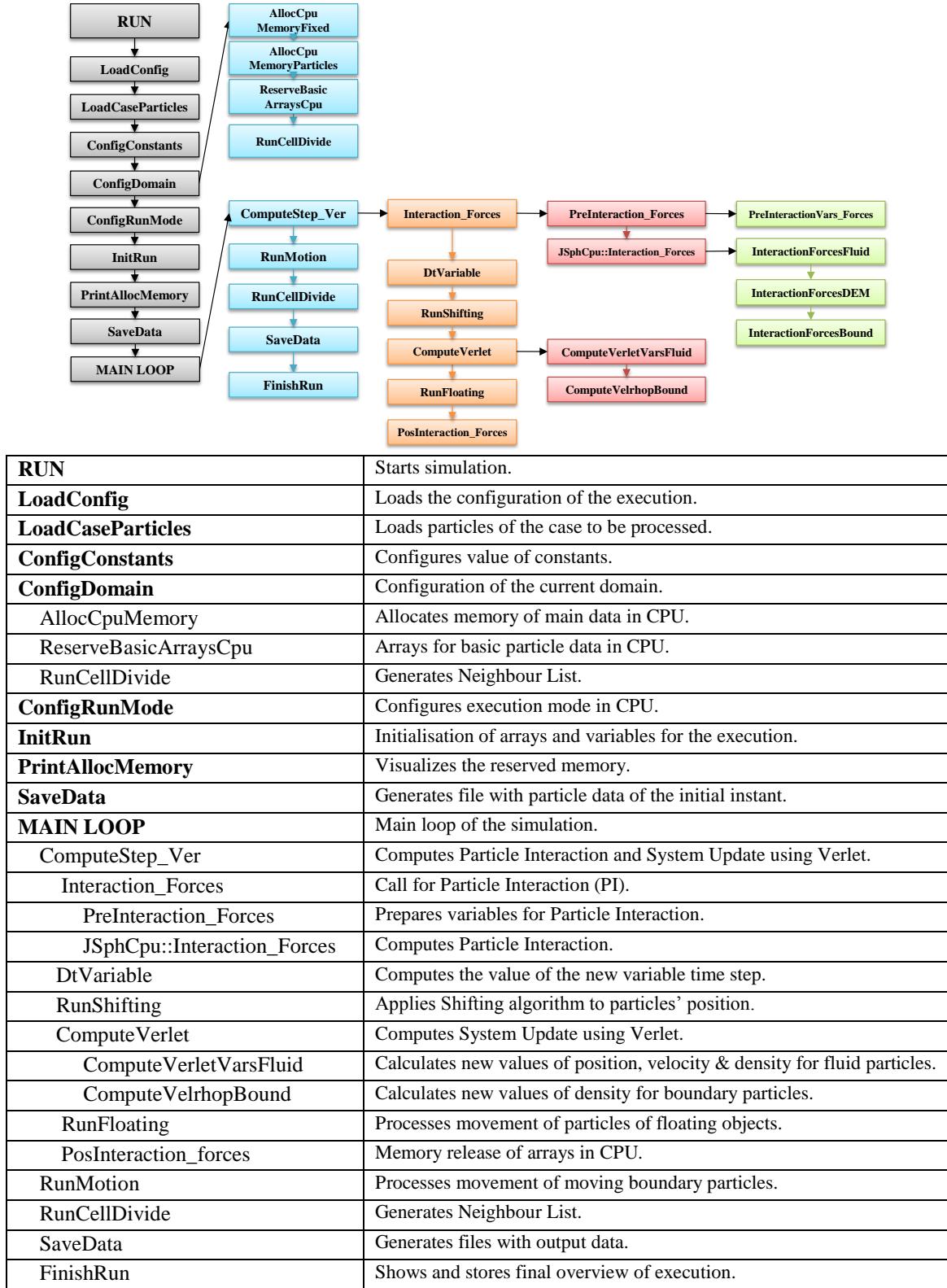
Declares/implements the class that manages arrays with memory allocated in GPU.

### **JBlockSizeAuto.h & JBlockSizeAuto.cpp**

Declares/implements the class that manages the automatic computation of optimum Blocksize in kernel interactions.

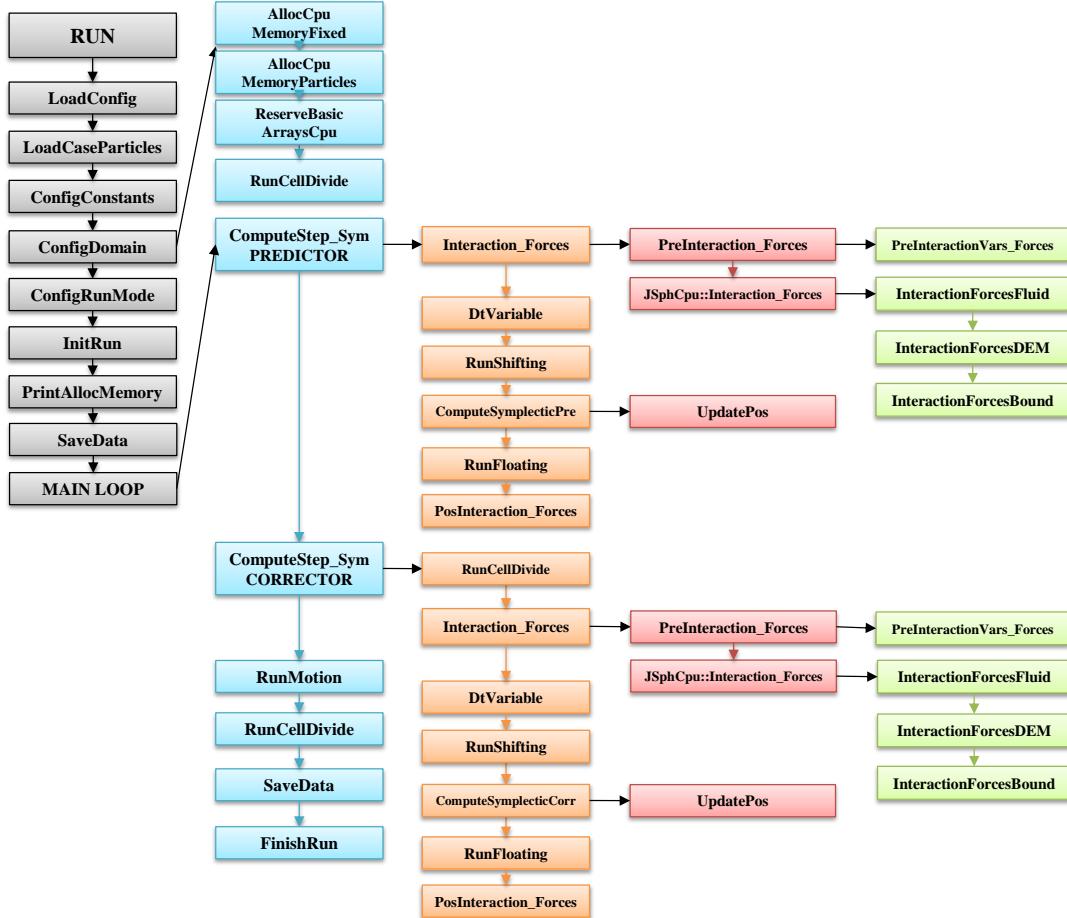
## 6.1 CPU source files

The source file **JSphCpuSingle.cpp** can be better understood with the help of the diagram of calls represented in Figure 6-2. Note that now particle interactions are no longer performed in terms of cells as used in previous versions.



**Figure 6-2.** Workflow of **JSphCpuSingle.cpp** when using Verlet time algorithm.

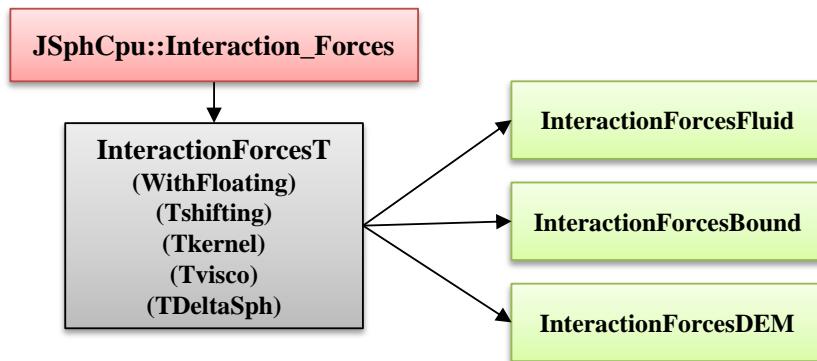
When the Symplectic timestepping integration scheme is used the step is split in predictor and corrector steps. Thus, Figure 6-3 shows the workflow and calls of the CPU code using this time scheme:



<b>ComputeStep_Sym</b>	Computes Particle Interaction and System Update using Symplectic.
<b>PREDICTOR</b>	Predictor step.
Interaction_Forces	Call for Particle Interaction (PI).
PreInteraction_Forces	Prepares variables for Particle Interaction.
Interaction_Forces	Computes Particle Interaction.
DtVariable	Computes the value of the new variable time step.
RunShifting	Applies Shifting algorithm to particles' position.
ComputeSymplecticPre	Computes System Update using Symplectic-Predictor.
UpdatePos	Computes new positions of the particles.
RunFloating	Processes movement of particles of floating objects.
PosInteraction_Forces	Memory release of arrays in CPU.
<b>CORRECTOR</b>	Corrector step.
Interaction_Forces	Call for Particle Interaction (PI).
PreInteraction_Forces	Prepares variables for Particle Interaction.
Interaction_Forces	Computes Particle Interaction.
DtVariable	Computes the value of the new variable time step.
RunShifting	Applies Shifting algorithm to particles' position.
ComputeSymplecticCorr	Computes System Update using Symplectic-Corrector.
UpdatePos	Computes new positions of the particles.
RunFloating	Processes movement of particles of floating objects.
PosInteraction_Forces	Memory release of arrays in CPU.

**Figure 6-3.** Workflow of **JSphCpuSingle.cpp** when using Symplectic timestepping algorithm.

Note that *JSphCpu::Interaction\_Forces* performs the particle interaction in CPU using the template *InteractionForcesT*. Thus, the interaction between particles is carried out considering different parameters and considering the type of particles involved in the interaction as it can be seen in Figure 6-4 and Table 6-2:



**Figure 6-4.** Call graph for the template *InteractionForcesT*.

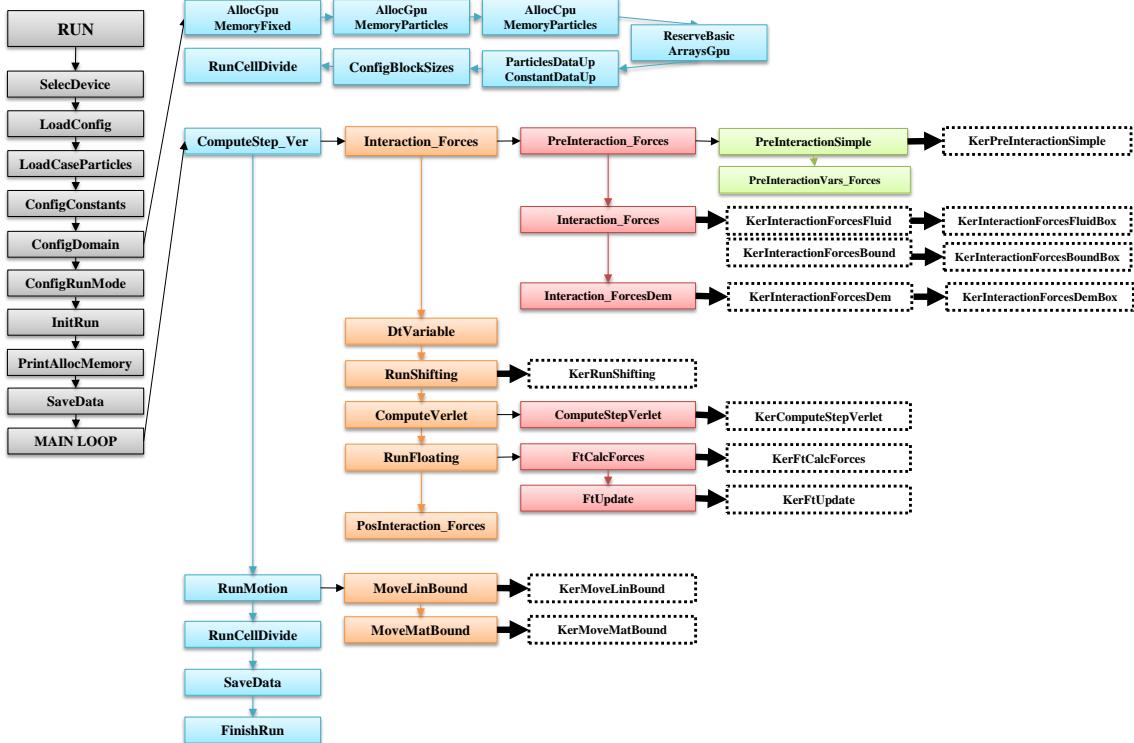
**Table 6-2.** Different particle interactions can be performed depending on the type of particles.

<b>InteractionForcesFluid</b>	SPH interaction between particles <i>Fluid/Floating-Fluid/Floating</i> <i>Fluid/Floating-Bound</i>
<b>InteractionForcesBound</b>	SPH interaction between particles <i>Bound-Fluid/Floating</i>
<b>InteractionForcesDEM</b>	DEM interaction between particles <i>Floating-Bound</i> <i>Floating-Floating</i>

As mentioned before, a more complete documentation has been generated using Doxygen.

## 6.2 GPU source files

The source file **JSpHGPUSingle.cpp** can be better understood with the workflow represented in Figure 6-5 that includes the functions implemented in the GPU files. The dashed boxes indicates the CUDA kernels implemented in the CUDA files (*JSpHGPU\_ker.cu*).



<b>RUN</b>	Starts simulation.
<b>SelecDevice</b>	Initialises CUDA device
<b>LoadConfig</b>	Loads the configuration of the execution.
<b>LoadCaseParticles</b>	Loads particles of the case to be processed.
<b>ConfigConstants</b>	Configures value of constants.
<b>ConfigDomain</b>	Configuration of the current domain.
<b>AllocGpuMemoryFixed</b>	Allocates memory in GPU of for arrays with fixed size.
<b>AllocGpuMemoryParticles</b>	Allocates memory in GPU of main data of particles.
<b>AllocCpuMemoryParticles</b>	Allocates memory in CPU of main data of particles.
<b>ReserveBasicArraysGpu</b>	Arrays for basic particle data in GPU.
<b>ParticlesDataUp</b>	Uploads particle data to the GPU.
<b>ConstantDataUp</b>	Uploads constants to the GPU.
<b>ConfigBlockSizes</b>	Calculates optimum BlockSize.
<b>RunCellDivide</b>	Generates Neighbour List.
<b>ConfigRunMode</b>	Configures execution mode in GPU.
<b>InitRun</b>	Initialisation of arrays and variables for the execution.
<b>PrintAllocMemory</b>	Visualizes the reserved memory in CPU and GPU.
<b>SaveData</b>	Generates file with particle data of the initial instant.

<b>MAIN LOOP</b>	Main loop of the simulation.
ComputeStep_Ver	Computes Particle Interaction and System Update using Verlet.
Interaction_Forces	Call for Particle Interaction (PI).
PreInteraction_Forces	Prepares variables for Particle Interaction.
Interaction_Forces	Computes Particle Interaction.
Interaction_ForcesDEM	Computes Particle Interaction with DEM.
DtVariable	Computes the value of the new variable time step.
RunShifting	Applies Shifting algorithm to particles' position.
ComputeVerlet	Call for System Update using Verlet.
ComputeStepVerlet	Computes System Update using Verlet.
RunFloating	Processes movement of particles of floating objects.
FtCalcForces	Computes forces on floating.
FtUpdate	Updates information and particles of floating bodies.
PosInteractionForces	Memory release of arrays in GPU.
RunMotion	Processes movement of moving boundary particles.
MoveLinBound	Applies a linear movement to a set of particles.
MoveMatBound	Applies a matrix movement to a set of particles.
RunCellDivide	Generates Neighbour List.
SaveData	Generates files with output data.
FinishRun	Shows and stores final overview of execution.

**Figure 6-5.** Workflow of **JSphGpuSingle.cpp** when using Verlet time algorithm.



## 7. Compiling DualSPHysics

The code can be compiled for either CPU or CPU&GPU. Please note that both the C++ and CUDA version of the code contain the same features and options. Most of the source code is common to CPU and GPU, which allows the code to be run on workstations without a CUDA-enabled GPU, using only the CPU implementation.

To run DualSPHysics on a GPU using an executable, only an Nvidia CUDA-enabled GPU card is needed and the latest version of the GPU driver must be installed. However, to compile the source code, the GPU programming language CUDA and *nvcc* compiler must be installed on your computer. CUDA Toolkit X.X can be downloaded from Nvidia website <http://developer.nvidia.com/cuda-toolkit-XX>. CUDA versions from 4.0 till 7.5 have been tested.

Once the C++ compiler (for example *gcc*) and the CUDA compiler (*nvcc*) have been installed in your machine, you can download the relevant files from the directory **DUALSPHYSICS/SOURCE/DualSPHysics\_4**:

### 7.1 Windows compilation

In **DUALSPHYSICS/SOURCE/DualSPHysics\_4** there are also several folders:

- Source: contains all the source files;
- Libs: precompiled libraries for x64 (debug and release);
- Doxygen: including the documentation generated by Doxygen;

The project file *DualSPHysics4\_vs2010.sln* is provided to be opened with Visual Studio 2010 and *DualSPHysics4\_vs2013.sln* to be opened with Visual Studio 2013.

Also different configurations can be chosen for compilation:

- a) **Release** for CPU and GPU
- b) **ReleaseCPU** only for CPU

The result of the compilation is the executable *DualSPHysics4\_win64.exe* or *DualSPHysics4CPU\_win64.exe* created in **DUALSPHYSICS/EXECS**.

The Visual Studio project is created including the libraries for OpenMP in the executable. To not include them, user can modify *Props config -> C/C++ -> Language -> OpenMp* and compile again

The use of OpenMP can be also deactivated by commenting the code line in **Types.h**:

```
#define _WITHOMP           ///<Enables/Disables OpenMP.
```

The binaries are already compiled and available in **DUALSPHYSICS/EXECS**. The GPU codes were compiled for compute capabilities sm20, sm30, sm35, sm37, sm50, sm52 and with CUDA v7.5.

## 7.2 Linux compilation

In **DUALSPHYSICS/SOURCE/DualSPHysics\_4** there are also several folders:

- Source: contains all the source files, the libraries and *makefiles*;
- Doxygen: including the documentation generated by Doxygen;

*Makefiles* can be used to compile the code in linux:

- a) make -f **Makefile** full compilation just using *make* command
- b) make -f **Makefile\_cpu** only for CPU

The result of the compilation is the executable *DualSPHysics4\_linux64* or *DualSPHysics4CPU\_linux64* created in **DUALSPHYSICS/EXECS**.

To exclude the use of OpenMP you have to remove the flags *-fopenmp* and *-lgomp* in the Makefile and comment line **#define \_WITHOMP** in **Types.h**.

This is the content of the file **Makefile** for linux:

```
===== Compilation Options =====
USE_DEBUG=NO
USE_FAST_MATH=YES
USE_NATIVE_CPU_OPTIMIZATIONS=YES

EXECS_DIRECTORY=../../EXECS

ifeq ($(USE_DEBUG), YES)
    CCFLAGS=-c -O0 -g -Wall -fopenmp -D_WITHGPU
else
    CCFLAGS=-c -O3 -fopenmp -D_WITHGPU
ifeq ($(USE_FAST_MATH), YES)
    CCFLAGS+= -ffast-math
endif
ifeq ($(USE_NATIVE_CPU_OPTIMIZATIONS), YES)
    CCFLAGS+= -march=native
endif
endif
CC=g++
CCLINKFLAGS=-fopenmp -lgomp

===== CUDA toolkit directory (make appropriate for local CUDA installation) =====
DIRTOOLKIT=/usr/local/cuda
DIRTOOLKIT=/exports/opt/NVIDIA/cuda-7.5

===== Files to compile =====
OBJ_BASIC=main.o Functions.o FunctionsMath.o JArraysCpu.o JBinaryData.o JCellDivCpu.o JCfgRun.o JException.o
OBJ_BASIC:=$(OBJ_BASIC) JLog2.o JObject.o JPartDataBi4.o JPartFloatBi4.o JPartOutBi4Save.o JPartsOut.o
OBJ_BASIC:=$(OBJ_BASIC) JRadixSort.o JRangeFilter.o JReadDatafile.o JSaveDt.o JSpaceCtes.o JSpaceEParms.o
JSpaceParts.o
OBJ_BASIC:=$(OBJ_BASIC) JSpaceProperties.o JSph.o JSphAccInput.o JSphCpu.o JSphDtFixed.o JSphVisco.o randomc.o
OBJ_BASIC:=$(OBJ_BASIC) JTimeOut.o
OBJ_CPU_SINGLE=JCellDivCpuSingle.o JSphCpuSingle.o JPartsLoad4.o
OBJ_GPU=JArraysGpu.o JCellDivGpu.o JObjectGpu.o JSphGpu.o JBlockSizeAuto.o JMeanValues.o
OBJ_GPU_SINGLE=JCellDivGpuSingle.o JSphGpuSingle.o
OBJ_CUDA=JCellDivGpu_ker.o JSphGpu_ker.o
OBJ_CUDA_SINGLE=JCellDivGpuSingle_ker.o
OBJECTS=$(OBJ_BASIC) $(OBJ_CPU_SINGLE) $(OBJ_GPU) $(OBJ_CUDA) $(OBJ_GPU_SINGLE) $(OBJ_CUDA_SINGLE)
```

```

#===== Select GPU architectures =====
GENCODE:=$(GENCODE) -gencode=arch=compute_20,code=\"sm_20,compute_20\""
GENCODE:=$(GENCODE) -gencode=arch=compute_30,code=\"sm_30,compute_30\""
GENCODE:=$(GENCODE) -gencode=arch=compute_35,code=\"sm_35,compute_35\""
GENCODE:=$(GENCODE) -gencode=arch=compute_37,code=\"sm_37,compute_37\""
GENCODE:=$(GENCODE) -gencode=arch=compute_50,code=\"sm_50,compute_50\""
GENCODE:=$(GENCODE) -gencode=arch=compute_52,code=\"sm_52,compute_52\""

#===== DualSPHysics libs to be included =====
JLIBS=-L./ -lxml_64 -lformatfiles2_64 -lsphmotion_64 -lwavegen_64

#===== GPU Code Compilation =====
CCFLAGS := $(CCFLAGS) -I./ -I$(DIRTOOLKIT)/include
CCLINKFLAGS := $(CCLINKFLAGS) -L$(DIRTOOLKIT)/lib64 -lcudart
NCC=nvcc
ifeq ($(USE_DEBUG), NO)
    NCCFLAGS=-c $(GENCODE) -O3
else
    NCCFLAGS=-c $(GENCODE) -O0 -g
endif
ifeq ($(USE_FAST_MATH), YES)
    NCCFLAGS+= -use_fast_math
endif

all: $(EXECS_DIRECTORY)/DualSPHysics4_linux64
    rm -rf *.o
ifeq ($(USE_DEBUG), NO)
    @echo " --- Compiled Release GPU version ---"
else
    @echo " --- Compiled Debug GPU version ---"
    mv DualSPHysics4_linux64 DualSPHysics4_linux64_debug
endif

$(EXECS_DIRECTORY)/DualSPHysics4_linux64: $(OBJECTS)
    $(CC) $(OBJECTS) $(CCLINKFLAGS) -o $@ $(JLIBS)

.cpp.o:
    $(CC) $(CCFLAGS) $<
JSphGpu_ker.o: JSphGpu_ker.cu
    $(NCC) $(NCCFLAGS) JSphGpu_ker.cu
JCellDivGpu_ker.o: JCellDivGpu_ker.cu
    $(NCC) $(NCCFLAGS) JCellDivGpu_ker.cu
JCellDivGpuSingle_ker.o: JCellDivGpuSingle_ker.cu
    $(NCC) $(NCCFLAGS) JCellDivGpuSingle_ker.cu

clean:
    rm -rf *.o DualSPHysics4_linux64 DualSPHysics4_linux64_debug

```

The user can modify the compilation options, the path of the CUDA toolkit directory, the GPU architecture. The GPU code is already compiled (EXECS) for compute capabilities sm20, sm30, sm35, sm37, sm50, sm52 and with CUDA v7.5.

### 7.3 Alternative building method via CMAKE

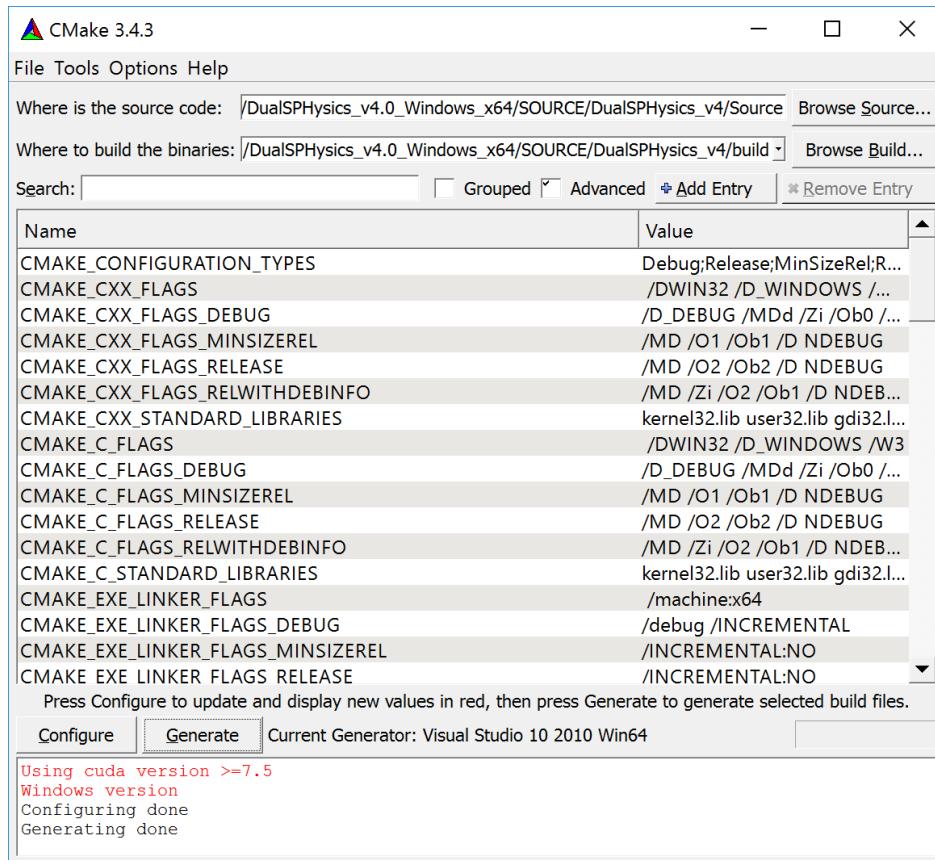
A new building method is supported in the new version 4.0 of DualSPHysics using CMAKE (<https://cmake.org/>). CMAKE is a cross-platform and an independent building system for compilation. This software generates native building files (like makefiles or Visual Studio projects) for any platform. The location of dependencies and the needed flags are automatically determined. *Note that this method is on trial for version 4.*

## Compile instructions for Microsoft Windows with Cmake

The building system needs the following dependencies:

- Cmake version 2.8.10 or greater (can be freely downloaded in <https://cmake.org/download/> )
  - Nvidia CUDA Toolkit version 4.0 or greater.
  - Visual Studio 2010 or 2013 version.
  - File “CMakeLists.txt” in **Source**.

The folder **build** will be created in **DUALSPHYSICS/SOURCE/DualSPHysics\_4**. This folder will contain the building files so it can be safely removed in case of rebuilding, it can actually be placed anywhere where the user has writing permissions. Afterwards, open the Cmake application, a new window will appear:



Paste the **Source** folder path in the textbox labeled as **Where is the source code**, and paste the **build** folder path into the **Where to build the binaries** textbox. Once the paths are introduced, the **Configure** button should be pressed. A new dialog will appear asking for the compiler to be used in the project. Please, remember that only Visual Studio 2010 and Visual Studio 2013 for 64bit are supported.

If the configuration succeeds, now press the **Generate** button. This will generate a Visual Studio project file into the **build** directory.

In order to compile both CPU and GPU versions, just change configuration to **Release** and compile. If the user only wants to compile one version, one can choose one of the solutions **dualspysics4cpu** or **dualspysics4gpu** for CPU or GPU versions respectively, and compile it.

The user can freely customize the **Source/CMakeLists.txt** file to add new source files or any other modifications. For more information about how to edit this file, please, refer to official Cmake documentation (<https://cmake.org/documentation/>).

## Compile instructions for Linux with Cmake

The building system needs the following dependencies:

- Cmake version 2.8.10 or greater.
- Nvidia CUDA Toolkit version 4.0 or greater.
- GNU G++ compiler 4.4 version or greater.
- File “CMakeLists.txt” in **Source**.

The folder **build** will be created in **DUALSPHYSICS/SOURCE/DualSPHysics\_4**. This folder will contain the building files so it can be safely removed in case of rebuilding, it can actually be placed anywhere where the user has writing permissions. To create this folder and run Cmake just type:

```
> cd DUALSPHYSICS/SOURCE/DualSPHysics_4
> mkdir build
> cd build
> cmake ..../Source
-- The C compiler identification is GNU 4.4.7
-- The CXX compiler identification is GNU 4.4.7
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
          -- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
Using cuda version <7.5
Using libraries for gcc version <5.0
-- Try OpenMP C flag = [-fopenmp]
-- Performing Test OpenMP_FLAG_DETECTED
-- Performing Test OpenMP_FLAG_DETECTED - Success
-- Try OpenMP CXX flag = [-fopenmp]
-- Performing Test OpenMP_FLAG_DETECTED
```

```
-- Performing Test OpenMP_FLAG_DETECTED - Success
-- Found OpenMP: -fopenmp
-- Configuring done
-- Generating done
-- Build files have been written to: /home/user/DUALSPHYSICS/SOURCE/DualSPHysics_v4/build
```

The command **cmake ..**/**Source** will search for a Cmake file (**CMakeLists.txt**) in the specified folder. As mentioned before, the user can freely customize this file to add new source files or any other modifications. For more information about how to edit this file, please, refer to Cmake official documentation (<https://cmake.org/documentation/>).

Once the cmake command runs without error, a Makefile can be found in the **build** folder. To build both CPU and GPU versions of DualSPHysics just type **make**. If the user only needs to build one of the executable files he can use the commands **make dualsphysics4cpu** or **make dualsphysics4gpu** for CPU and GPU versions respectively. In order to install the compiled binaries into the **EXECs** folder, the user can either copy the executable files or type the command **make install**.

## 8. Format Files

The codes provided within the DualSPHysics package present some important improvements in comparison to the codes available within SPHysics. One of them is related to the format of the files that are used as input and output data throughout the execution of DualSPHysics and the pre-processing and post-processing codes. Different format files for the input and the output data are involved in the DualSPHysics execution: XML, binary and VTK-binary.

### XML File

The XML (EXtensible Markup Language) is a textual data format that can easily be read or written using any platform and operating system. It is based on a set of labels (tags) that organise the information and can be loaded or written easily using any standard text or dedicated XML editor. This format is used for input files for the code.

### BINARY File

The output data in the SPHysics code is written in text files, so ASCII format is used. ASCII files present some interesting advantages such as visibility and portability, however they also present important disadvantages particularly with simulations with large numbers of particles: data stored in text format consumes at least six times more memory than the same data stored in binary format, precision is reduced when values are converted from real numbers to text while reading and writing data in ASCII is more expensive (two orders of magnitude). Since DualSPHysics allows performing simulations with a high number of particles, a binary file format is necessary to avoid these problems. Binary format reduces the volume of the files and the time dedicated to generate them. These files contain the meaningful information of particle properties. In this way, some variables can be removed, e.g., the pressure is not stored since it can be calculated starting from the density using the equation of state. The mass values are constant for fluid particles and for boundaries so only two values are used instead of an array. Data for particles that leave the limits of the domain are stored in an independent file (PartOut\_000.obi4) which leads to an additional saving. Hence, the advantages can be summarised as: (i) memory storage reduction, (ii) fast access, (iii) no precision lost and (iv) portability (i.e. to different architectures or different operating systems).

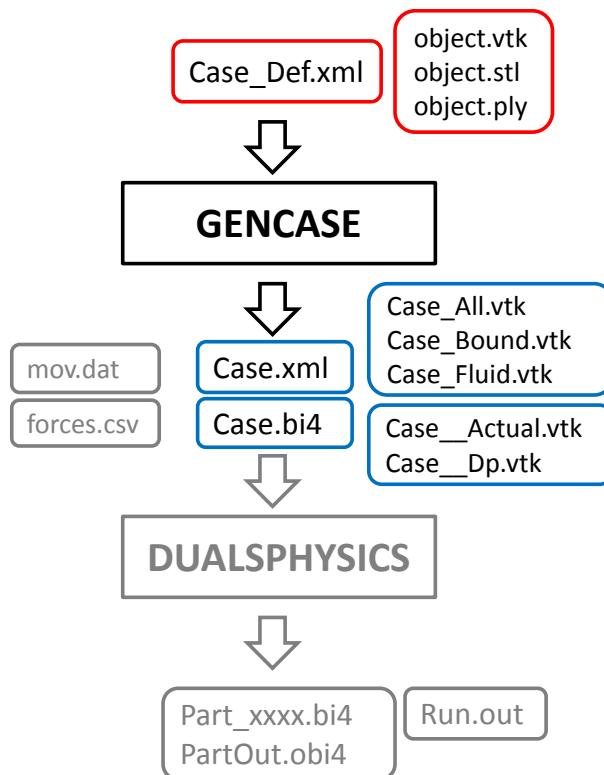
The file format used now in DualSPHysics v4.0 is named BINX4 (.bi4) which is the new binary format and can save particle position in single or double precision. This format file is a container so the user can add new metadata and new arrays can be processed in an automatic way using the current post-processing tools of the package.

## **VTK File**

VTK (Visualization ToolKit) files are used for final visualization of the results and can either be generated as a pre-processing step or output directly by DualSPHysics instead of the standard BINX format (albeit at the expense of computational overhead). VTK not only supports the particle positions, but also physical quantities that are obtained numerically for the particles involved in the simulations. VTK supports many data types, such as scalar, vector, tensor, texture, and also supports different algorithms such as polygon reduction, mesh smoothing, cutting, contouring and Delaunay triangulation. The VTK file format consists of a header that describes the data and includes any other useful information, the dataset structure with the geometry and topology of the dataset and its attributes. Here VTK files of POLYDATA type with legacy-binary format is used. This format is also easy for input-output (IO) or read-write operations.

## 9. Pre-processing

A program named **GenCase** is included to define the initial configuration of the simulation, movement description of moving objects and the parameters of the execution in DualSPHysics. All this information is contained in a definition **input file** in XML format; **Case\_Def.xml**. Two **output files** are created after running GenCase: **Case.xml** and **Case.bi4** (the input files for DualSPHysics code). These input (red boxes) and output files (blue boxes) can be observed in Figure 9-1. **Case.xml** contains all the parameters of the system configuration and its execution such as key variables (smoothing length, reference density, gravity, coefficient to calculate pressure, speed of sound...), the number of particles in the system, movement definition of moving boundaries and properties of moving bodies. **Case.bi4** contains the initial state of the particles (number of particles, position, velocity and density) in BINX4 (.bi4) format.

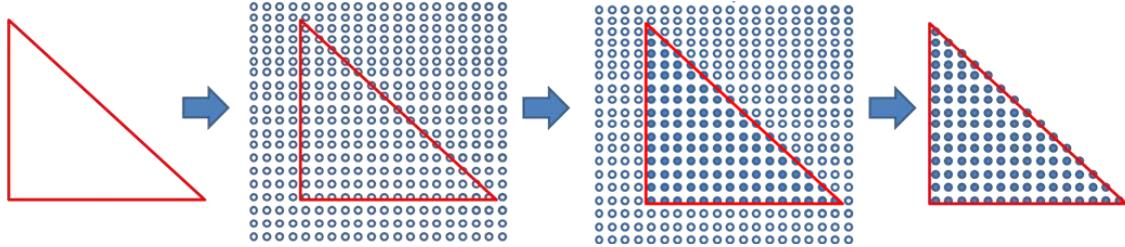


**Figure 9-1.** Input (red) and output (blue) files of GenCase code.

Particle geometries created with **GenCase** can be initially checked by visualising in Paraview the files **Case\_All.vtk**, **Case\_Bound.vtk** and **Case\_Fluid.vtk**.

GenCase employs a 3-D Cartesian mesh to locate particles. The idea is to build any object using particles. These particles are created at the nodes of the 3-D Cartesian mesh. Firstly, the mesh nodes around the object are defined and then particles are created only in the nodes needed to draw the desired geometry. Figure 9-2 illustrates how this mesh is used; in this case a triangle is generated in 2D. First the nodes of a

mesh are defined starting from the maximum dimensions of the desired triangle, then the edges of the triangle are defined and finally particles are created at the nodes of the Cartesian mesh which are inside the triangle.



**Figure 9-2.** Generation of a 2-D triangle formed by particles using **GenCase**.

All particles are placed over a regular Cartesian grid. The geometry of the case is defined independently to the inter-particle distance. This allows the discretization of each test case with a different number of particles simply by varying the resolution (or particle size)  $dp$ . Furthermore, **GenCase** is very fast and able to generate millions of particles only in seconds on the CPU.

Very complex geometries can be easily created since a wide variety of commands (**labels in the XML file**) are available to create different objects; points, lines, triangles, quadrilateral, polygons, pyramids, prisms, boxes, beaches, spheres, ellipsoids, cylinders, waves (`<drawpoint />`, `<drawpoints />`, `<drawline />`, `<drawlines />`, `<drawtriangle />`, `<drawquadri />`, `<drawtrianglesstrip />`, `<drawtrianglesfan />`, `<drawpolygon>`, `<drawtriangles />`, `<drawpyramid />`, `<drawprism />`, `<drawbox />`, `<drawbeach />`, `<drawsphere />`, `<drawellipsoid />`, `<drawcylinder />`, `<drawwave />`).

Once the mesh nodes that represent the desired object are selected, these points are stored as a matrix of nodes. The shape of the object can be transformed using a translation (`<move />`), a scaling (`<scale />`) or a rotation (`<rotate />`, `<rotateline />`). With the generation process creating particles at the nodes, different types of particles can be created; a fluid particle (`<setmkfluid />`), a boundary particle (`<setmkbound />`) or none (`<setmkvoid />`). Hence, **mk** is the marker value used to mark a set of particles with a common feature in the simulation. Note that values of the final **mk** are different to **mkfluid**, **mkboud** and **mkvoid** following the rules:

**mk** for boundaries = **mkboud** + 11

**mk** for fluid particles = **mkfluid** + 1

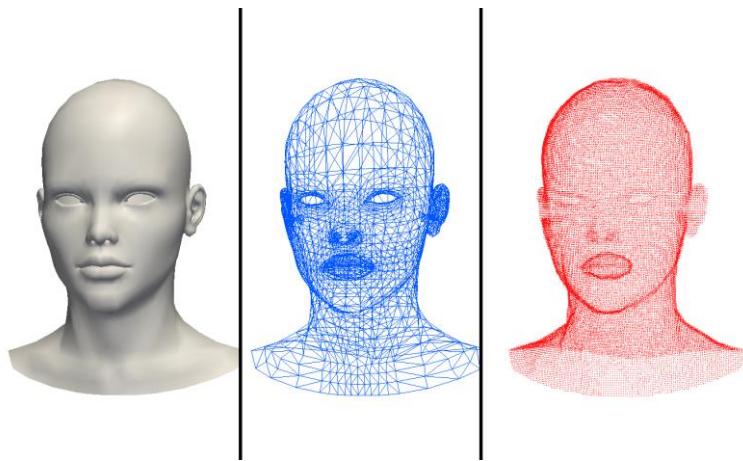
Particles can be created only at the object surface (`<setdrawmode mode="face" />`), or only inside the bounds of the objects (`<setdrawmode mode="solid" />`) or both (`<setdrawmode mode="full" />`).

The set of fluid particles can be labelled with features or special behaviours (`<initials />`). For example, initial velocity (`<velocity />`) can be imposed for fluid particles or a solitary

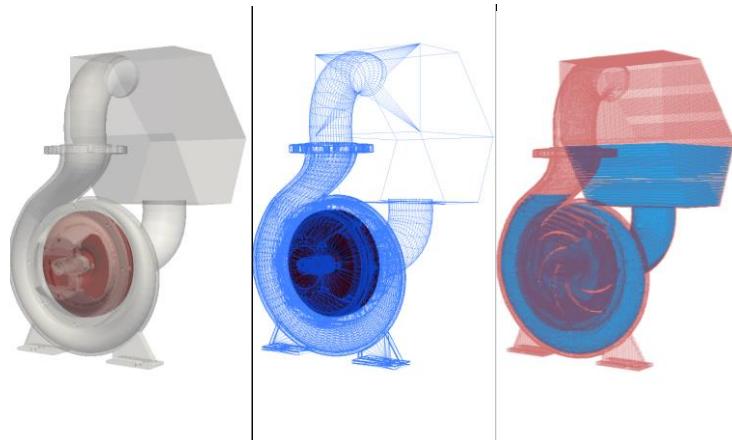
wave can be defined (`<velwave />`). Furthermore, particles can be defined as part of a floating object (`<floating />`).

Once boundaries are defined, filling a region with fluid particles can be easily obtained using the following commands: (`<fillpoint />`, `<fillbox />`, `<fillfigure />`, `<fillprism />`). This works also in the presence of arbitrarily complex geometries.

In cases with more complex geometries, external objects can be imported from 3DS files (Figure 9-3) or CAD files (Figure 9-4). This enables the use of realistic geometries generated by 3D designing application with the drawing commands of GenCase to be combined. These files (3DS or CAD) must be converted to STL format (`<drawfilestl />`), PLY format (`<drawfileply />`) or VTK format (`<drawfilevtk />`), formats that are easily loaded by **GenCase**. Any object in STL, PLY or VTK (*object.vtk*, *object.stl* or *object.ply* in Figure 9-1) can be split in different triangles and any triangle can be converted into particles using the **GenCase** code.



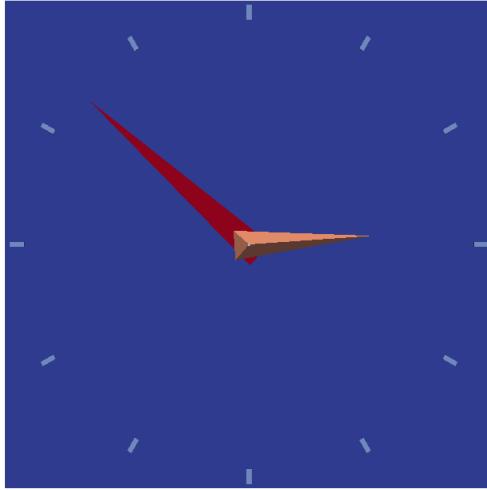
**Figure 9-3.** Example of a 3D file imported by **GenCase** and converted into particles.



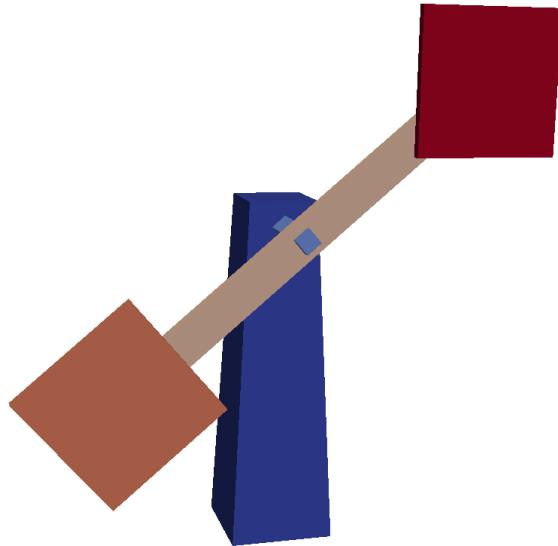
**Figure 9-4.** Example of a CAD file imported by **GenCase** and converted into particles.

Different kinds of movements can be imposed to a set of particles; linear, rotational, circular, sinusoidal, etc. To help users define movements, a directory with some examples is also included in the DualSPHysics package. Thus, the directory **MOTION** includes:

- *Motion01*: uniform rectilinear motion (`<mvrect />`) that also includes pauses (`<wait />`)
- *Motion02*: combination of two uniform rectilinear motion (`<mvrect />`)
- *Motion03*: movement of an object depending on the movement of another (hierarchy of objects)
- *Motion04*: accelerated rectilinear motion (`<mvrectace />`)
- *Motion05*: rotational motion (`<mvrot />`). See Figure 9-5.
- *Motion06*: accelerated rotation motion (`<mvrotace />`) and accelerated circular motion (`<mvcirace />`). See Figure 9-6.
- *Motion07*: sinusoidal movement (`<mvrectsinu />`, `<mvrotsinu />`, `<mvcirsinu />`)
- *Motion08*: prescribed with data from an external file (time , position) (`<mvfile />`)
- *Motion09*: prescribed with data from an external file (time , angle) (`<mvrotfile />`)



**Figure 9-5.** Example of rotational motion.



**Figure 9-6.** Example of accelerated rotation motion and accelerated circular motion.

## **TO RUN GENCASE:**

**example: GenCase Case\_Def Case [options]**

where *Case\_Def* is the name of the input file (Case\_Def.xml as seen in Figure 9-1) and *Case* will be the name of the output files (Case.xml and Case.bi4) and the *options* are:

### **-h**

Shows information about the different parameters. Typing “GenCase -h” in the command window generates a brief help manual (available in **HELP** folder).

### **-template**

Generates the example file *CaseTemplate.xml* that is already generated and saved in **HELP** folder.

### **-dp:<float>**

Defines the distance between particles. By varying this parameter the number of particles will be modified without changing any other data since all dimensions are given in global dimensions.

### **-ompthreads:<int>**

Indicates the number of threads by host for parallel execution, it takes the number of cores of the device by default (or using zero value).

### **-save:<values>**

Indicates the format of output files.

**+/-all:** To choose or reject all options

**+/-bi:** Binary format for the initial configuration (by default)

**+/-vtkall:** VTK with all particles of the initial configuration

**+/-vtkbound:** VTK with boundary particles of the initial configuration

**+/-vtkfluid:** VTK with fluid particles of the initial configuration

Note that when using –save:all, the files **Case\_All.vtk**, **Case\_Bound.vtk** and **Case\_Fluid.vtk** of Figure 9-1 are also generated and should be visualised to check the generated particles before launching the simulation.

### **-debug:<int>**

Debug level (-1:Off, 0:Explicit, n:Debug level)

### **Examples:**

GenCase4 citydef city

GenCase4 case case -dp:0.01

A more complete description of the code and the XML files can be found in **XML\_GUIDE.pdf available on the DualSPHysics website. This document helps to create a new case using the input XML file and all XML parameters are explained and related to SPH equations.**

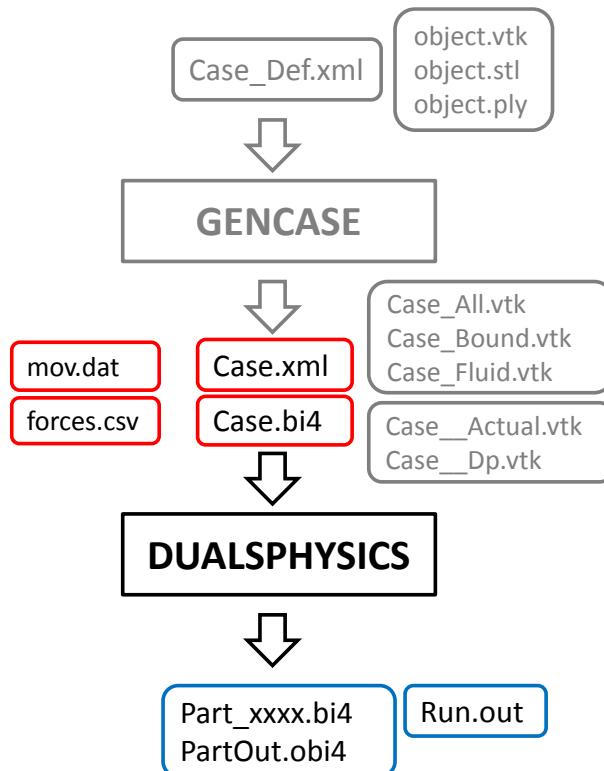
An example of input XML is shown here; the CaseWavemaker\_Def.xml included in **RUN\_DIRECTORY/CASEWAVEMAKER**:



## 10. Processing

The main code which performs the SPH simulation is named **DualSPHysics**.

The **input files** to run DualSPHysics code include one XML file (**Case.xml** in Figure 10-1) and a binary file (**Case.bi4** in Figure 10-1). **Case.xml** contains all the parameters of the system configuration and its execution such as key variables (smoothing length, reference density, gravity, coefficients to compute pressure starting from density, speed of sound...), the number of particles in the system, movement definition of moving boundaries and properties of moving bodies. The binary file **Case.bi4** contains the particle data; arrays of position, velocity and density and headers. The **output files** consist of binary format files with the particle information at different instants of the simulation (**Part0000.bi4**, **Part0001.bi4**, **Part0002.bi4** ...) file with excluded particles (**PartOut.obi4**) and text file with execution log (**Run.out**).



**Figure 10-1.** Input (red) and output (blue) files of DualSPHysics code.

Different parameters defined in the XML file can be changed using executions parameters of DualSPHysics: time stepping algorithm specifying Symplectic or Verlet (-**symplectic**, -**verlet[:steps]**), choice of kernel function which can be Cubic or Wendland (-**cubic**, -**wendland**), the value for artificial viscosity (-**viscoart: <float>**) or laminar+SPS viscosity treatment (-**viscolamsp: <float>**), activation of the Delta-SPH formulation (-**deltasph: <float>**), use of shifting algorithm (-**shifting: <mode>**) the maximum time of simulation and time intervals to save the output data (-**tmax**, -**tout**). To run the code, it is

also necessary to specify whether the simulation is going to run in CPU or GPU mode (**-cpu**, **-gpu[:id]**), the format of the output files (**-sv:[formats,...]**, **none**, **binx**, **ascii**, **vtk**, **csv**), that summarises the execution process (**-svres:<0/1>**) with the computational time of each individual process (**-svtimers:<0/1>**). It is also possible to exclude particles as being out of limits according to prescribed minimum and maximum values of density (**-rhopout:min:max**) or that travel further than maximum Z position (**-incz:<float>**).

For CPU executions, a multi-core implementation using OpenMP enables executions in parallel using the different cores of the machine. It takes the maximum number of cores of the device by default or users can specify it (**-ompthreads:<int>**). On the other hand, different cell divisions of the domain can be used (**-cellmode:<mode>**) that differ in memory usage and efficiency.

One of the novelties version 4 of DualSPHysics is the use of double precision in variables of position of the particles (**-posdouble:<mode>**) for the computation of particle interactions. The particle interaction is one of the most time-consuming parts of the simulation, hence the precision in this part can be controlled using the **-posdouble** parameter, which takes the following values:

- 0: particle interaction is performed using single precision for position variables ( $x, y, z$ )  
When “dp” is much smaller than size of the domain, the user is recommended to choose one of the following:
- 1: particle interaction is performed using double precision for position variables but final position is stored using simple precision
- 2: particle interaction is performed using double precision for position variables and final position is stored using double precision.

Other important novelty in v4.0 is the determination of the optimum BlockSize for the CUDA kernels that execute particle interaction (**-blocksize:<mode>**):

- Fixed (-blocksize:0): A fixed block size of 128 threads is used. This value does not always provides the maximum performance but it usually offers good performance for those type of kernels.
- Occupancy (-blocksize:1): Occupancy Calculator of CUDA is used to determine the optimum block size according to the features of the kernel (registers and shared memory however data used in the kernels are not considered). This option is available from CUDA 6.5
- Empirical (-blocksize:2): Here, data used in the CUDA kernels is also considered. The optimum BlockSize is evaluated every certain number of steps (500 by default). In this way, block size can change during the simulation according to input data.

## TO RUN DUALSPHYSICS:

### example: DualSPHysics Case [options]

where *Case* is the name of the input files (Case.xml & Case.bi4 as seen in Figure 10-1).

`$dualsphysics $dirout/$name $dirout -svres -cpu`

enables the simulation on the cpu, where `$dirout` is the directory with the file `$name.bi4`

`$dualsphysics $dirout/$name $dirout -svres -gpu`

enables the same simulation on the gpu.

`$dualsphysics $dirout/$name $dirout -svres -gpu -partbegin:69 dirbegin`

restarts the simulation from the time corresponding to files output Part0069.bi4 located in the directory `dirbegin`

The configuration of the execution is mostly defined in the XML file, but it can be also defined or changed using **execution parameters**. Furthermore, new options and possibilities for the execution can be imposed using **[options]**:

#### **-h**

Shows information about parameters. Typing “DualSPHysics –h” in the command window generates a brief help manual (available in **HELP** folder).

#### **-opt <file>**

Loads configuration from a file.

#### **-cpu**

Execution on CPU (option by default).

#### **-gpu[:id]**

Execution on GPU and id of the device.

#### **-stable**

Ensures the same results when a simulation is repeated since operations are always carried out in the same order.

#### **-posdouble:<mode>** Precision used in position for particle interactions

- 0** Use and store in single precision (option by default)
- 1** Use double precision but saves result in single precision
- 2** Use and store in double precision

#### **-ompthreads:<int>**

Only for CPU. Indicates the number of threads by host for parallel execution, this takes the number of cores of the device by default (or using zero value).

#### **-blocksize:<mode>**

Defines BlockSize to use in particle interactions on GPU

- 0** Fixed value (128) is used (option by default)
- 1** Optimum BlockSize indicated by Occupancy Calculator of CUDA
- 2** Optimum BlockSize is calculated empirically (option by default)

**-cellmode:<mode>**

Specifies the cell division mode, by default, the fastest mode is chosen

- 2h** lowest and the least expensive in memory
- h** fastest and the most expensive in memory

**-symplectic**

Symplectic algorithm as time step algorithm.

**-verlet[:steps]**

Verlet algorithm as time step algorithm and number of time steps to switch equations.

**-cubic**

Cubic spline kernel.

**-wendland**

Wendland kernel.

**-viscoart:<float>**

Artificial viscosity [0-1].

**-viscolamsp:<float>**

Laminar+SPS viscosity [order of 1E-6].

**-viscoboundfactor:<float>**

Multiplies the viscosity value of boundary.

**-deltasph:<float>**

Constant for DeltaSPH. Typical value is 0.1 (0 by default)

**-shifting:<mode>**

Specifies the use of Shifting correction

- none** Shifting is disabled (by default)
- nobound** Shifting is not applied near boundary
- nofixed** Shifting is not applied near fixed boundary
- full** Shifting is always applied

**-sv:[formats,...]**

Specifies the output formats:

- none** No particles files are generated
- binx** Binary files (option by default)
- info** Information about execution in .ibi4 format
- vtk** VTK files
- csv** CSV files

**-svres:<0/1>**

Generates file that summarises the execution process.

**-svtimers:<0/1>**

Obtains timing for each individual process.

**-svdomainvtk:<0/1>**

Generates VTK file with domain limits.

**-name <string>**

Specifies path and name of the case.

**-runname <string>**

Specifies name for case execution.

**-dirout <dir>**

Specifies the output directory.

**-partbegin:begin[:first] dir**

RESTART option. Specifies the beginning of the simulation starting from a given PART (begin) and located in the directory (dir), (first) indicates the number of the first PART to be generated.

**-incz:<float>**

Allows increase in Z+ direction of the computational domain. Case domain is fixed as function of the initial particles, however the maximum Z position can be increased with this option in case particles reach higher locations.

**-rhopout:min:max**

Excludes fluid particles out of these density limits.

**-ftpause:<float>**

Time to start floating bodies movement. By default 0.

**-tmax:<float>**

Maximum time of simulation.

**-tout:<float>**

Time between output files.

**-domain\_particles[:xmin,ymin,zmin,xmax,ymax,zmax]**

The domain is fixed as a function of the initial article positions and modified for xmin,...

**-domain\_particles\_prc:xmin,ymin,zmin,xmax,ymax,zmax**

The values in proportion with the case dimensions according to the initial particles.

**-domain\_fixed:xmin,ymin,zmin,xmax,ymax,zmax**

The domain is fixed with the specified values.

**Examples:**

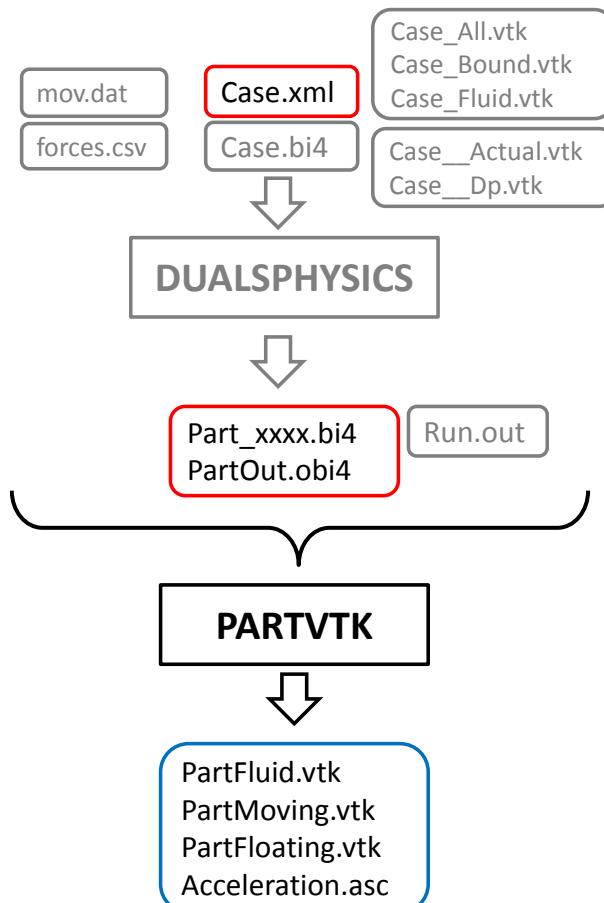
DualSPHysics4 case out\_case -sv:binx,csv



## 11. Post-processing

### 11.1 Visualization of particle output data

The **PartVTK** code is used to convert the output binary files of DualSPHysics into different formats that can be visualised and /or analysed. Thus, the output files of DualSPHysics, the binary files (.bi4), are now the **input files** for the post-processing code **PartVTK**.



**Figure 11-1.** Input (red) and output (blue) files of PartVTK code.

The **output files** can be VTK-binary (`-savevtk`), CSV (`-savecsv`) or ASCII (`-saveascii`). In this way the results of the simulation can be plotted using Paraview, gnuplot, Octave, etc.... For example; PartVtkBin\_0000.vtk,... These files can be generated by selecting a set of particles defined by **mk** (`-onlymk:`), by the **id** of the particles (`-onlyid:`), by the type of the particle (`-onlytype:`), by the position of the particles (`-onlypos:` and `-onlyposfile`) or by the limits of velocity of the particles (`-onlyvel:`), so we can check or uncheck all the particles (`+/-all`), the boundaries (`+/-bound`), the fixed boundaries (`+/-fixed`), the moving boundaries (`+/-moving`), the floating bodies (`+/-floating`) or the fluid particles (`+/-fluid`). The output files can contain different particle data (`-vars:`); all the physical quantities (`+/-all`), velocity (`+/-vel`), density (`+/-rhop`), pressure (`+/-press`), mass (`+/-mass`), volume (`+/-vol`), acceleration (`+/-ace`), vorticity (`+/-vor`), the id of the particle (`+/-idp`), the mk of the

particle (+/-**mk**) and the type (+/-**type**:). The user can define new variables in DualSPHysics and make reference to those in PartVTK using **-vars:NewVar** or **-vars:all**.

## TO RUN PARTVTK:

### example:

**PartVTK -savevtk PartFluid.vtk -onlytype:-fluid [options]**

#### Basic options:

**-h**

Shows information about parameters. Typing “PartVTK –h” in the command window generates a brief help manual (available in **HELP** folder).

**-opt <file>**

Loads configuration from a file.

#### Define input file:

**-dirin <dir>**

Indicates the directory with particle data.

**-casein <casename>**

Name of case file with particle data.

**-filexml file.xml**

Loads xml file with information of mk and type of particles, this is needed for the filter **-onlymk** and for the variable **-vars:mk**.

**-first:<int>**

Indicates the first file to be computed.

**-last:<int>**

Indicates the last file to be computed.

**-files:<values>**

Indicates the number of files to be processed.

**-move:x:y:z**

Particles are moved using this offset.

**-threads:<int>**

Indicates the number of threads for parallel execution of the interpolation, it takes the number of cores of the device by default (or uses zero value).

#### Define parameters for acceleration or vorticity calculation:

**-viscoart:<float>**

Artificial viscosity [0-1].

**-viscolam:<float>**

Laminar viscosity [order of 1E-6].

**-gravity:<float:float:float>**

Gravity value.

**-distinter\_2h:<float>**

Coefficient of  $2h$  that defines the maximum distance for the interaction among particles depending on  $2h$  (default value = 1.0).

**-distinter:<float>**

Defines the maximum distance for the interaction among particles in an absolute way.

**Define output file:**

**-savevtk <file.vtk>**

Generates vtk (polydata) files with particles according to the filters with options onlymk, onlyid and onlytype.

**-saveascii <file.asc>**

Generates ASCII files without headers.

**-savecsv <file.csv>**

Generates CSV files to use with calculation sheets.

**-savestatscsv<file.csv>**

Generates CSV files with statistics.

**Configuration for each output file:**

**-onlypos:xmin:ymin:zmin:xmax:ymax:zmax**

Indicates limits of particles.

**-onlyposfile filters.xml**

Indicates XML file with filters to apply.

**-onlyvel:vmin:vmax**

Indicates the velocity of selected particles.

**-onlymk:<values>**

Indicates the mk of selected particles.

**-onlyid:<values>**

Indicates the id of selected particles.

**-onlytype:<values>**

Indicates the type of selected particles:

(+ means include, - means do not include)

**+/-all:** To choose or reject all options

**+/-bound:** Boundary particles (fixed, moving and floating)

**+/-fixed:** Boundary fixed particles

**+/-moving:** Boundary moving particles

**+/-floating:** Floating body particles

**+/-fluid:** Fluid particles (no excluded)

(Preselected types: all)

**-vars:<values>**

Indicates the variables to be computed and stored:

(+ means include, - means do not include)

**+/-all:** To choose or reject all options

**+/-idp:** Idp of particles

**+/-vel:** Velocity

**+/-rhop:** Density

**+/-press:** Pressure

**+/-mass:** Mass

**+/-vol:** Volume

**+/-type:** Type (fixed, moving, floating, fluid)

**+/-mk:** Value of mk associated to the particles

**+/-ace:** Acceleration

**+/-vor:** Vorticity

**+/-XXX:** Variable XXX defined by the user

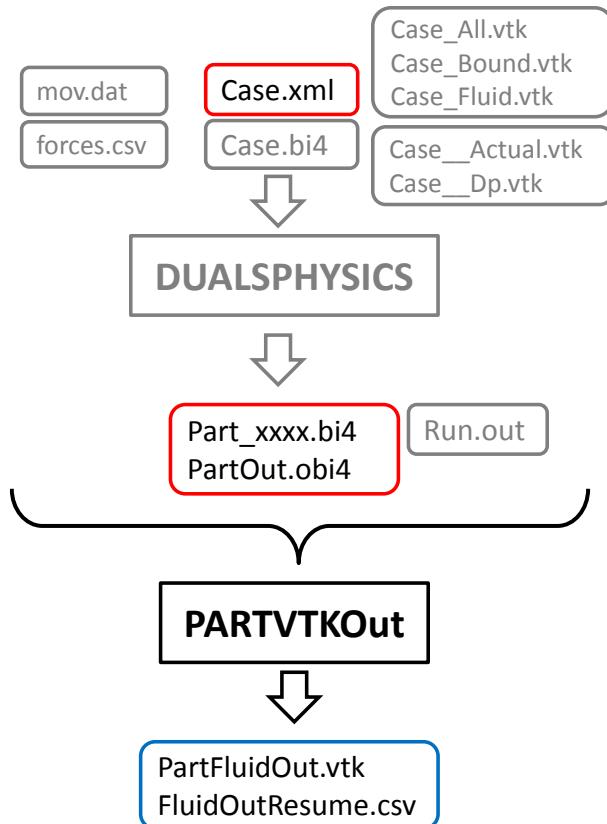
(Preselected variables: type)

**Examples:**

PartVtk4 -savevtk partfluid.vtk -onlytype:-bound

-savevtk partbound.vtk -onlytype:-all,bound -vars:vel.x,mk,var1

In addition, the **PartVTKOut** code is used to generate files with the particles that were excluded from the simulation (stored in **PartOut.obi4**). The output file of DualSPHysics, **PartOut.obi4** is the **input** file for the post-processing code **PartVTKOut**. Information with excluded particles can be stored in CSV files (**-savecsv**:  
**-SaveResume**) and VTK (**-savevtk:**) can be generated with those particles.



**Figure 11-2.** Input (red) and output (blue) files of PartVTKOut code.

Particles can be excluded from the simulation for three reasons:

- POSITION: Limits of the domain are computed starting from particles that were created in **GenCase**. Note that these limits are different from **pointmin** and **pointmax** defined in section *<definition>* of the input XML and can be also changed by the user when executing **DualSPHysics** code. The actual limits of the domain can be seen in Run.out: MapRealPos(final). Therefore, when one particle moves beyond those limits, the particle is excluded. Only in the Z+ direction, can particles move to higher positions according to parameter *IncZ.*, where new cells are created to contain the particles.
  - DENSITY: Valid values of particle density are between *RhopOutMin* (default=700) and *RhopOutMax* (default=1300), but the user can also change those values.
  - VELOCITY: One particle can be also removed from the system when its displacement exceeds  $0.9 * S_{cell}$  during one time step (*S<sub>cell</sub>* is the size of the cell).

## **TO RUN PARTVTKOut:**

**example:**

**PartVTKOut -savevtk PartFluidOut.vtk csv [options]**

### **Basic options:**

**-h**

Shows information about parameters. Typing “PartVTKOut –h” in the command window generates a brief help manual (available in **HELP** folder).

**-opt <file>**

Loads configuration from a file.

### **Define input file:**

**-dirin <dir>**

Indicates the directory with particle data.

**-filexml file.xml**

Loads xml file with information of mk to save value of mk.

**-first:<int>**

Indicates the first file to be computed.

**-last:<int>**

Indicates the last file to be computed.

**-files:<values>**

Indicates the number of files to be processed.

### **Define output file:**

**-savevtk <file.vtk>**

Generates vtk(polydata) files with excluded particles.

**-savecsv <file.csv>**

Generates CSV file with particles info.

**-SaveResume <file.csv>**

Generates CSV file with resume info.

### **Configuration for output file:**

**-onlypos:xmin:ymin:zmin:xmax:ymax:zmax**

Indicates limits of particles.

**-onlynew**

Stores only new excluded particles of each PART file (default value = false)

**-limitpos:xmin:ymin:zmin:xmax:ymax:zmax**

Changes limits of simulation.

**-limitrhop:min:max**

Changes limits of rhop values.

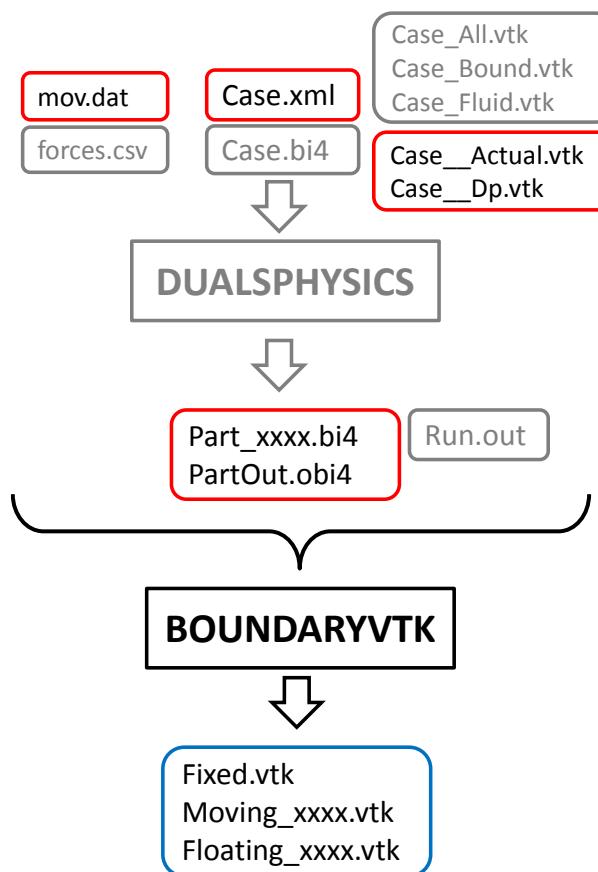
### **Examples:**

PartVtkOut4 -savevtk out.vtk

## 11.2 Visualization of boundaries

In order to visualise the boundary shapes formed by the boundary particles, different geometry files can be generated using the **BoundaryVTK code**. The code creates triangles or planes to represent the boundaries.

As **input data**, shapes can be loaded from a VTK file (`-loadvtk`), a PLY file (`-loadply`) or an STL file (`-loadstl`) while boundary movement can be imported from an XML file (`-loadxml file.xml`) using the timing of the simulation (`-motiontime`) or with the exact instants of output data (`-motiondatatime`). The movement of the boundaries can also be determined starting from the particle positions (`-motiondata`). The **output files** consist of VTK files (`-savevtk`), PLY files (`-saveply`) or STL files (`-savestl`) with the loaded information and the moving boundary positions at different instants. For example the output files can be named `motion_0000.vtk`, `motion_0001.vtk`, `motion_0002.vtk`... These files can be also generated by only a selected object defined by **mk** (`-onlymk:`), by the **id** of the object (`-onlyid:`).



**Figure 11-3.** Input (red) and output (blue) files of BoundaryVTK code.

## **TO RUN BOUNDARYVTK:**

**example:**

**BoundaryVTK -loadvtk bound.vtk -savevtk box.vtk -onlymk:10 [options]**

### **Basic options:**

**-h**

Shows information about parameters. Typing “BoundaryVTK –h” in the command window generates a brief help manual (available in **HELP** folder).

**-opt <file>**

Loads configuration from a file.

**-info**

Shows information of loaded data.

### **Load shapes:**

**-loadvtk <file.vtk>**

Load shapes from vtk files (PolyData).

**-onlymk:<values>**

Indicates the mk of the shapes to be loaded, affects to the previous -loadvtk and cannot be used with –onlyid.

**-onlyid:<values>**

Indicates the code of object of the shapes to be loaded, only affects to the previous –loadvtk instruction.

**-changemk:<value>**

Changes the mk of the loaded shapes to a given value, only affects to the previous –loadvtk instruction.

**-loadply:<mk> <file.ply>**

Load shapes from ply files with indicated mk.

**-loadstl:<mk> <file.stl>**

Load shapes from stl files with indicated mk.

### **Load configuration for a mobile boundary:**

**-filexml file.xml**

Loads xml file with information of the movement and type of particles.

**-motiontime:<time>:<step>**

Configures the duration and time step to simulate the movement defined in the xml file, cannot be used with -motiondatatime and –motiondata.

**-motiondatatime <dir>**

Indicates the directory where real times of the simulation are loaded for the mobile boundary cannot be used with -motiontime and –motiondata.

**-motiondata <dir>**

Indicates the directory where position of particles are loaded to generate the mobile boundary (cannot be used with -motiontime and –motiondatatime). This should be used with floatings or piston of automatic wave generation.

**Define output files:**

**-savemotion <file>**

Generates file with movement of the objects

**-savevtk <file.vtk>**

Generates vtk(polydata) files with the loaded shapes.

**-saveply <file.ply>**

Generates ply file with the loaded information.

**-savestl <file.stl>**

Generates stl file with the loaded information.

**-savevtkdata <file.vtk>**

Generates vtk(polydata) file with the loaded shapes including mk and shape code.

**-onlymk:<values>**

Indicates the mk value of the shapes to be stored, affects to the previous out option and cannot be used with -onlyid.

**-onlyid:<values>**

Indicates the code of the object of the shapes to be stored, affects to the previous out option.

**-filemove <file>**

Loads file with displacement.

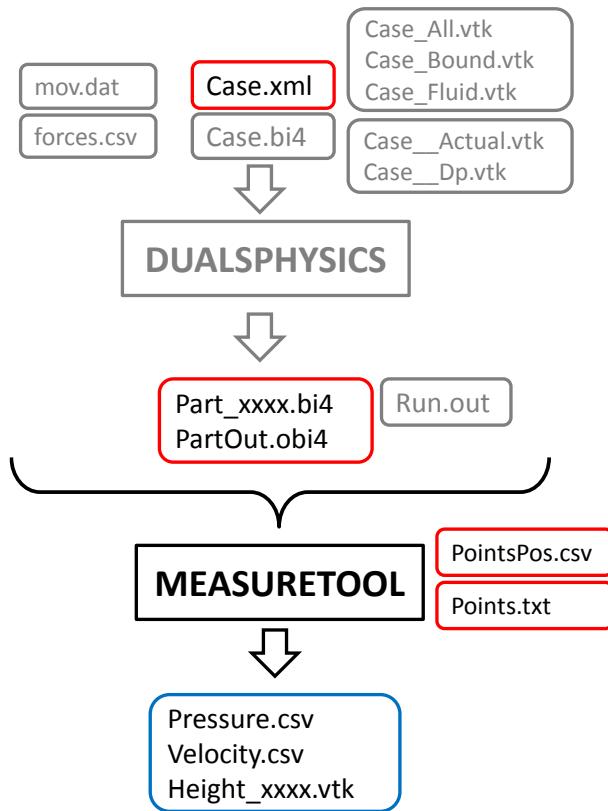
**Examples:**

BoundaryVtk4 -loadvtk bound.vtk -savevtk box.vtk -onlymk:10,12

BoundaryVtk4 -loadvtk bound.vtk -filexml case.xml -motiondata . -saveply motion.ply

### 11.3 Analysis of numerical measurements

To compare experimental and numerical values, a tool to analyse these numerical measurements is needed. The **MeasureTool** code allows different physical quantities at a set of given points to be computed. The binary files (.bi4) generated by DualSPHysics are the **input files** of the **MeasureTool** code and the **output files** are again VTK-binary or CSV or ASCII. The numerical values are computed by means of an SPH interpolation of the values of the neighbouring particles around a given position (see also PostprocessingCalculations.pdf).



**Figure 11-4.** Input (red) and output (blue) files of MeasureTool code.

The interpolation is computed using the Wendland kernel. Kernel correction is also applied when the summation of the kernel values around the position is higher than a value (**-kclimit:**) defining a dummy value if the correction is not applied (**-kedummy:**). The positions where the interpolation is performed are given in a text file for fixed position (**-points <file>**) or with position that changes in time (**-pointspos <file>**). Variables can be also computed at the position of existing particles with a given mk (**-particlesmk:**) or by indicating their id (**-particleid:**). The distance of interpolation can be  $2h$  (the size of the kernel) or can be changed (**-distinter\_2h:**, **-distinter:**). The interpolation is carried out using a selected set of particles, so the same commands for PartVTK can be used (**-onlymk:**, **-onlyid:**, **-onlytype:**, **-onlypos:**, **-onlyposfile**). Different interpolated variables (**-vars**) can be numerically calculated; all available ones (**+/-all**), velocity (**+/-vel**), density (**+/-rhop**), pressure (**+/-press**), mass (**+/-mass**), volume (**+/-vol**), id (**+/-idp**), vorticity (**+/-vor**),

acceleration (+/-**ace**) the summation of the kernel multiplied by volume (+/-**kcorr**), and variables defined by the user (+/-**XXX**). The maximum water depth can be also computed. Height values (-**height:**) are calculated according to the interpolated mass, if the nodal mass is higher than a given reference mass, that Z-position will be considered as the maximum height. The reference value can be calculated in relation to the mass values of the selected particles (-**height:0.5**, half the mass by default in 3D and -**height:0.4** in 2D) or can be given in an absolute way (-**heightlimit:**).

## TO RUN MEASURETOOL:

### example:

**MeasureTool -points points.txt -vars:press -savecsv Press [options]**

#### Basic options:

**-h**

Shows information about parameters. Typing “MeasureTool –h” in the command window generates a brief help manual (available in **HELP** folder).

**-opt <file>**

Loads configuration from a file.

#### Define input file:

**-dirin <dir>**

Indicates the directory with particle data.

**-casein <casename>**

Name of case file with particle data.

**-filexml file.xml**

Loads xml file with information of mk and type of particles, this is needed for the filter –onlymk and for the variable -vars:mk

**-first:<int>**

Indicates the first file to be computed.

**-last:<int>**

Indicates the last file to be computed.

**-files:<values>**

Indicates the number of files to be processed.

**-threads:<int>**

Indicates the number of threads for parallel execution of the interpolation, it takes the number of cores of the device by default (or using zero value).

#### Define parameters for acceleration or vorticity calculation:

**-viscoart:<float>**

Artificial viscosity [0-1].

**-viscolam:<float>**

Laminar viscosity [order of 1E-6].

**-gravity:<float:float:float>**

Gravity value.

**Set the filters that are applied to particles for interpolation:**

**-onlypos:xmin:ymin:zmin:xmax:ymax:zmax**

Indicates limits of particles.

**-onlyposfile filters.xml**

Indicates XML file with filters to apply.

**-onlymk:<values>**

Indicates the mk of selected particles.

**-onlyid:<values>**

Indicates the id of selected particles.

**-onlytype:<values>**

Indicates the type of selected particles:

**+/-all:** To choose or reject all options

**+/-bound:** Boundary particles (fixed, moving and floating)

**+/-fixed:** Boundary fixed particles

**+/-moving:** Boundary moving particles

**+/-floating:** Floating body particles

**+/-fluid:** Fluid particles (no excluded)

(by default: all, + means include, - means do not include)

**Set the configuration of interpolation:**

**-pointstemplate**

Creates example file to be used with -points or -pointspos.

**-points <file>**

Defines the points where interpolated data will be computed (each value separated by space or a new line).

**-pointspos <file>**

Defines the position of points where interpolated data will be computed (each line contains the points for one part file, points are defined as x0;y0;z0;x1;y1;z1...).

**-particlesmk:<values>**

Indicates the points where interpolated data will be computed using the positions of the particles with the given mk.

**-particlesid:<values>**

Indicates the points where interpolated data will be computed using the positions of the particles with the given id.

**-kclimit:<float>**

Defines the minimum value of sum\_wab\_vol to apply the Kernel Correction (default value = 0.5).

**-kcdummy:<float>**

Defines the dummy value for the interpolated quantity if Kernel Correction is not applied (default value = 0)

**-kcusedummy:<0/1>**

Defines whether or not to use the dummy value (default value = 1).

**-distinter\_2h:<float>**

Coefficient of  $2h$  that defines the maximum distance for the interaction among particles depending on  $2h$  (default value = 1.0).

**-distinter:<float>**

Defines the maximum distance for the interaction among particles in an absolute way.

**Set the values to be calculated:**

**-vars:<values>**

Defines the variables or magnitudes that are going to be computed as an interpolation of the selected particles around a given position:

(+ means include, - means do not include)

**+/-all:** To choose or reject all options

**+/-vel:** Velocity

**+/-rhop:** Density

**+/-press:** Pressure

**+/-mass:** Mass

**+/-vol:** Volume

**+/-idp:** Id of particles

**+/-ace:** Acceleration

**+/-vor:** Vorticity

**+/-kcorr:** Summation of interaction kernel multiplied by volume

**+/-XXX:** Defined variable for user

(by default: vel,rhop)

**-height[:<float>]**

Height value is calculated starting from mass values for each point  $x,y$ . The reference mass to obtain the height is calculated according to the mass values of the selected particles; by default 0.5 in 3D (half the mass) and 0.4 in 2D.

**-heightlimit:<float>**

The same as -height but the reference mass is given by an absolute value.

**Define output files format:**

**-savevtk <file.vtk>**

Generates VTK (polydata) with the given interpolation points.

**-savecsv <file.csv>**

Generates one CSV file with the time history of the obtained values.

**-saveascii <file.asc>**

Generates one ASCII file without headers with the time history of the obtained values.

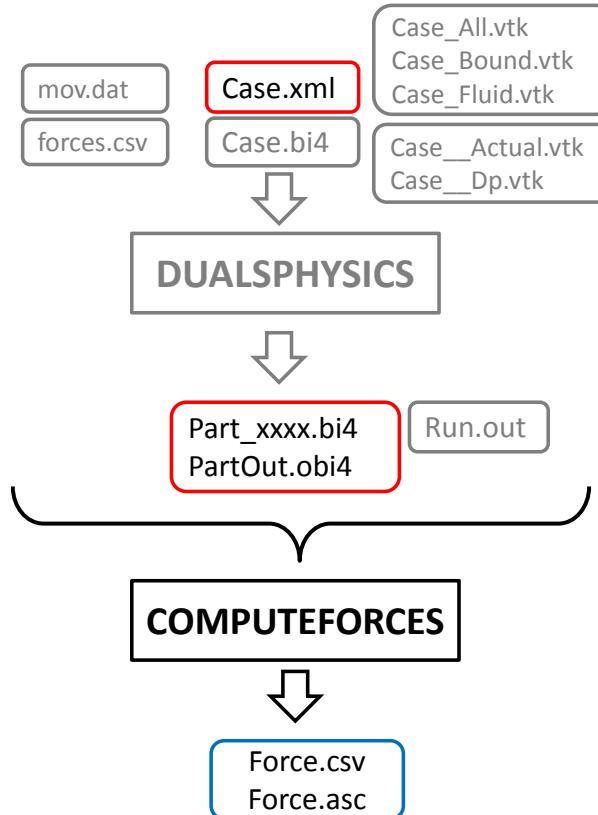
**Examples:**

MeasureTool4 -points fpoints.txt -onlytype:-all,fluid -savecsv dataf

MeasureTool4 -points fpoints.txt -vars:press -savevtk visupress

## 11.4 Force computation

The **ComputeForces** code is employed to compute the force exerted by the fluid onto a boundary object. The value of force is calculated as the summation of the acceleration values (solving the momentum equation) multiplied by the mass of each boundary particle (see also PostprocessingCalculations.pdf).



**Figure 11-5.** Input (red) and output (blue) files of ComputeForces code.

The momentum equation to solve the acceleration values is computed using the Wendland kernel. The distance of interpolation can be  $2h$  (the size of the kernel) or can be changed (**-distinter\_2h:**, **-distinter:**). The interpolation is carried out using a selected set of particles, so the same commands for PartVTK can be used (**-onlymk:**, **-onlyid:**, **-onlypos:**). The **output files** can be VTK-binary (**-savevtk**), CSV (**-savecsv**) or ASCII (**-saveascii**).

### TO RUN COMPUTEOFORCES:

**example:**

**ComputeForces -onlymk:[value] -savecsv Force [options]**

#### Basic options:

**-h**

Shows information about parameters. Typing “ComputeForces -h” in the command window generates a brief help manual (available in **HELP** folder).

**-opt <file>**

Loads configuration from a file.

**Define input file:**

**-dirin <dir>**

Directory with particle data.

**-filexml file.xml**

Loads xml file with information of mk and type of particles, this is needed for the filter -onlymk and for the variable -vars:mk.

**-first:<int>**

Indicates the first file to be computed.

**-last:<int>**

Indicates the last file to be computed.

**-files:<values>**

Indicates the number of files to be processed.

**-threads:<int>**

Indicates the number of threads for parallel execution of the interpolation, it takes the number of cores of the device by default (or uses zero value).

**Define parameters for force calculation:**

**-viscoart:<float>**

Artificial viscosity [0-1].

**-viscolam:<float>**

Laminar viscosity [order of 1E-6].

**-gravity:<float:float:float>**

Gravity value.

**-distinter\_2h:<float>**

Coefficient of  $2h$  that defines the maximum distance for the interaction among particles depending on  $2h$  (default value = 1.0).

**-distinter:<float>**

Defines the maximum distance for the interaction among particles in an absolute way.

**Configuration for each output file:**

**-onlypos:xmin:ymin:zmin:xmax:ymax:zmax**

Indicates limits of particles.

**-onlyposfile filters.xml**

Indicates XML file with filters to apply.

**-onlymk:<values>**

Indicates the mk of selected particles.

**-onlyid:<values>**

Indicates the id of selected particles.

**Define output file:**

**-savecsv <file.csv>**

Generates one CSV file with the time history of the obtained values.

**-saveascii <file.asc>**

Generates one ASCII file without headers with the time history of the obtained values.

**-savevtk <file.vtk>**

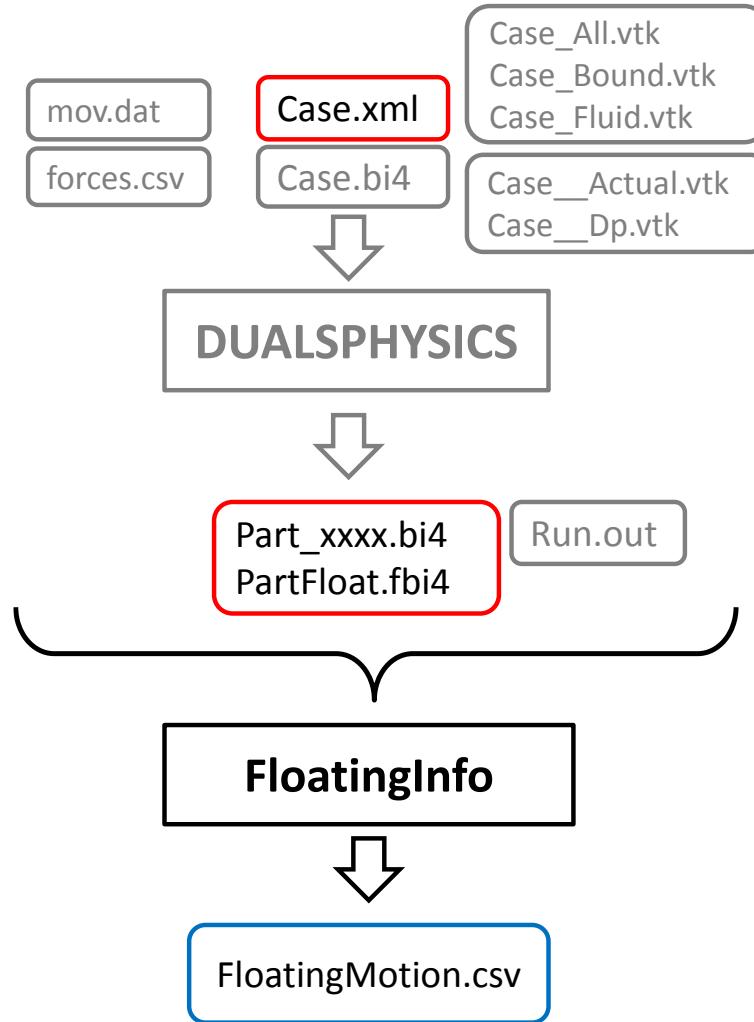
Generates VTK (polydata) with the given interpolation points.

**Examples:**

ComputeForces4 -onlymk:10 -savecsv results

## 11.5 Analysis of floating data

The **FloatingInfo** code is employed to obtain different data of the floating objects such as linear velocity, angular velocity, displacement of the center, motions and angles of rotation. The binary files (.bi4) generated by DualSPHysics and the file PartFloat.fbi4 are the **input files** of the **FloatingInfo** code and the **output files** are CSV files.



By default, the code always saves *time*, *fvel* (linear velocity), *fomega* (angular velocity), *center*. Motions and rotation can be also obtained (**-savemotion**) in 2D (*surge*, *heave*, *roll*) and in 3D (*surge*, *sway*, *heave*, *roll*, *pitch*, *yaw*). A CSV file will be created with the information of each floating in the simulation, but user can choose a given floating object (**-onlymk:**) or the name of the file (**-savedata**).

## **TO RUN FLOATINGINFO:**

**example:**

**FloatingInfo –onlymk:[value] -savedata CSVname [options]**

### **Basic options:**

**-h**

Shows information about parameters. Typing “FloatingInfo –h” in the command window generates a brief help manual (available in **HELP** folder).

**-opt <file>**

Loads configuration from a file.

### **Define input file:**

**-filexml file.xml**

Path and XML file with case of simulation.

**-first:<int>**

Indicates the first file to be computed.

**-last:<int>**

Indicates the last file to be computed.

**-files:<values>**

Indicates the number of files to be processed.

### **Define output file:**

**-savedata <file>**

Generates CSV file with data of floating.

**-savemotion**

Includes motion data.

**-onlymk:<values>**

Indicates the mk of selected floatings.

### **Examples:**

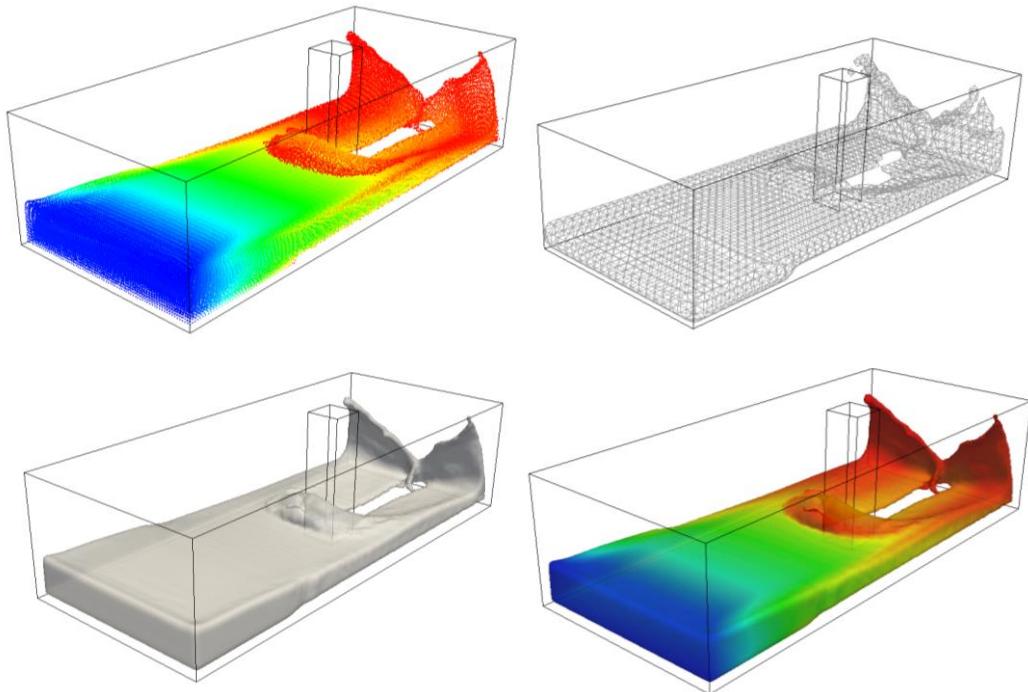
FloatingInfo4 -filexml case/floating.xml -savedata results -savemotion

FloatingInfo4 -filexml floating.xml -onlymk:61-65 -savemotion

## 11.6 Surface representation

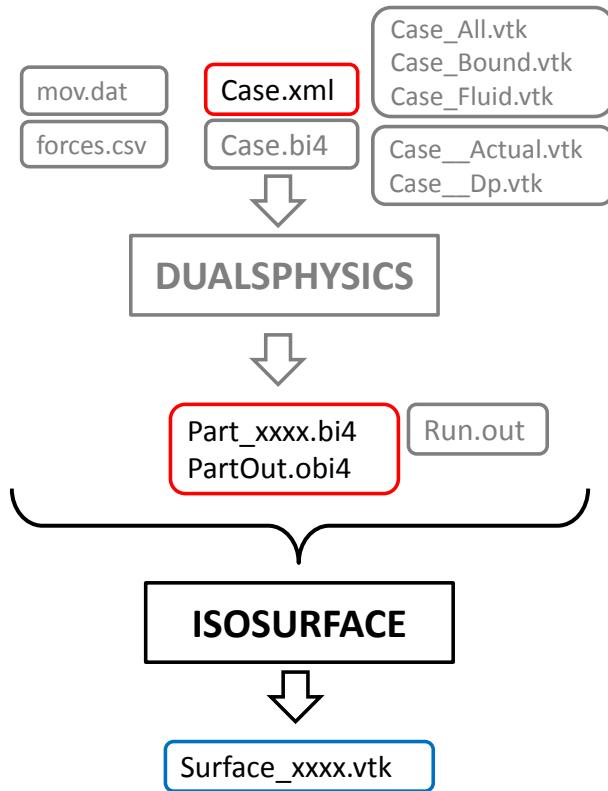
Using a large number of particles, the visualization of the simulation can be improved by representing surfaces instead of particles. To create the surfaces, the *marching cubes* algorithm is used [Lorensen and Cline, 1987]. This computer graphics technique extracts a polygonal mesh (set of triangles) of an isosurface from a 3-D scalar field.

Figure 11-6, represents a 3-D dam-break simulation using 300,000 particles. The first snapshot shows the particle representation. Values of mass are interpolated at the nodes of a 3-D Cartesian mesh that covers the entire domain using an SPH interpolation. Thus a 3-D mesh vertex that belongs to the free surface can be identified. The triangles of this surface (generated by means of the marching cubes algorithm) are represented in the second frame of the figure. The last snapshots correspond to the surface representation where the colour corresponds to the interpolated velocity value of the triangles.



**Figure 11-6.** Conversion of points to surfaces.

The output binary files of DualSPHysics are the **input files** of the IsoSurface code and the **output files** are VTK files (`-saveiso[:<var>]`) with the isosurfaces calculated using a variable (`<var>`) or can be structured points (`-savegrid`) with data obtained after the interpolation. The Cartesian mesh size can be defined by specifying the discretisation size using either the particle size `dp` (`-distnode_dp:`) or by specifying an absolute internode distance (`-distnode:`). On the other hand, the maximum distance for the interaction between particles to interpolate values on the nodes can be also defined depending on  $2h$  (`-distinter_dp:`) or in an absolute way (`-distinter:`). The particles to be considered to create the isosurface can be defined by the user using positions (`-onlypos & -onlyposfile`) or the limits of the isosurface can be indicated (`-iso_limits:`)



**Figure 11-7.** Input (red) and output (blue) files of IsoSurface code.

## TO RUN ISOSURFACE:

**example:**

**IsoSurface -saveiso fileiso [options]**

### Basic options:

**-h**

Shows information about parameters. Typing “IsoSurface –h” in the command window generates a brief help manual (available in **HELP** folder).

**-opt <file>**

Loads configuration from a file.

### Define input file:

**-dirin <dir>**

Indicates the directory with particle data.

**-casein <casename>**

Name of case file with particle data.

**-filexml file.xml**

Loads xml file with information of mk and type of particles, this is needed for the filter –onlymk and for the variable -vars:mk

**-first:<int>**

Indicates the first file to be computed.

**-last:<int>**

Indicates the last file to be computed.

**-files:<values>**

Indicates the number of files to be processed.

**-move:x:y:z**

Particles are moved using this offset

**-threads:<int>**

Indicates the number of threads for parallel execution of the interpolation, it takes the number of cores of the device by default (or using zero value).

**Define input data:****-onlypos:xmin:ymin:zmin:xmax:ymax:zmax**

Indicates limits of particles.

**-onlyposfile filters.xml**

Indicates XML file with filters to apply.

**-onlymk:<values>**

Indicates the mk of selected particles.

**-onlyid:<values>**

Indicates the id of selected particles.

**-onlytype:<values>**

Indicates the type of selected particles:

(+ means include, - means do not include)

**+/-all:** To choose or reject all options

**+/-bound:** Boundary particles (fixed, moving and floating)

**+/-fixed:** Boundary fixed particles

**+/-moving:** Boundary moving particles

**+/-floating:** Floating body particles

**+/-fluid:** Fluid particles (no excluded)

(Preselected types: fluid)

**-vars:<values>**

Indicates the variables to be computed and stored

(+ means include, - means do not include)

**+/-all:** To choose or reject all options

**+/-idp:** Id of particles

**+/-vel:** Velocity

**+/-rhop:** Density

**+/-press:** Pressure

**+/-mass:** Mass

**+/-vol:** Volume

**+/-type:** Type (fixed, moving, floating, fluid)

**+/-mk:** Value of mk associated to the particles

**+/-ace:** Acceleration

**+/-vor:** Vorticity

**+/-XXX:** Defined variable for user

(by default: vel, rhop)

**Define parameters for acceleration or vorticity calculation:****-viscoart:<float>**

Artificial viscosity [0-1].

**-viscolam:<float>**

Laminar viscosity [order of 1E-6].

**-gravity:<float:float:float>**

Gravity value.

**Set the configuration of interpolation:**

**-distinter\_2h:<float>**

Coefficient of 2h that defines the maximum distance for the interaction among particles depending on 2h (default value = 1.0).

**-distinter:<float>**

Defines the maximum distance for the interaction among particles in an absolute way.

**-kclimit:<float>**

Defines the minimum value of sum\_wab\_vol to apply the Kernel Correction (default value = 0.05).

**-kcdummy:<float>**

Defines the dummy value for the interpolated quantity if Kernel Correction is not applied (default value = 0)

**-kcusedummy:<0/1>**

Defines whether or not to use the dummy value (default value = 1).

**Set the configuration of isosurface:**

**-iso\_limits:xmin:ymin:zmin:xmax:ymax:zmax**

Isosurface limits are adjusted to the given limits

**-distnode\_dp:<float>**

Defines the distance between nodes by multiplying dp and the given value (option by default)

**-distnode:<float>**

Distance between nodes is given

**Define output files:**

**-saveiso <file.vtk>**

Generates VTK files (polydata) with the isosurface starting from a variable mass and using the limit value. When limit value is not given, the threshold mass value of the fluid is considered (0.5\*fluid particle mass)

**-saveiso:var:<values> <file.vtk>**

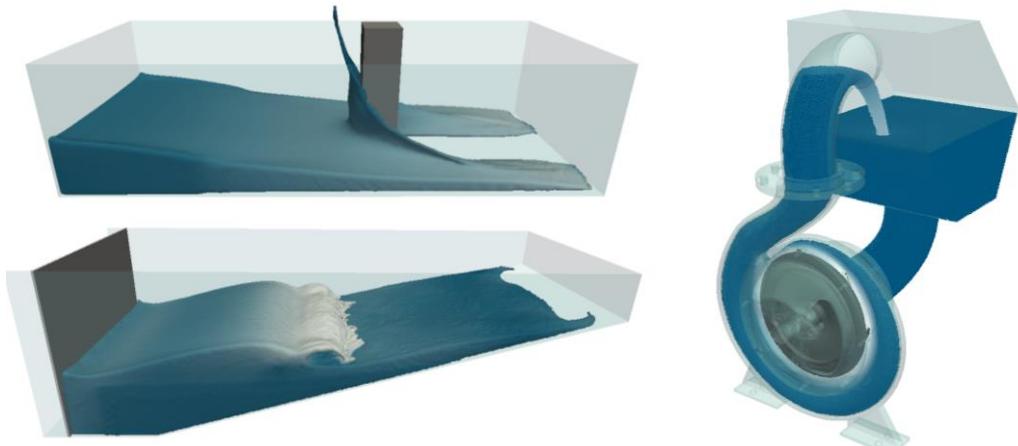
Generates VTK files (polydata) with the isosurface calculated starting from a variable given and using the limit value.

**Examples:**

IsoSurface4 –saveiso fileiso

## 12. Testcases

Some demonstration cases are included in the DualSPHysics package (Figure 12.1); such as a dam break interacting with an obstacle in a tank, waves on a beach created by a piston or the real geometry of a pump mechanism that translates water within a storage tank.



**Figure 12-1.** Test cases: dam break, wave maker, and pump.

The different test cases are described in the following sections. The scripts (*.bat* in Windows and *.sh* in Linux) are presented and the different command line instructions to execute the different programs are described. The XML files are also explained in detail to describe the execution parameters and the different labels to plot boundaries, fluid volumes, movements, etc.

For each case there are 4 options;

- Case\_win\_CPU.bat* (windows on CPU)
- Case\_win\_GPU.bat* (windows on GPU)
- Case\_linux\_CPU.sh* (linux on CPU)
- Case\_linux\_GPU.sh* (linux on GPU)

Since many 3-D applications can be reduced to a 2-D problem, the executable of DualSPHysics can also perform 2-D simulations, however this option is not fully optimized. A 2-D simulation can be achieved easily by imposing the same values in the Y-direction (`<pointmin> = <pointmax>` in the input XML file). Thus, the initial configuration is a 2-D system and DualSPHysics will detect automatically that the simulation must be carried out using the 2-D formulation.

The different CASES provided are intended as worked examples for users to create their own cases, to learn the different available options and what postprocessing tools can be used. The following list summarises the testcases that can be found in **RUN\_DIRECTORY**:

## 1\_CASEDAMBREAK

- 3-D dam break flow impacting on a structure: numerical velocity, pressure and force are computed.
- 2-D dam break and validation data from [[Koshizuka and Oka, 1996](#)] experiment.

## 2\_CASEPERIODICITY

- 2-D case with Periodicity in X direction.
- Delta-SPH is also used.

## 3\_CASEMOVINGSQUARE

- 2-D case with square that moves with rectilinear movement.
- Example of no gravity so parameter “b” needs to be specified by the user.
- Shifting is used for this internal flow (no need to detect free surface).

## 4\_CASEFORCES

- External acceleration is loaded from a file and applied to two different volumes of fluid.
- Delta-SPH is also used.

## 5\_CASESLOSHING

- 2-D sloshing tank that reads external acceleration from a file.
- The same 2-D tank but instead of reading externally imposed accelerations, the CASE reads the rotational movement of the tank itself from a file.
- Validation with [SPHERIC Benchmark #10](#) where pressure is computed.

## 6\_CASEWAVEMAKER

- 3-D tank with Periodicity in Y direction and piston with sinusoidal movement.  
Delta-SPH and Shifting are used.
- 2-D tank with piston motion loaded from external file and external structure (STL).  
Validation data from CIEMito experiment: numerical computation of wave surface elevation and force exerted onto the wall.

## 7\_CASEWAVEGENERATION

- 2-D automatic generation of regular waves ( $H,T,d$ ) and comparison with 2<sup>nd</sup> order wave theory.
- 2-D automatic generation of irregular waves ( $H_s,T_p,d$ ) and comparison with 2<sup>nd</sup> order wave theory.

## 8\_CASEFLOATING

- 3-D floating box in a wave tank with Periodicity in Y direction and piston with sinusoidal movement. Delta-SPH is used.
- 2-D floating box under action of non-linear waves in a tank with flap that reads rotational motion from an external file and uses laminar+SPS viscosity. Validation data from [[Hadzic et al., 2005](#)].
- 2-D falling sphere that uses laminar+SPS viscosity. Validation data from [[Fekken, 2004](#)] and [[Moyo and Greenhow, 2000](#)].

## 9\_CASEPUMP

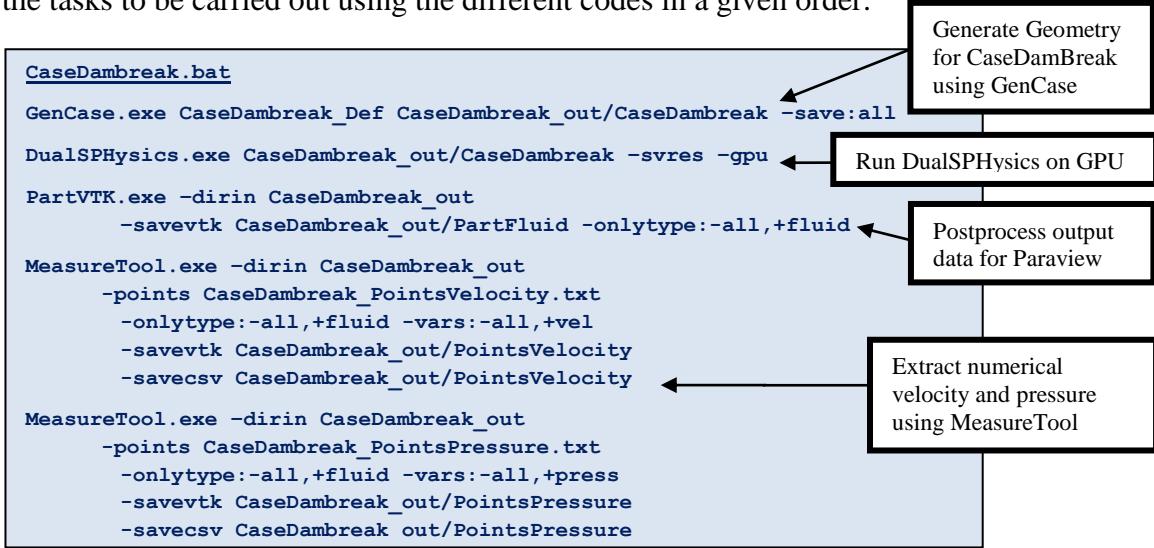
- 3-D external geometries are imported (STL) and rotational movement is imposed.

## 10\_CASEDEM

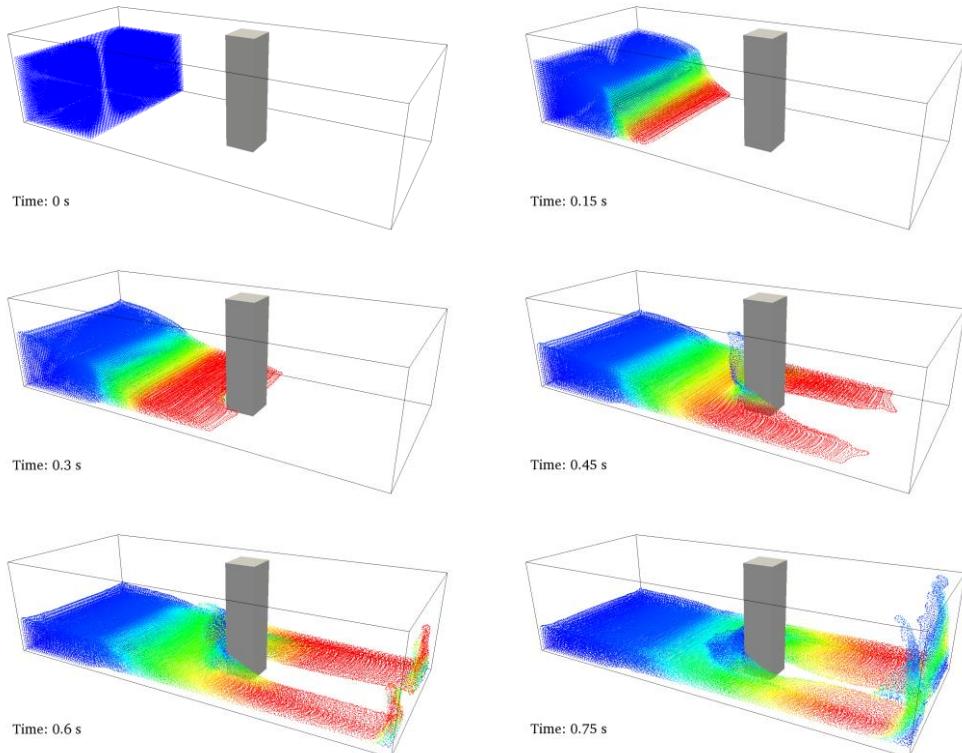
- 2-D case only with DEM of a ball that impacts with blocks. Example without fluid particles.
- 3-D dam break and blocks where interaction between blocks and with walls used DEM and properties of materials. Delta-SPH is also used.

## 12.1 CASEDAMBREAK

The first test case consists of a 3-D dam-break flow impacting with a structure inside a numerical tank. No moving boundary particles are involved in this simulation and no movement is defined. *CaseDambreak.bat* (*CaseDambreak.sh* in linux) file summarises the tasks to be carried out using the different codes in a given order:

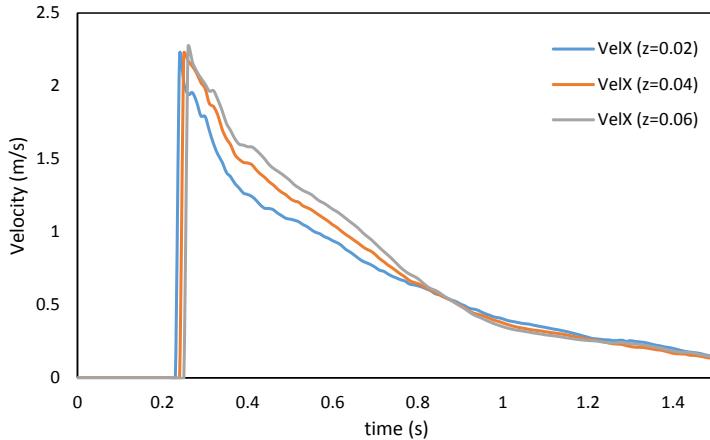


Different instants of the CaseDambreak simulation are shown in Figure 12-2 where the *PartFluid\_xxxx.vtk* of the fluid particles, the *CaseDambreak\_Box\_Dp.vtk* and *CaseDambreak\_Building\_Dp.vtk* are depicted.



**Figure 12-2.** Instants of the CaseDambreak simulation visualized using Paraview.  
Colour represents velocity of the particles.

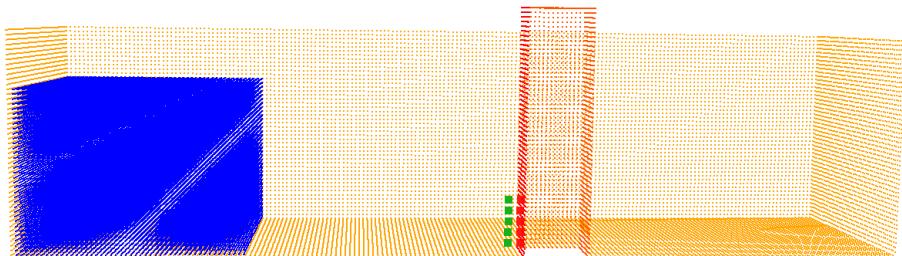
Using **MeasureTool** code, numerical velocities have been computed in the list of points described in the file *PointsVelocity.txt*, which are (0.754, 0.31, 0.02), (0.754, 0.31, 0.04) and (0.754, 0.31, 0.06) using information of fluid particles within the kernel interaction. Numerical values of  $V_x$ -velocity in the  $x$ -direction are shown in Figure 12-3 for these three points.



**Figure 12-3.** Numerical velocity computed at three different locations.

Pressure values are also computed using **MeasureTool** in the list of points described in the file *PointsPressure.txt*, which corresponds to locations in the front wall of the structure (Figure 12-4) using information of fluid particles within the kernel interaction. Due to the use of the dynamic boundaries, probes used for measuring quantities should not be placed at the exact position of boundary particles. The forces exerted by the boundary particles create a small gap between them and fluid particles (1.5 times  $h$ ); as a result, probes placed inside the gap will have a reduced fluid particle population and will produce either an incorrect or no result. To avoid this issue, it is proposed that probes are placed at a distance 1.5 $h$  from the boundary positions.

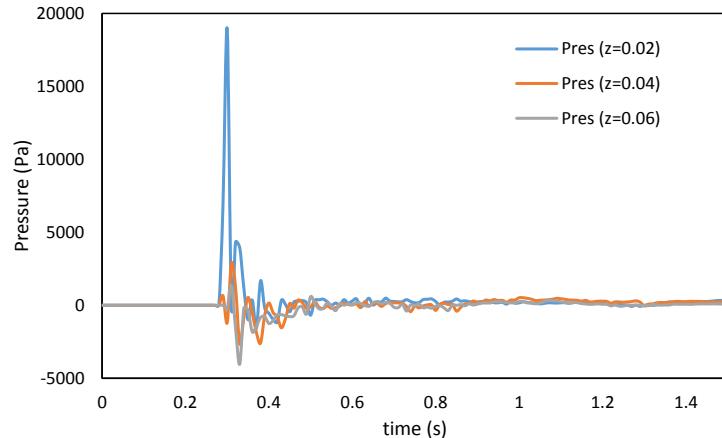
In this example, the *PointsPressure\_Incorrect* probes are placed at the exact position as the boundary particles at the building, while the *PointsPressure\_Correct* probes are placed at a distance of 1.5 $h$  (for the default conditions and parameters of the case) from the boundaries. The first probes produce no pressure results as the number of fluid particles for the interpolation is too small, while the second shows the pressure impact by the water flow.



**Figure 12-4.** Position of incorrect (red) and correct (green) pressure probes in CaseDambreak.

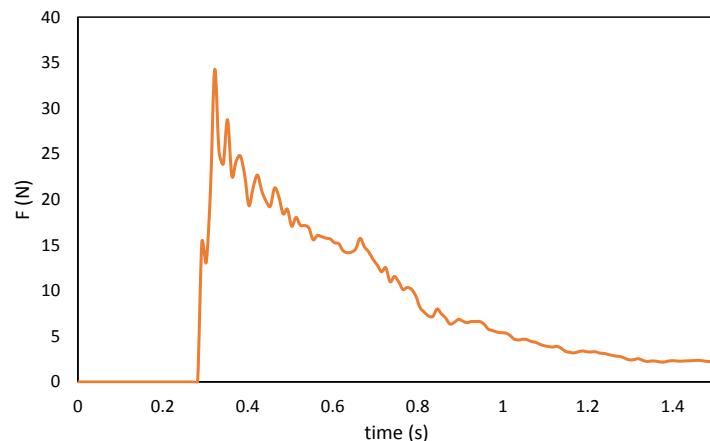
Note that the probes are not automatically associated with the smoothing length  $h$  and their position has to be changed manually if the resolution changes. Also, note that by increasing resolution (i.e. using a smaller “dp”) the gap is reduced.

Figure 12-5 shows the pressure computed at the correct numerical probes.



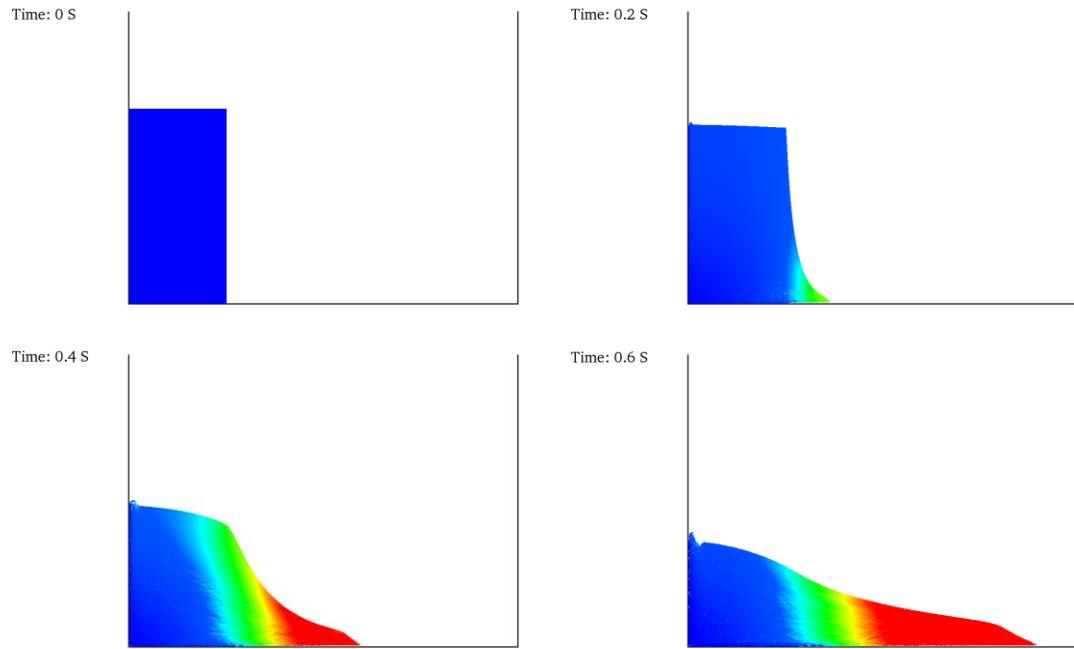
**Figure 12-5.** Pressure values computed at different locations in CaseDambreak.

Finally, this case also shows how to compute forces exerted against a structure. In this way the force is computed using **ComputeForces** as the summation of the acceleration values of the boundary particles that form the front wall of the structure ( $mk=21$ ) multiplied by mass. Figure 12-6 shows the time series of the numerical force.



**Figure 12-6.** Force (X-component) exerted on the front wall of the structure in CaseDambreak.

In addition, a validation in 2D is also included for a new dam break (Figure 12-7) where the experimental data of [Koshizuka and Oka, 1996] can be numerically reproduced. Please note that, in this test case there are no obstacles. The file *EXP\_X-DamTipPosition\_Koshizula&Oka1996.txt* includes the X-position of the tip of the dam for several instants (before the impact with the other wall). This can be used to check the results.

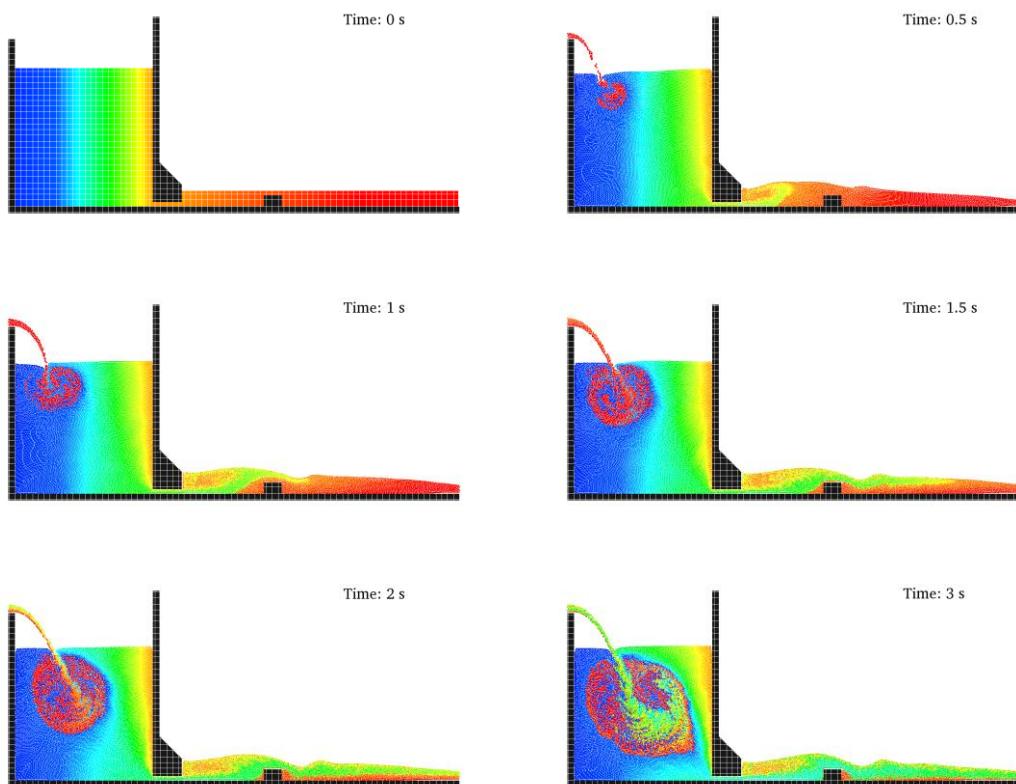


**Figure 12-7.** Instants of the CaseDambreakVal2D simulation.  
Colour represents velocity of the particles.

## 12.2 CASEPERIODICITY

This testcase is an example of periodicity applied to 2-D case where particles that leave the domain through the right wall are introduced through the left wall with the same properties but where the vertical position can change (with an increase of +0.3 in Z-position). *CasePeriodicity.bat* (*CasePeriodicity.sh* in linux) file summarises the tasks to be carried out using the different codes.

Different instants of the CasePeriodicity simulation are shown in Figure 12-8 where the *PartFluid\_xxxx.vtk* of the fluid particles and the *CasePeriodicity\_Bound.vtk* are depicted.

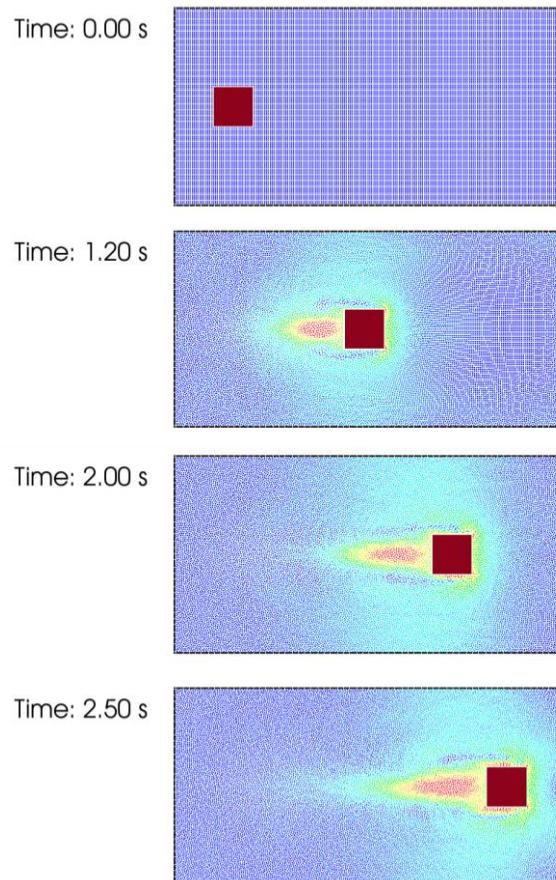


**Figure 12-8.** Different instants of the CasePeriodicity simulation.  
Colour represents Id of the fluid particles.

## 12.3 CASEMOVINGSQUARE

A 2-D case without gravity is simulated. A square box is moving with uniform velocity (`<mvrect>`) inside the numerical tank. Gravity is set to zero, which means that parameter “*b*” cannot be automatically computed following equation of state (Eq. 14) and needs to be specified in `<constantsdef>` (`<b value="1.1200e+05" auto="false" />`). The value of “*b*” is chosen to ensure that the speed of sound,  $c_s$ , is at least 10 times the maximum particle velocity to maintain weak compressibility, that is  $v_{max}, c_s \geq 10 v_{max}$ . Shifting is used for this internal flow to prevent the appearance of voids in the fluid. The case can be also executed without applying shifting to observe the problem of the voids. `CaseMovingSquare.bat` & `CaseMovingSquare_NoShifting.bat` (`CaseMovingSquare.sh` in linux) file summarises the tasks to be carried out using the different codes.

Different instants of the CaseMovingSquare simulation are shown in Figure 12-9 where the *PartFluid\_xxxx.vtk* of the fluid particles and the *Square\_xxxx.vtk* are depicted.



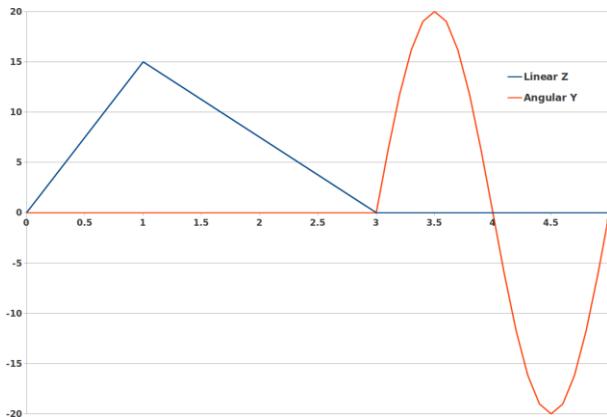
**Figure 12-9.** Different instants of the CaseMovingSquare simulation.  
Colour represents the velocity of the fluid.

## 12.4 CASEFORCES

This is a testcase where external forces are applied to the system. The external forces can be loaded from the files *CaseForcesData\_0.csv* and *CaseForcesData\_1.csv*.

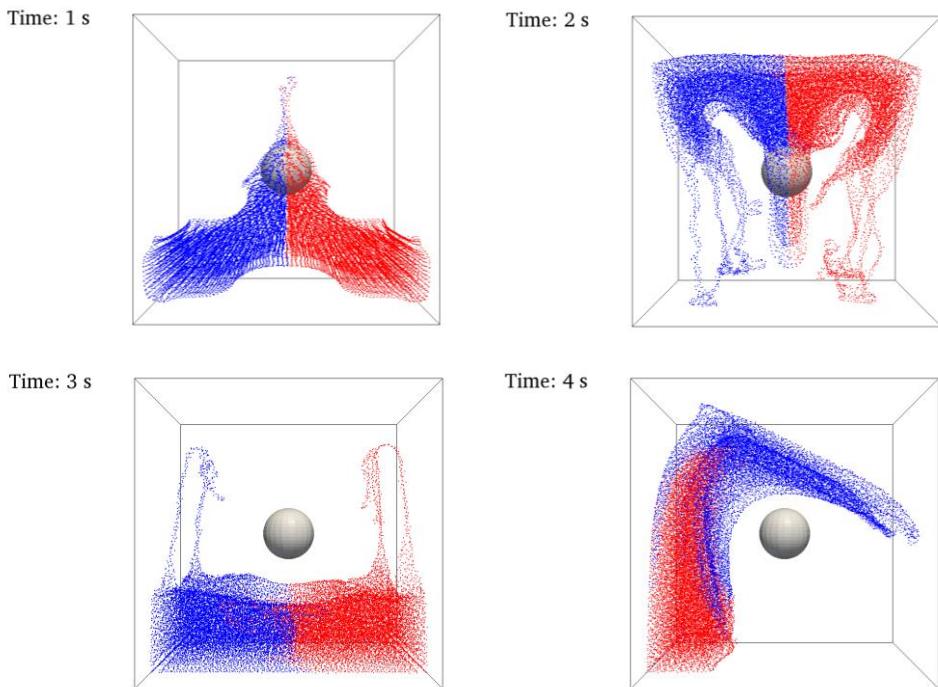
*CaseForces.bat* (*CaseForces.sh* in linux) file summarises the tasks to be carried out using the different codes.

Figure 12-10 shows the external forces that will be applied to the fluid particles inside the numerical tank. This mimics the effect of a box being accelerated in one direction and rotating around the y-axis.



**Figure 12-10.** External forces applied to the fluid particles in CaseForces.

Different instants of the simulation can be observed in Figure 12-11 where the *PartFluid\_xxxx.vtk* of the fluid particles and the file *CaseForces\_Dp.vtk* are depicted. The colours of the fluid particles correspond to the two different MK values.

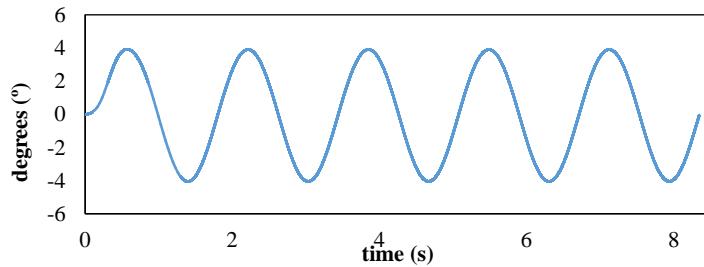


**Figure 12-11.** Different instants of the CaseForces simulation.

## 12.5 CASESLOSHING

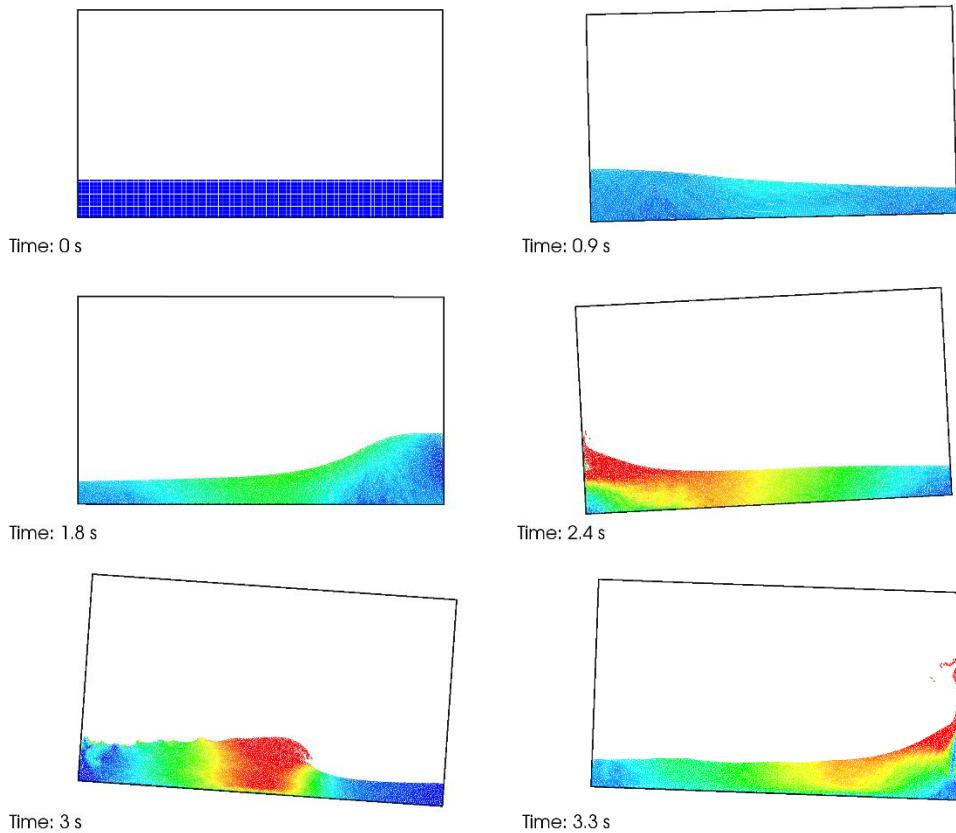
A 2-D sloshing tank is simulated in two different forms:

- a) CaseSloshingAcc: External acceleration is loaded from the CSV file *CaseSloshingAccData.csv* and applied to the fluid particles.
- b) CaseSloshingMotion: Rotational movement is applied to the tank using prescribed rotation defined in the text file *CaseSloshingMotionData.dat* (<mvrotfile>) that contains time and degrees (as shown in Figure 12-12).



**Figure 12-12.** Rotation movement prescribed in file *CaseSloshingMotion.dat*.

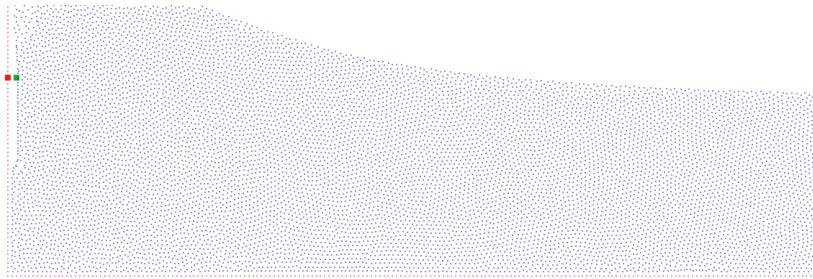
*CaseSloshing.bat* (*CaseSloshing.sh* in linux) file summarises the tasks to be carried out using the different codes for the two cases. Different instants of the simulation can be observed in Figure 12-13 where the files *PartAll\_xxxx.vtk* are depicted.



**Figure 12-13.** Different instants of the CaseSloshingMotion simulation.  
Colour represents the velocity of the fluid.

Validation with [SPHERIC Benchmark #10](#) is also included. Experimental pressure is available in *EXP\_Pressure\_SPHERIC\_Benchmark#10.txt*. Numerical pressures are computed using MeasureTool in both cases. CaseSloshingAcc reads a text file with the fixed position (-0.45, 0, 0.093) while CaseSloshingMotion reads a CSV file with the position of the sensor at each time since the point to measure also moves with the tank.

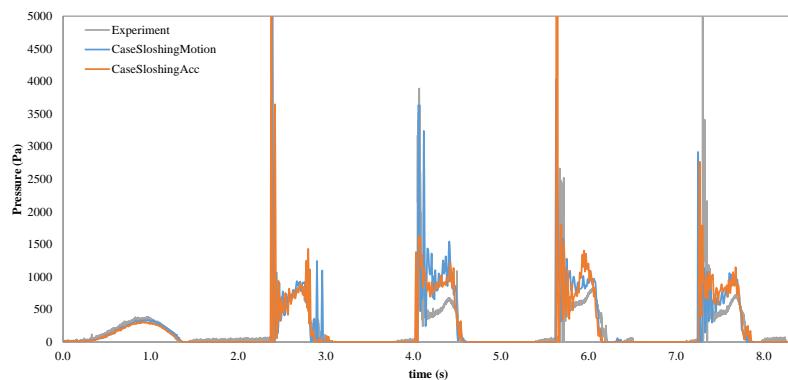
Hence, numerical pressure is computed at the location of the experimental sensor (*PointsPressure\_Incorrect*) and at a distance of  $1.5h$  from the boundaries (*PointsPressure\_Correct*) as shown in Figure 12-14. The first probe produces no pressure results as the number of fluid particles for the interpolation is too small, while the second shows the pressure impact by the water flow.



**Figure 12-14.** Position of incorrect and correct pressure probes in CaseSloshing.

Note that the probes are not automatically associated with the smoothing length  $h$  and their position has to be changed manually if the resolution (or particle size “dp”) changes. Also, note that by increasing resolution the gap is reduced. Using “*\_Sloshing\_Rotation.xlsb*”, the user can modify the value of  $1.5h$  to obtain new positions to be used with MeasureTool.

Figure 12-15 shows the numerical pressure computed using **MeasureTool** in both cases and compared with the experimental signal. Results improve using higher resolution than the one used here.

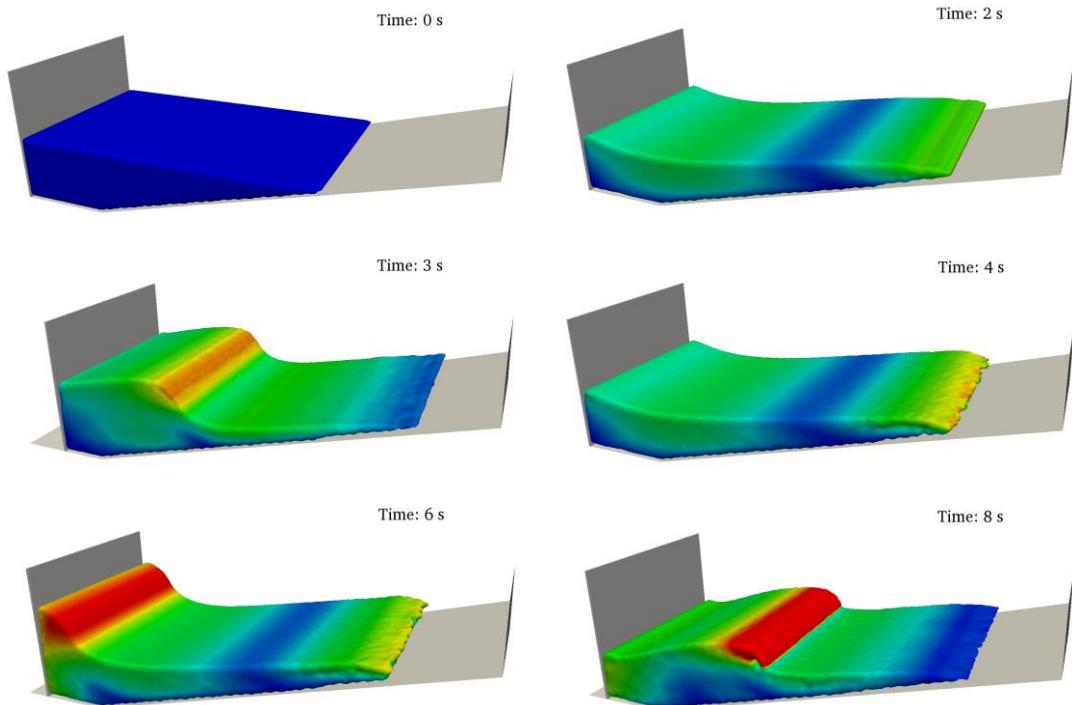


**Figure 12-15.** Experimental and numerical pressure at Sensor1 in CaseSloshing.

## 12.6 CASEWAVEMAKER

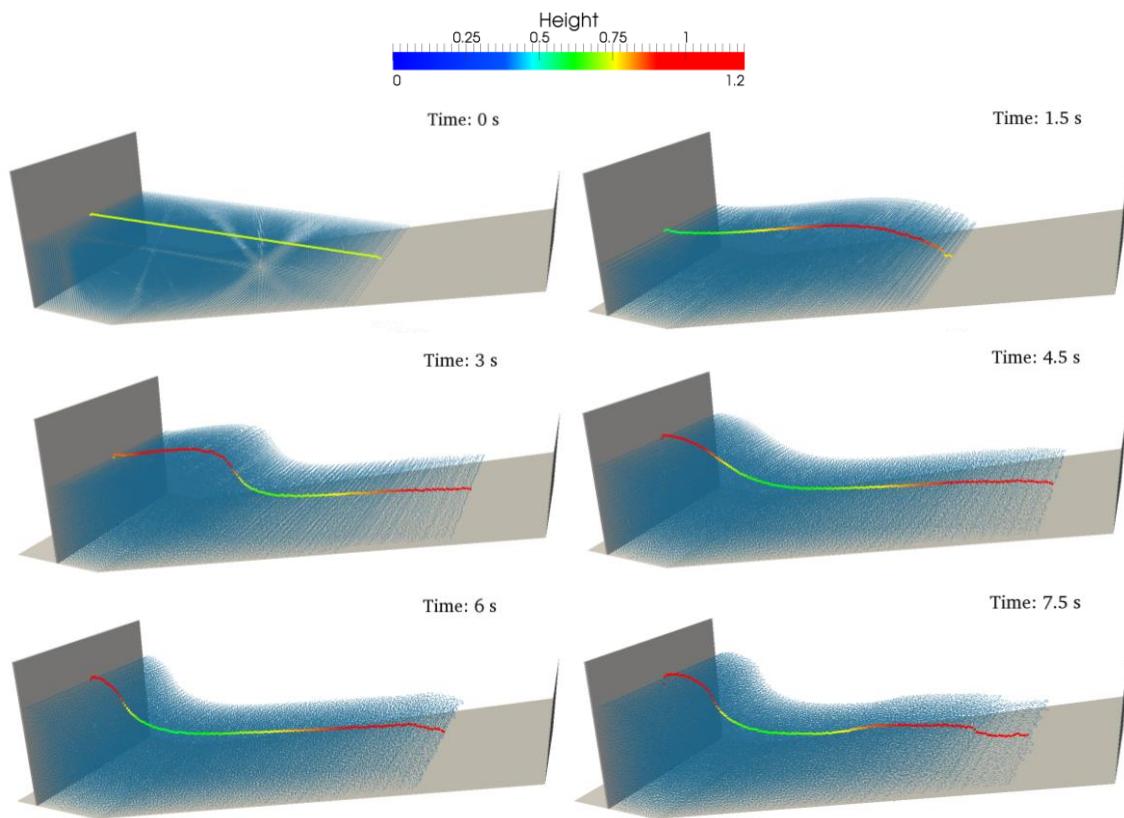
CaseWavemaker simulates several waves breaking on a numerical beach. A wavemaker is performed to generate and propagate the waves. In this testcase, a sinusoidal movement (<mvrectsinu>) is imposed to the boundary particles of the wavemaker and Periodicity in Y direction is also used. *CaseWavemaker.bat* (*CaseWavemaker.sh* in linux) file describes the execution task to be carried out in this directory.

The files *MotionPiston\_xxxx.vtk* containing the position of the wavemaker at different instants and *Surface\_xxxx.vtk* containing the surface representation are used to display the results in Figure 12-16:



**Figure 12-16.** Different instants of the CaseWavemaker simulation.  
Colour represents surface velocity.

Using the **MeasureTool** code, numerical wave elevations have been computed in the list of points described in the file *CaseWavemaker\_PointsHeights.txt*. So that, *PointsHeights\_h\_xxxx.vtk* are now depicted in Figure 12-17.



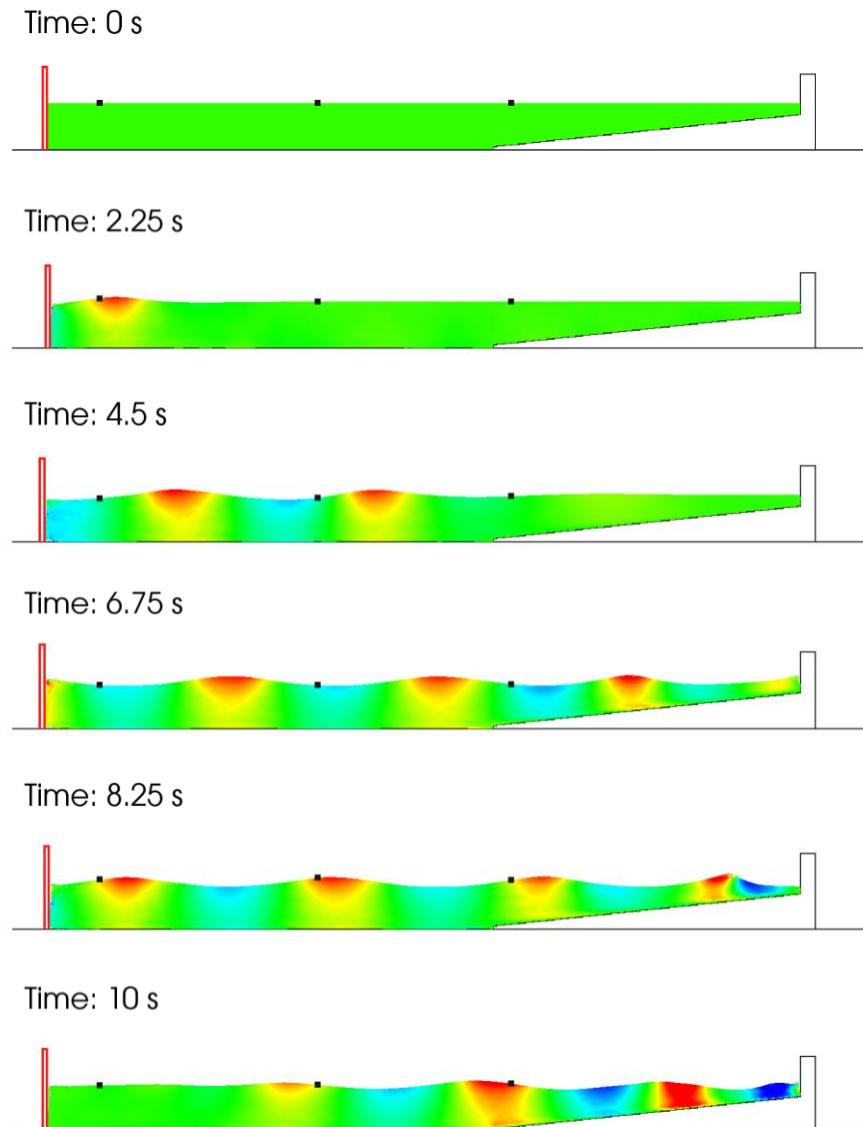
**Figure 12-17.** Different instants of the CaseWavemaker simulation.  
Colour represents wave elevation.

**MeasureTool** is also used to compute the wave elevation at one wave gauge located at  $x=1$ ,  $y=1$  (*CaseWavemaker\_wg0\_3D.txt*) and the output result can be analysed in *WG0\_Height.csv*.

A second 2-D case is also available in this folder where an experiment performed in [CIEMito wave flume](#) at the Universitat Politècnica de Catalunya (UPC) in Barcelona has been numerically reproduced. The movement of the piston is prescribed in the file *CaseWavemaker2D\_Piston\_Movement.dat* (<mvfile>) and an external structure is loaded (*CaseWavemaker2D\_Structure.stl*).

The *CaseWavemaker2D.bat* (*CaseWavemaker2D.sh* in linux) file describes the execution tasks to be carried out in this directory.

Different instants of the CaseWavemaker2D simulation are shown in Figure 12-18 where the *PartFluid\_xxxx.vtk* of the fluid particles and the *PartMoving\_xxxx.vtk* of the piston are depicted.

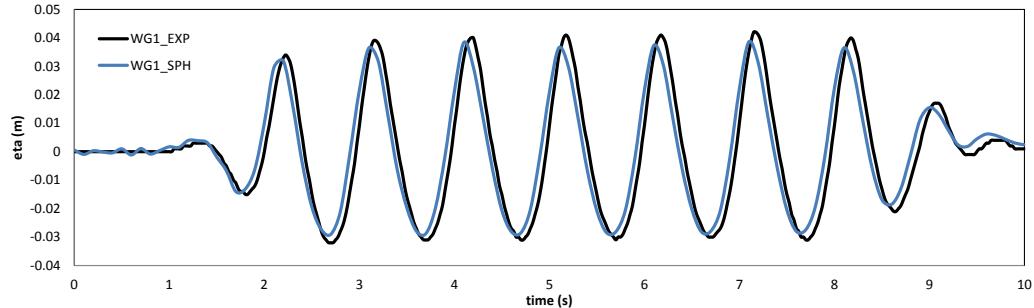


**Figure 12-18.** Different instants of the CaseWavemaker2D simulation.

The three black dots represent the wave gauges  $wg1$ ,  $wg2$ ,  $wg3$  used for validation.

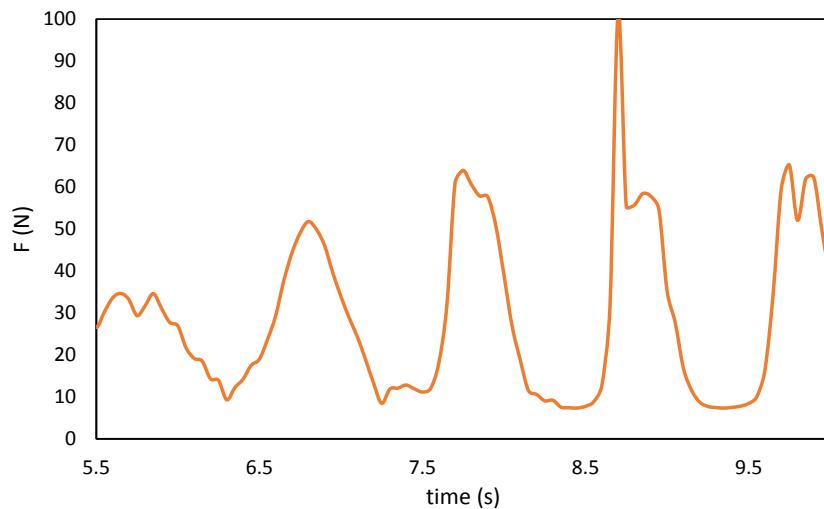
Colour represents the horizontal velocity of the fluid.

This case also includes validation data from the CIEMito experiment: wave surface elevation at different locations (*wg1\_2D.txt*, *wg2\_2D.txt* and *wg3\_2D.txt*) and force exerted onto the right wall. Figure 12-19 shows the experiment and numerical time series of wave surface elevation at wg1.



**Figure 12-19.** Numerical and experimental wave elevation at wg1 in CaseWavemaker2D.

This case also computes forces exerted against the structure in the final of the tank. The force is computed using **ComputeForces** as the summation of the acceleration values of the boundary particles that form the front wall (id:1616-1669) multiplied by mass. Figure 12-20 shows the time series of the numerical force. Note that initial hydrostatic force is 20.72 N (the initial depth in front of the wall is 0.065 m)



**Figure 12-20.** Force (X-component) exerted on the right wall in CaseWavemaker2D.

## 12.7 CASEWAVEGENERATION

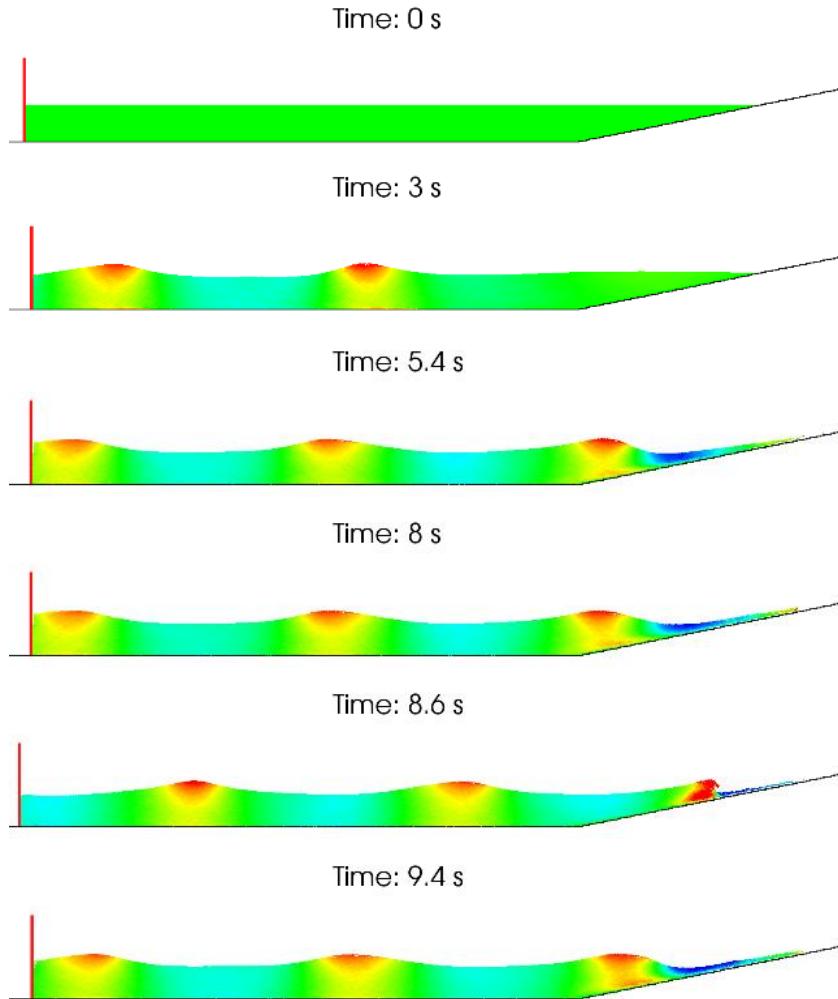
Two examples of automatic generation of waves are shown here: (i) regular waves, (ii) irregular waves.

### *Regular monochromatic waves*

The first one computes the movement of a piston to generate waves of the given wave height ( $H$ ), wave period ( $T$ ) and depth ( $d$ ) following second-order wave theory (<piston>).

*CaseWavesREG.bat* (*CaseWavesREG.sh* in linux) file describes the execution tasks to be carried out in this directory.

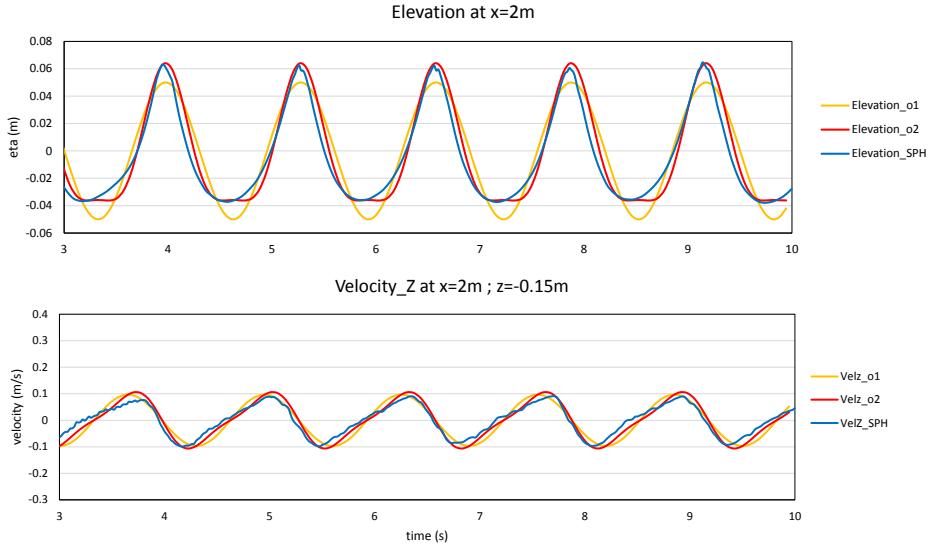
Different instants of the CaseWavesREG simulation are shown in Figure 12-21 where the *PartFluid\_xxxx.vtk* of the fluid particles and the *PartPiston\_xxxx.vtk* of the piston are depicted.



**Figure 12-21.** Different instants of the CaseWavesREG simulation.  
Colour represents the horizontal velocity of the fluid.

Numerical results using **MeasureTool** are compared with theoretical ones (that can also be computed by DualSPHysics using option <savemotion> in the XML configuration). Figure 12-22 shows the comparison of numerical wave surface elevation measured (at a

distance of 2m of the piston) and numerical vertical velocity (at 2m of the piston and 0.15m from still water level) with the theoretical result using first- and second-order solutions. Note that numerical results agree with second-order theory, which validates the correct generation and propagation using DualSPHysics.

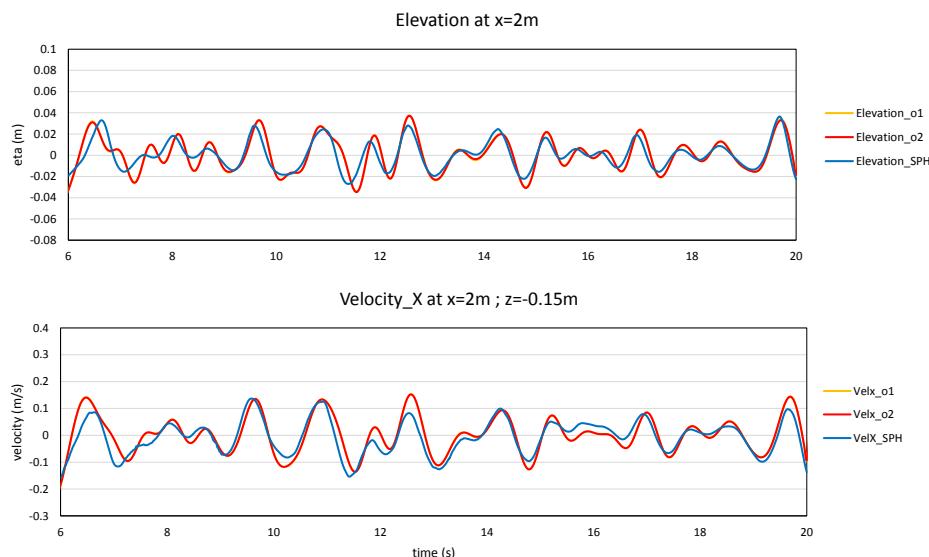


**Figure 12-22.** Comparison between theoretical and numerical water surface elevation and vertical velocity for regular waves in CaseWavesREG.

#### Irregular waves

The second case computes the movement of a piston to generate irregular waves significant wave height ( $H_s$ ), peak wave period ( $T$ ) and depth ( $d$ ) with spectrum type JONSWAP (<piston\_spectrum>). *CaseWavesIRREG.bat* (*CaseWavesIRREG.sh* in linux) file describes the execution tasks to be carried out in this directory.

Figure 12-23 also shows the comparison of numerical wave surface elevation and horizontal velocity with the theoretical solution. Good agreement is also observed for irregular waves.

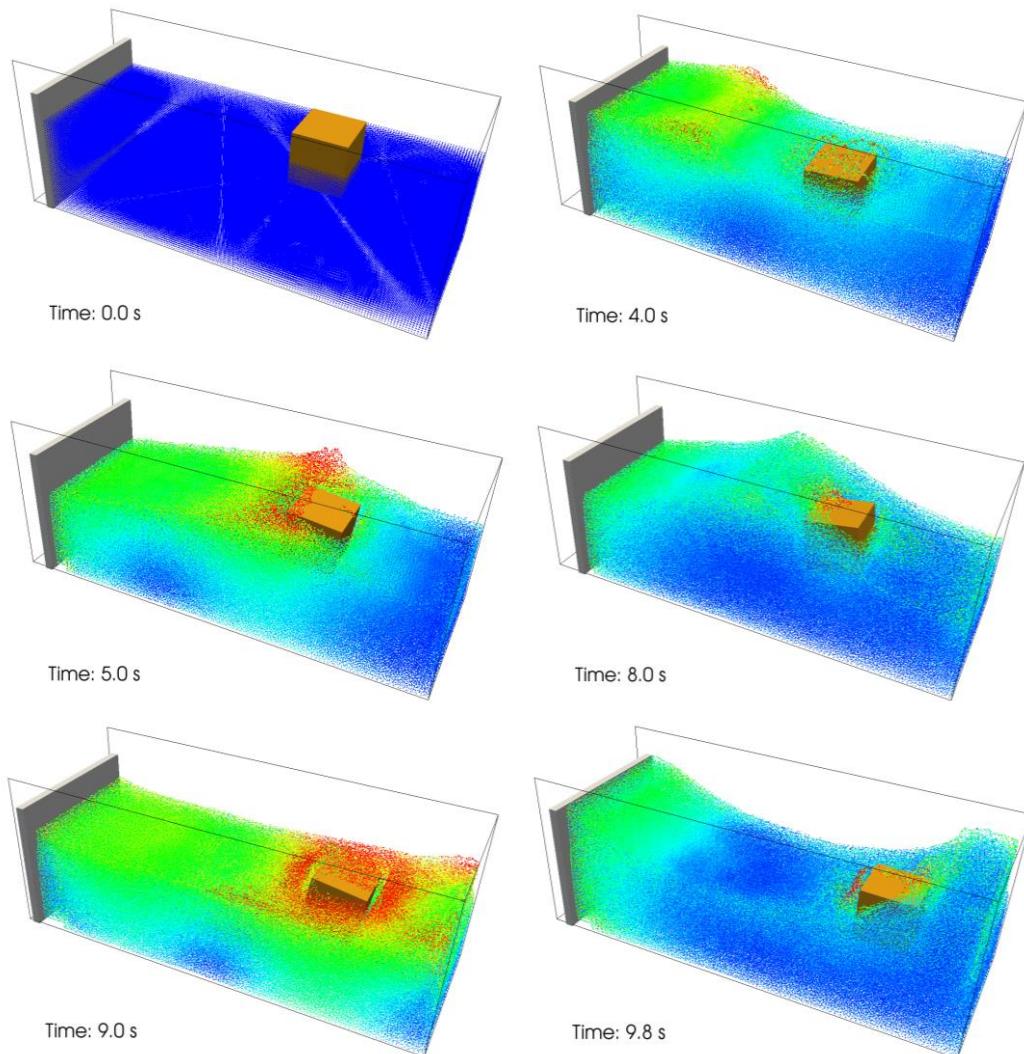


**Figure 12-23.** Comparison between theoretical and numerical water surface elevation and horizontal velocity for irregular waves in CaseWavesIRREG.

## 12.8 CASEFLOATING

In this testcase, a 3-D floating box moves due to waves generated with a piston. A sinusoidal movement (`<mvrectsinu>`) is imposed to the boundary particles of the wavemaker and Periodicty in Y direction is also used. `CaseFloating.bat` (`CaseFloating.sh` in linux) file summarises the tasks to be carried out using the different codes.

Figure 12-24 shows different instants of the simulation of this test case. The floating object is represented using VTK files generated using the BAT file provided, this can help users with their own post-processing:

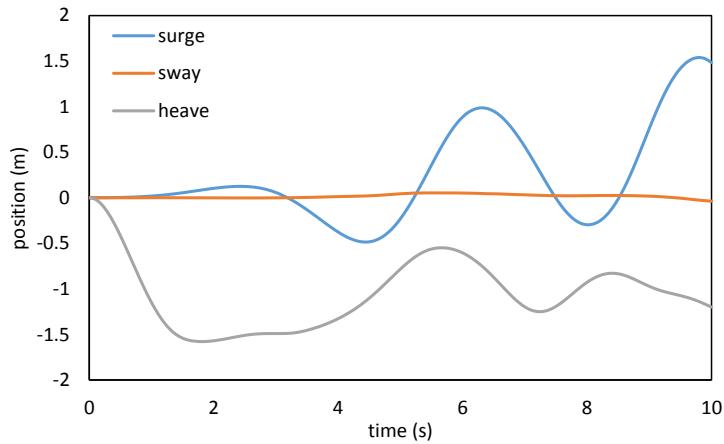


**Figure 12-24.** Different instants of the CaseFloating simulation.

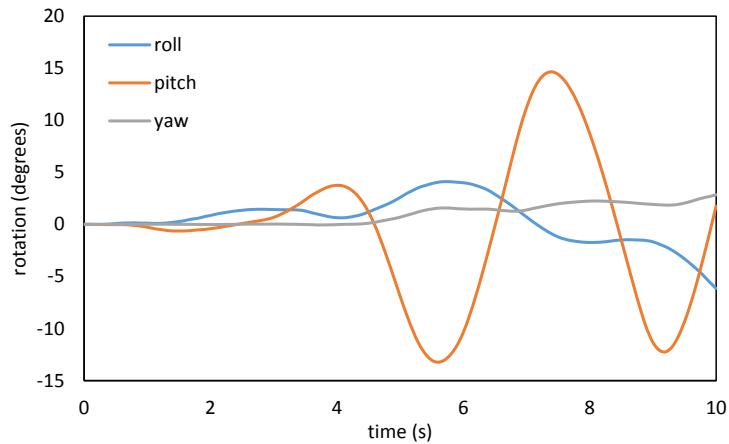
Colour represents velocity of the particles.

The floating box is created using `<setdrawmode mode="full" />` so particles are located in the faces of the box and inside. Only floating objects created with `<setdrawmode mode="full" />` or `<setdrawmode mode="solid" />` (page 20 of the XML\_GUIDE\_v4.pdf) have been validated using DualSPHysics.

The 3-D motions and 3-D rotations of the floating box (mk:61) can be computed using **FloatingInfo** code. Hence, Figure 12-25 shows the time series of the surge, sway and heave motions while Figure 12-26 shows the pitch, roll and yaw angles.



**Figure 12-25.** Surge, sway and heave motions of the floating box.

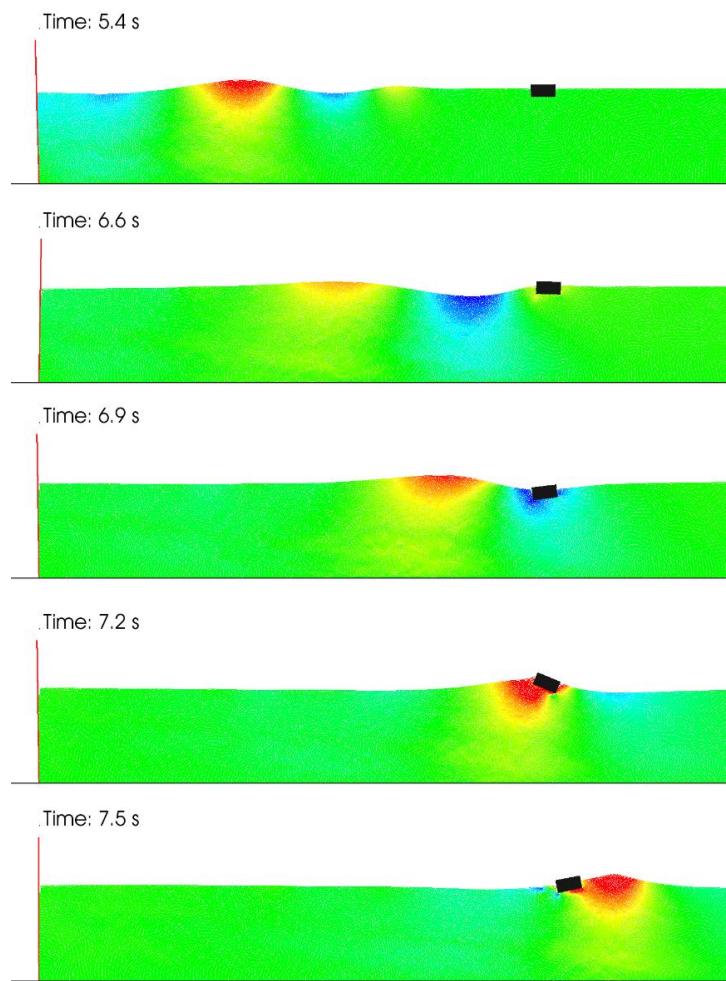


**Figure 12-26.** Pitch, roll and yaw rotation of the floating box.

A 2-D case is also designed where a floating box (density=680 kg/m<sup>3</sup>) moves under the action of non-linear waves in a tank with flap that reads rotational motion from an external file *CaseFloatingWavesVal\_Flap.dat* (time & angle) using <mvrotfile>.

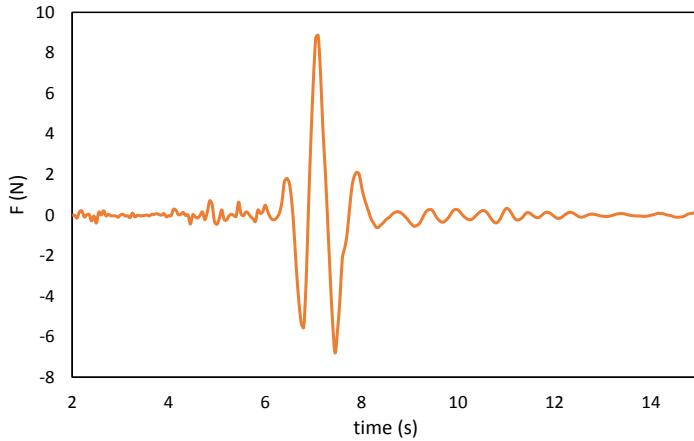
*CaseFloatingWavesVal.bat* (*CaseFloatingWavesVal.sh* in linux) file summarises the tasks to be carried out using the different codes.

Different instants of the *CaseFloatingWavesVal* simulation are shown in Figure 12-27 where the *PartFluid\_xxxx.vtk* of the fluid particles, the *PartMotion\_xxxx.vtk* of the piston and the *PartFloating\_xxxx.vtk* of the floating box are depicted.



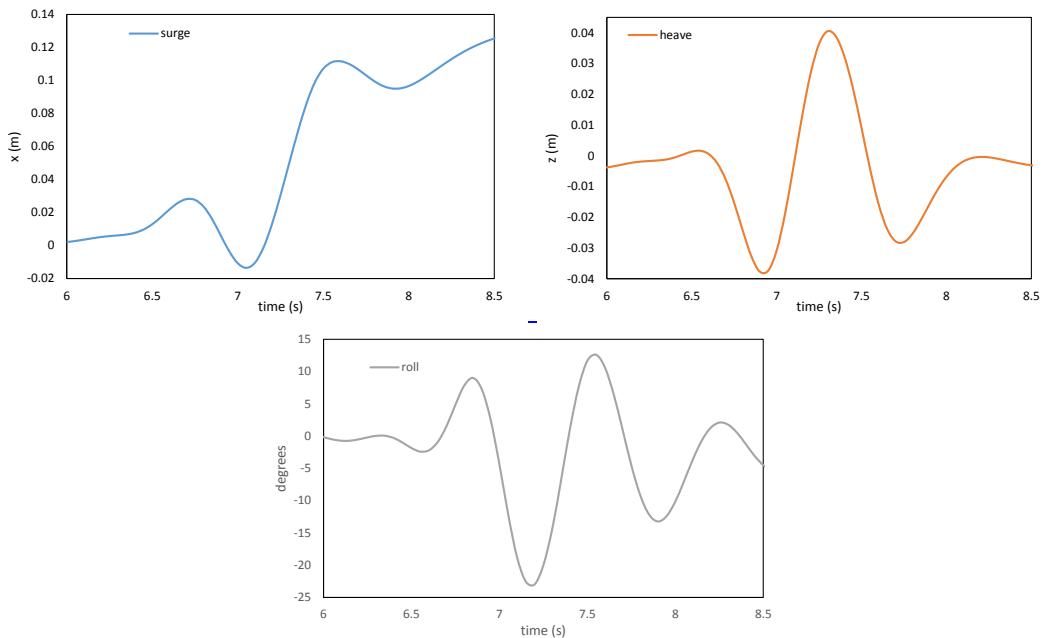
**Figure 12-27.** Different instants of the CaseFloatingWavesVal simulation.  
Colour represents the horizontal velocity of the fluid.

The force exerted on the floating box has been also computed using **ComputeForces** ( $\text{mk}=61$ ). Figure 12-28 shows the time series of the horizontal numerical force.



**Figure 12-28.** Force (X-component) exerted on the floating box in CaseFloatingWavesVal.

The 2-D motion of the floating box (surge, heave) and 2-D rotation (roll) can be computed using **FloatingInfo** code as shown in Figure 12-29.

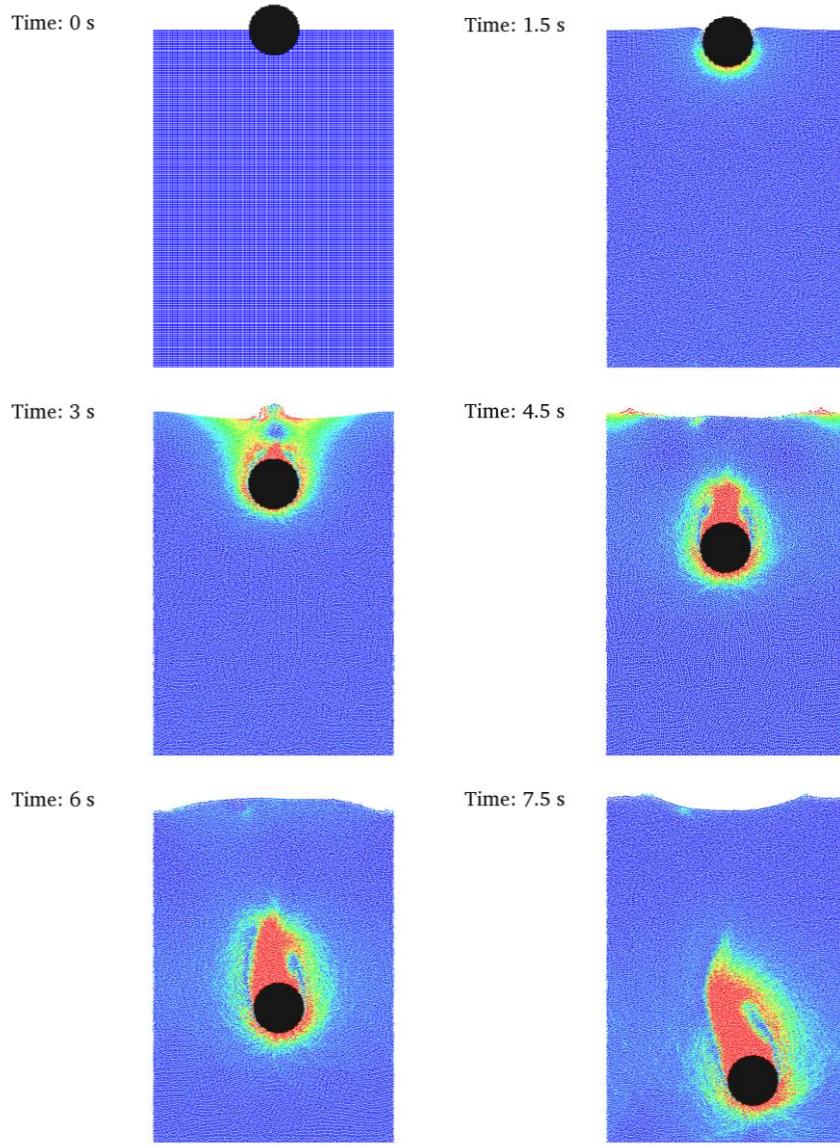


**Figure 12-29.** Surge, heave and roll of the floating box in CaseFloatingWavesVal.

Experimental data from [Hadzic et al., 2005] is also included in the testcase. This validation can be seen at <http://dual.sphysics.org/index.php/validation/wavesfloatings/>.

Finally, a 2-D case of a falling sphere is also provided to simulate the buoyancy-driven motion of an unrestricted rigid body. The sphere has a relative density compared with water of 1.2 (density=1200 kg/m<sup>3</sup>).

Different instants of the CaseFloatingSphereVal2D simulation can be observed in Figure 12-30.



**Figure 12-30.** Different instants of the CaseFloatingSphereVal2D simulation.  
Colour represents velocity of the particles.

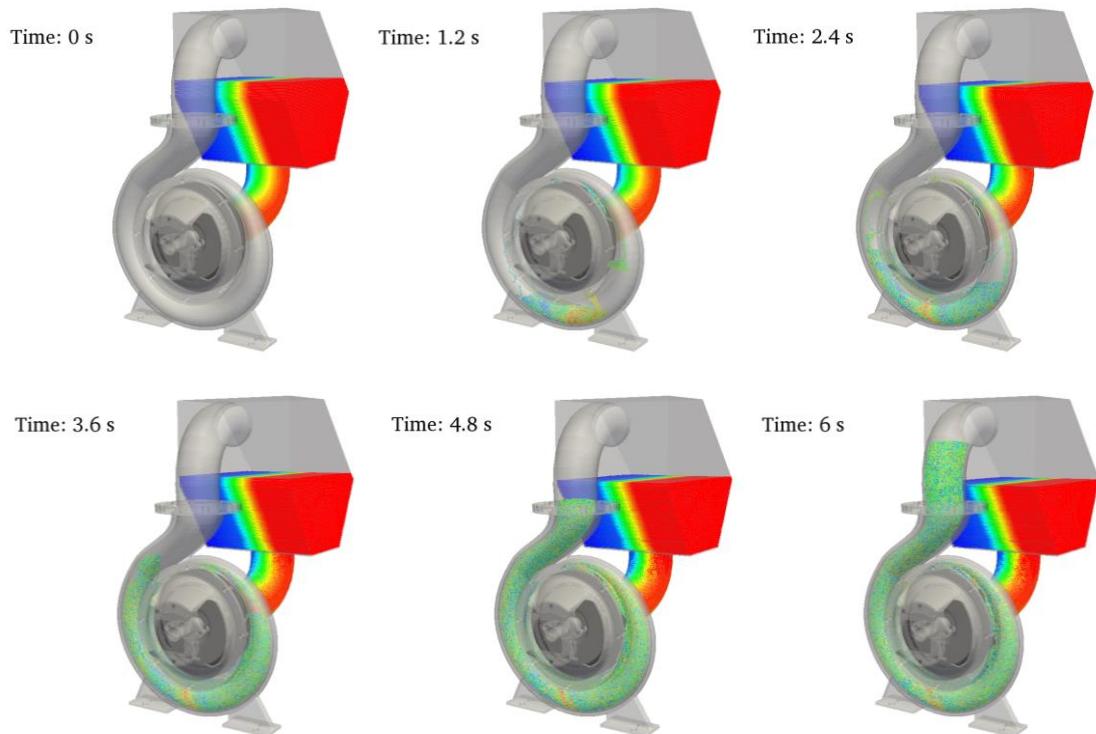
The text files included in the folder contains the experimental displacement and velocity of experiments in [Fekken, 2004] and [Moyo and Greenhow, 2000] that can be numerically computed using the **FloatingInfo** code.

Validation results can be also seen in [Canelas et al., 2015] where spheres are also created using <setdrawmode mode="full" />.

## 12.9 CASEPUMP

This 3-D testcase loads an external geometry of a pump created with CAD (Figure 9-4). Two files are used; a fixed object (*pump\_fixed.vtk*) and a moving part (*pump\_moving.vtk*) that describes a rotational movement (<mvrotace>) and the reservoir is pre-filled with fluid particles (<fillbox>). *CasePump.bat* (*CasePump.sh* in linux) file summarises the tasks to be carried out using the different codes.

In Figure 12-31, the fixed *Pumpfixed.vtk*, the different *MotionPump.vtk* and the *PartFluid\_xxxx.vtk* files of the fluid particles are represented at different instants of the simulation.



**Figure 12-31.** Different instants of the CasePump simulation.

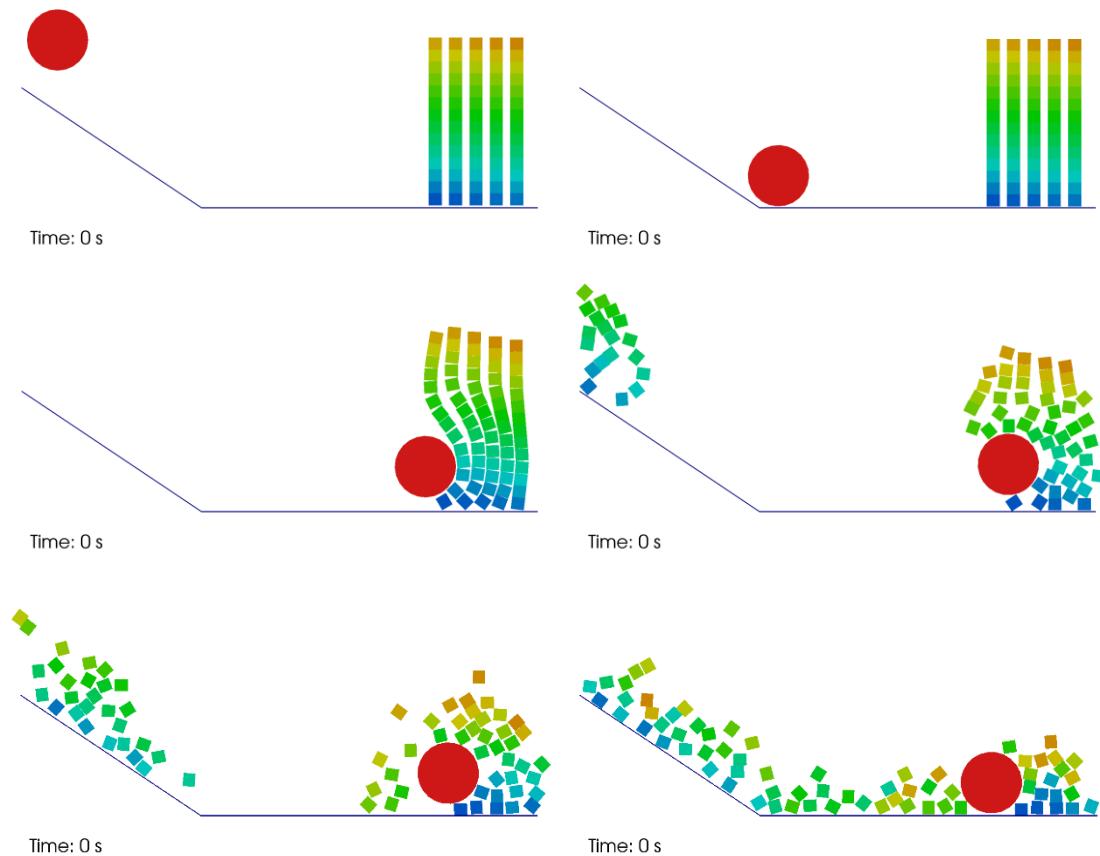
Colour represents Id of the fluid particles.

## 12.10 CASEDEM

This folder includes cases to be executed using DEM for the interaction between solids (RigidAlgorithm=2). The first simulation demonstrates a ball hitting several blocks without any fluid particles being created. The file *Floating\_Materials.xml* contains properties of different materials; the ball and the bottom are defined as “steel” and the blocks are “pvc”.

*CaseBowling.bat* (*CaseBowling.sh* in linux) file summarises the tasks to be carried out using the different codes.

Different instants of the simulation plotting the files *PartSolids\_xxxx.vtk* can be seen in Figure 12-32.

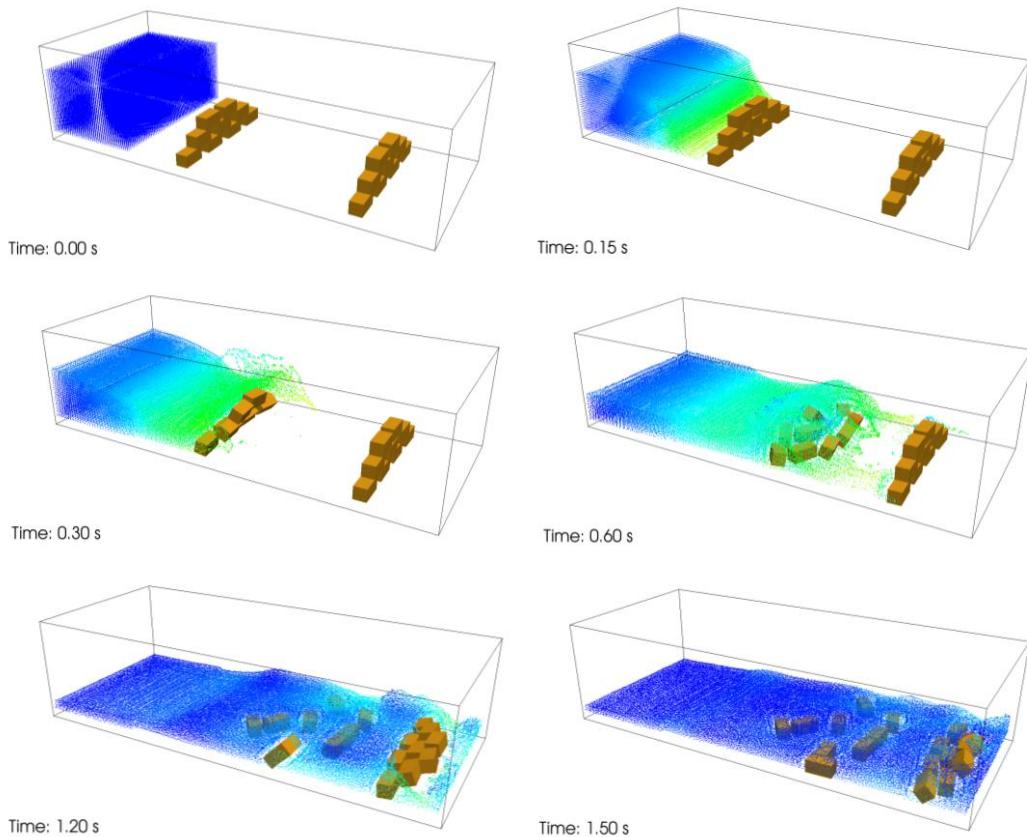


**Figure 12-32.** Different instants of the CaseBowling simulation.

The second DEM simulation consists of a 3-D dam-break flow impacting several blocks. Interaction between fluid and solids are computed using SPH and interaction between blocks and the blocks and the tank walls are computed using DEM. The file *Floating\_Materials.xml* contains properties of different materials; the walls and bottom of the tank are defined as “steel” and the blocks are “pvc”.

*CaseSolids.bat* (*CaseSolids.sh* in linux) file summarises the tasks to be carried out using the different codes.

Figure 12-33 shows different instants of the simulation using the files *MotionBlocks\_xxxx.vtk* and *PartFluid\_xxxx.vtk*.



**Figure 12-33.** Different instants of the CaseSolids simulation.

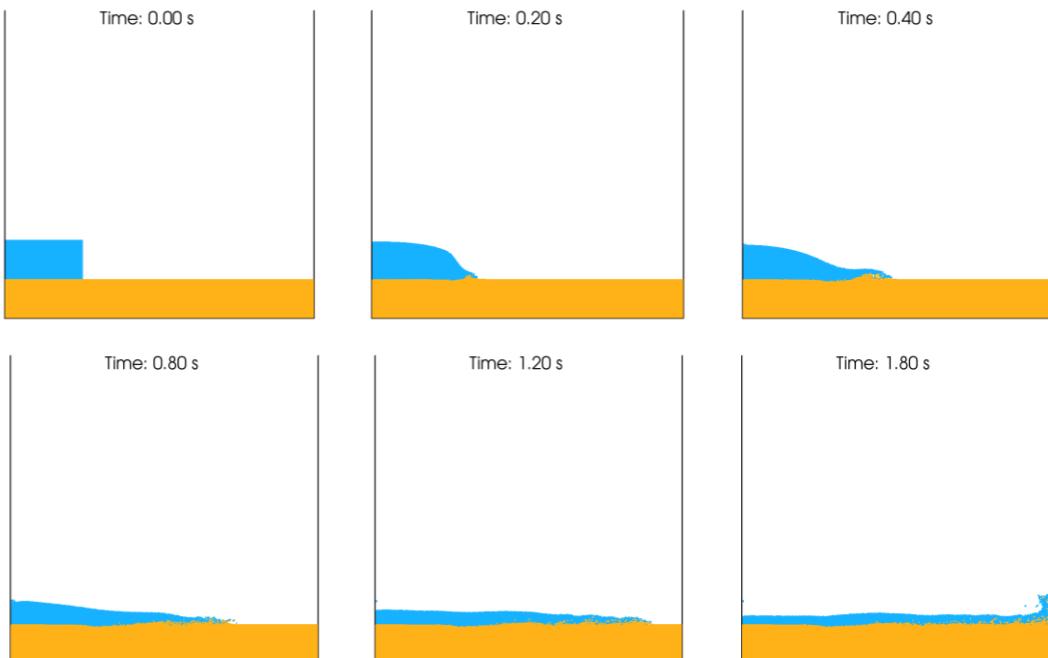
Colour represents the velocity of the fluid.

## 12.11 CASEMULTIPHASE

The last folder includes a case to be executed with **DualSPHysicsMultiphase\_3.4** code. The case consists of two phases; water and sediment. A PDF document is attached with detailed description of the case “Guide\_CaseTwoPhases.pdf”

*CaseTwoPhases.bat* (*CaseTwoPhases.sh* in linux) file summarises the tasks to be carried out using the different codes.

Different instants of the simulation plotting *CaseTwoPhases\_Bound.vtk*, the files *PartFluid\_xxxx.vtk* and *PartSediment\_xxxx.vtk* can be seen in Figure 12-34.



**Figure 12-34.** Different instants of the CaseTwoPhases simulation.

## 13. How to modify DualSPHysics for your application

### Creating new cases

More information is provided to help users to create their own cases of study:

*XML\_GUIDE\_v4.0.pdf*

*ExternalModelsConversion\_GUIDE.pdf*

### Source files of the SPH solver

To add new formulation or changes, the following files are the ones that require your attention:

CPU and GPU executions: main.cpp  
JCfgRun (.h .cpp)  
JSph (.h .cpp)  
JPartsLoad4 (.h .cpp)  
JPartsOut (.h .cpp)  
Types.h

For only CPU executions: JSphCpu (.h .cpp)  
JSphCpu (.h .cpp)  
JSphCpuSingle (.h .cpp)  
JSphTimersCpu.h  
JCellDivCpu (.h .cpp)  
JCellDivCpuSingle (.h .cpp)  
JArraysCpu (.h .cpp)

For only GPU executions: JSphGpu (.h .cpp)  
JSphGpu\_ker (.h .cu)  
JSphGpuSingle (.h .cpp)  
JSphTimersGpu.h  
JCellDivGpu (.h .cpp)  
JCellDivGpu\_ker (.h .cu)  
JCellDivGpuSingle (.h .cpp)  
JCellDivGpuSingle\_ker (.h .cu)  
JArraysGpu (.h .cpp)

Please read Section 6 for a complete description of these files.

### Example code “ToVtk”

The release includes not only the source files of DualSPHysics v4.0 but also the source files of a code named “ToVtk”. This code is provided to show how to load and interpret particle data, how to read .bi4 files and how to create .vtk files.



## 14. FAQ: Frequently asked questions about DualSPHysics

- 1. What do I need to use DualSPHysics? What are the hardware and software requirements?**
- 2. Why DualSPHysics binary is not running?**
- 3. How can I compile the code with different environments/compilers?**
- 4. How many particles can I simulate with the GPU code?**
- 5. How should I start looking at the source code?**
- 6. How can I create my own geometry?**
- 7. How can I contribute to the project?**
- 8. How can I modify the code of the precompiled libraries?**
- 9. How does the code define the limits of the domain?**
- 10. How can I define the movement of boundaries?**
- 11. How can I include a new type of movement?**
- 12. How do I prevent the boundary particles from going outside of the domain limits when applying motion?**
- 13. Why do I observe a gap between boundaries, floating bodies and fluid in the solution?**
- 14. How do I prevent the fluid particles from penetrating inside floating bodies?**
- 15. How do I numerically compute the motion and rotation of floating bodies?**
- 16. When are the fluid particles excluded from the simulation?**
- 17. How do I create a 2-D simulation?**
- 18. How can I solve the error “Constant 'b' cannot be zero”?**
- 19. How can I create a case without gravity?**
- 20. How can I define the speed of sound?**
- 21. What is the recommended alpha value in artificial viscosity?**
- 22. How can I define new properties of the particles?**
- 23. How can I store new properties of the particles (e. g. *Temperature*)?**
- 24. How must I cite the use of the code in my paper?**

### **1. What do I need to use DualSPHysics? What are the hardware and software requirements?**

DualSPHysics can be executed either on CPU or GPU. In order to use DualSPHysics code on a GPU, you need a CUDA-enabled Nvidia GPU card on your machine (<http://developer.nvidia.com/cuda-gpus>). If you want to run GPU simulations (i.e. not develop the source code) the latest version of the driver for your graphics card must be installed. If no source code development is required, there is no need to install any compiler to run the binary of the code, only the driver must be updated. If you also want to compile the code you must install the *nvcc* compiler and a C++ compiler. The *nvcc* compiler is included in the CUDA Toolkit that can be downloaded from the Nvidia website and must be installed on your machine.

## **2. Why DualSPHysics binary is not running?**

If you are trying to run the executable GPU version on a CUDA-enabled Nvidia GPU card, the error message: *Exception (JSphGpuSingle::SelcDevice)*

*Text: Failed getting devices info. (CUDA error: CUDA driver version is insufficient for CUDA runtime version)* can be solved by installing the latest version of the driver for the GPU card.

## **3. How can I compile the code with different environments/compilers?**

The provided source files in this release can be compiled for linux using a ‘makefile’ along with *gcc* and *nvcc* compilers, and for windows using a project for Visual Studio (both VS2010 and VS2013 are provided). In case you use another compiler or other environment, you can adjust the contents of the makefile or you can also use CMAKE.

## **4. How many particles can I simulate with the GPU code?**

The amount of particles that can be simulated depends on (i) the memory space of the GPU card and (ii) the options of the simulation.

## **5. How should I start looking at the source code?**

Section 6 of the guide introduces the source files including some call graphs for a better understanding and it is also highly recommended that you read the documentation generated with Doxygen ([www.doxygen.org](http://www.doxygen.org)).

## **6. How can I create my own geometry?**

Users can follow the provided example cases in the package. Those input XML files can be modified following *XML\_GUIDE\_v4.0.pdf*. Different input formats of real geometries can be converted using *ExternalModelsConversion\_GUIDE.pdf*. This manuscript also describes in detail the input files of the different test cases.

## **7. How can I contribute to the project?**

You can contribute to the DualSPHysics project by reporting bugs, suggesting new improvements, citing DualSPHysics [See the answer to Question 24] in your paper if you use it, submitting your modified codes together with examples.

## **8. How can I modify the code of the precompiled libraries?**

Some code is provided in precompiled libraries to reduce the number of source files and to facilitate the comprehension of only the SPH algorithms by the users. These precompiled code covers secondary aspects during SPH simulations that are only used in specific simulations so they are not used in most of the cases. If the user wants to modify some of the codes included in the precompiled libraries, he can just replace that library by his own implementation.

## **9. How does the code define the limits of the domain?**

In the input XML file, the parameters *pointmin* and *pointmax* only define the domain to create particles beyond these limits fluid or boundary particles will not be created. The limits of the computational domain are computed at the beginning of the DualSPHysics simulation and use the initial minimum and maximum positions of the particles that were already created with GenCase. In order to modify the limits automatically computed by DualSPHysics, different execution parameters can be used:

```
-domain_particles[:xmin,ymin,zmin,xmax,ymax,zmax]  
-domain_particles_prc:xmin,ymin,zmin,xmax,ymax,zmax  
-domain_fixed:xmin,ymin,zmin,xmax,ymax,zmax
```

so that the limits can be specified instead of using initial particle positions.

## **10. How can I define the movement of boundaries?**

Examples of the different type of movements that can be described with DualSPHysics are addressed in directory MOTION. Different kind of movements can be defined such as rectilinear, rotational, sinusoidal or circular motion and they can be uniform or accelerated, with pauses or with hierarchy of movements. And a final option is to load the movement from an external file with a prescribed movement (info of time, X-position, Y-position, Z-position and velocities) or with rotational movement (info of time, angle) that will be interpolated at each time step during the simulation (see CaseSloshingMotion).

## **11. How can I include a new type of movement?**

A new movement can be always defined by using an external file with *mvfile* or *mvrotfile* where the desired movement can be computed in advance and loaded from that file. However if a user wants to create the new type of movement in the source code, this should be implemented in the functions *JSphCpu::RunMotion()* for CPU or *JSphGpu::RunMotion()* for GPU, since the code implemented now is in the library *JSphMotion*.

## **12. How do I prevent the boundary particles from going outside of the domain limits when applying motion?**

As explained in the previous question, the limits of the computational domain are computed starting from the initial minimum and maximum positions of the particles. Since these values use the initial configuration, any movement of boundaries that implies positions beyond these limits will give us the error ‘*boundary particles out the domain*’. The solutions to solve this problem and to avoid creating larger tanks or domains are:

- (i) defining boundary points *<drawpoint>* at the minimum and maximum positions that the particles are expected to reach during the simulation. The option *<drawpoint>* will create a particle at the given location *x,y,z*.
- (ii) using the parameters of DualSPHysics execution mentioned in the answer to Question 9.

### **13. Why do I observe a gap between boundaries, floating bodies and fluid in the solution?**

The gap is a result of pressure overestimation across density discontinuities. It is inherent to the boundary formulation used in DualSPHysics. The forces exerted by the boundary particles create a small gap between them and fluid particles (1.5 times  $h$ ). Note that by increasing the resolution (i.e. using a smaller “ $dp$ ”) the gap is reduced however new boundary conditions are being developed and should be available in future releases [Domínguez et al., 2015].

### **14. How do I prevent the fluid particles from penetrating inside floating bodies?**

Validation of floating using experimental data from [Hadzic et al., 2005], as shown in <http://dual.sphysics.org/index.php/validation/wavesfloating/>, has been performed only for floating objects created with `<setdrawmode mode="full" />` or `<setdrawmode mode="solid" />`, therefore the use of these options is suggested. In this way, floating particles are created in the faces of the object and inside the object so no particle penetration will be observed.

### **15. How do I numerically compute the motion and rotations of floating bodies?**

The new code FloatingInfo allows to obtain different variables of interest of the floating objects during the simulation; positions of the center, linear velocity, angular velocity, motions (surge, sway and heave) and angles of rotation (pitch, roll and yaw). As shown in <http://dual.sphysics.org/index.php/validation/wavesfloating/>, the study of a floating body subjected to a wave packet is validated with data from [Hadzic et al., 2005].

### **16. When are fluid particles excluded from the simulation?**

Fluid particles are excluded during the simulation: (i) if their positions are outside the limits of the domain, (ii) when density values are out of a range (700-1300 by default), (iii) when particles moves beyond 0.9 times the cell size during one time step.

### **17. How do I create a 2-D simulation?**

DualSPHysics can also perform 2-D simulations. To generate a 2-D configuration you only have to change the XML file; imposing the same values in the Y-direction that define the limits of the domain where particles are created (`pointmin.y=1` and `pointmax.y=1`).

### **18. How can I solve the error “Constant ‘b’ cannot be zero”?**

Constant ‘b’ appears in the equation of state (Eq. 14). Constant ‘b’ is zero when fluid height (hswl) is zero (or fluid particles were not created) or if gravity is zero.

First, you should check that fluid particles have been created. Possible errors can appear in 2-D simulations when the seed point of the option `<fillbox>` is not located at the correct y-position. Other solution is to specify the value of ‘hswl’ in `<constantsdef>` `<hswl value="0.8" auto="false" />`. When using gravity zero, the value of ‘b’ needs to be specified in `<constantsdef>` (`<b value="1.1200e+05" auto="false" />`) as occurs in CaseMovingSquare.

## **19. How can I create a case without gravity?**

When using gravity zero, the value of ‘b’ needs to be specified in `<constantsdef>` (`<b value="1.1200e+05" auto="false" />`) as occurs in CaseMovingSquare. Otherwise, ‘b’ is zero and gives the error shown in Question 18.

## **20. How can I define the speed of sound?**

By default, the speed of sound (`speedofsound=coefsound*speedsystem`) is calculated as  $10\sqrt{g*hswl}$ . In order to calculate a more suitable ‘speedofsound’ for a particular case requires the user to set the parameters ‘coefsound’ and ‘speedsystem’.

## **21. What is the recommended alpha value in artificial viscosity?**

The value of  $\alpha=0.01$  has proven to give the best results in the validation of wave flumes to study wave propagation and wave loadings exerted onto coastal structures [[Altomare et al., 2015a; 2015c](#)]. However in the simulation of other cases such as dam-breaks, the interaction between fluid and boundaries during dam propagation becomes more relevant and the value of  $\alpha$  should be changed according to the resolution (“*dp*”) to obtain accurate results.

## **22. How can I define new properties of the particles?**

The file format XML offers several resources to define new general parameters or specific properties for different type of particles. In order to load parameters from the section `<parameters>`, the user can mimic how this is also carried out by DualSPHysics. If different properties will be defined for different fluid volumes, section `<properties>` can be used (that is also explained in the XML guide).

## **23. How can I store new properties of the particles (e. g. *Temperature*)?**

The new file format (.bi4) and the post-processing tools have been designed to include new properties defined by the user for its own implementation. The function `JSph::SavePartData()` already includes an example of how to store new particle data. Then, the post-processing codes will automatically manage all variables included in the .bi4 files.

## **24. How must I cite the use of the code in my paper?**

Please refer to the code if you use it in a paper with reference [[Crespo et al., 2015](#)].



## 15. New in DualSPHysics version 4.0

The new version DualSPHysics v4.0 includes important improvements:

- **Structure of the CPU and GPU code.**

This new implementation helps to follow the common structure of the CPU and GPU code. Now only CUDA kernels are implemented in the CUDA files (*.cu*).

The new CPU structure mimics the GPU threads. In this way, CPU threads also compute all particle interactions for one particle and then computes all interactions of a second one. Note that symmetry of pair-wise interactions is not applied in version 4.0 where the code is optimised for execution in multi-core CPU machines (an efficiency of 86.2% is achieved using 32 cores to simulate 150k particles).

In addition, the new implementation of periodic boundaries is also much easier and can be automatically combined by new implementations performed by the user.

- **Optimisation of the size of blocks for execution of CUDA kernels.**

The version 4.0 includes a new automatic estimation of the optimum block size of CUDA kernels (particle interactions on GPU). This optimum block size depends on: (i) features of the kernel (registers and shared memory), (ii) compilation parameters and CUDA version, (iii) hardware to be used and GPU specifications and (iv) input data to be processed by the kernel (divergence, memory coalescent access). The CUDA Occupancy Calculator is available from CUDA version 6.5.

- **Double precision.**

Problems of precision in DualSPHysics, and more generally in SPH, can appear in simulations involving very large domains requiring a very high resolution. It has been shown that the source of the problem comes from the lack of precision to represent the position of the particles. DualSPHysics v4.0 now includes an implementation with double precision where necessary. For example, arrays of position use now double precision and updating state of particles is also implemented with double precision.

- **Improved formulation for adding external forces.**

Previous versions of the code contained only translational accelerations, but version 4.0 now includes all relative rotational acceleration terms (see [[Longshaw and Rogers 2015](#)]). Example: CaseSloshingAcc.

- **Movement from external file for rotation.**

A prescribed movement can be loaded from an external file with defined rotation time series like already exist for translation. The file contains time and angles (degrees or radians). Examples: Motion09, CaseSloshingMotion and CaseFloatingWavesVal.

- **Coupled SPH & DEM.**

The Discrete Element Method (DEM) allows for the computation of rigid particle dynamics, by considering contact laws to account for interaction forces. The coupled numerical solution, based on SPH and DEM discretisations, resolves solid-solid and solid-fluid interactions in broad range of scales [[Canelas et al., 2016](#)]. The source files of DEM implementation are released in version 4.0. Examples: CaseBowling and CaseSolids.

- **Multi-Phase soil-water.**

The DualSPHysics code has been validated for multi-phase simulations involving water and sediment for fully saturated flows [[Fourtakas and Rogers, 2016](#)]. This has been released in v4.0 as executables with the source code to following v4.2. Example: CaseTwoPhases.

- **Novel Shifting algorithm.**

Anisotropic particle spacing is an important stability issue in SPH as, especially in violent flows, particles cannot maintain a uniform distribution. The result is the introduction of noise in the velocity and pressure field, as well as the creation of voids within the water flow for certain cases. With the shifting algorithm, the particles are moved (“shifted”) towards areas with fewer particles (lower particle concentration) allowing the domain to maintain a uniform particle distribution and eliminating any voids that may occur due to the noise. The implementation follows the work of [[Lind et al., 2012](#)]. Example: CaseMovingSquare.

- **Automatic wave generation.**

Wave generation is included in this version of DualSPHysics, for long-crested waves only. In this way, the numerical model can be used to simulate a physical wave flume. Both regular and random waves can be generated. Examples: CaseWavesREG and CaseWavesIRREG.

- **More powerful GenCase.**

With the new GenCase, larger cases can be rapidly created using OpenMP. In addition, new features facilitate the design of more complex cases.

- **New options in MeasureTool.**

Now the MeasureTool code can compute magnitudes at locations/points that change position with time. These locations can be the positions of moving particles or can be loaded from an external file (-pointspose). Example: CaseSloshingMotion.

- **Force computation.**

A new postprocessing tool named ComputeForces is provided in this release. This code computes the force exerted by the fluid onto a solid object. The value of force is calculated as the summation of the acceleration values (solving the momentum equation during interaction between fluid and boundary) multiplied by the mass of each boundary particle. Examples: CaseDambreak (obstacle), CaseWavemaker2D (structure) and CaseFloatingWavesVal2D (floating).

- **Validation of floating bodies and motion analysis.**

The new code FloatingInfo allows to obtain different variables of interest of the floating objects during the simulation; positions of the center, linear velocity, angular velocity, motions (surge, sway and heave) and angles of rotation (pitch, roll and yaw). Examples: CaseFloating, CaseFloatingSphereVal2D and CaseFloatingWavesVal2D.

- **Store information with variable output time.**

The new version allows user to define intervals of time with different output time to save results. Examples: in OTHERS\UseDtDetailed.

- **New properties for different type of fluids or boundaries.**

The user can define new properties for the particles according to “mk” in the XML file and then he can load these values in DualSPHysics.

- **New variables in post-processing tools defined by the user.**

Post-processing tools are now designed to deal with variables defined by the user. In this way, a user can create a new variable in DualSPHysics source files and one can define its own output data that can be then used by PartVTK, MeasureTool, IsoSurface...



## 16. DualSPHysics future

An update to DualSPHysics will be released as v4.2. The updates planned for this release include:

- [Adami et al., 2012] boundary condition
- Local Uniform Stencil (LUST) boundary conditions [Fourtakas et al., 2014].
- MultiGPU with OpenMP.

The new version **DualSPHysics v5.0** that will be released in the future will include:

- Multi-GPU implementation [Domínguez et al., 2013b].
- Variable particle resolution [Vacondio et al., 2013; 2015].
- Multiphase (air-water) [Mokos et al., 2015].

Other features that are planned to be integrated into the DualSPHysics solver that are now under development:

- Wave absorption system [Altomare et al., 2015a].
- Incompressible SPH (ISPH)
- Inlet/outlet flow conditions.
- Coupling with SWASH Wave Propagation Model [Altomare et al., 2015b].
- Moorings [Barreiro, 2015].
- Coupling with MoorDyn library (<http://www.matt-hall.ca/software/moordyn/>).
- Coupling with Chrono-Engine library (<http://chronoengine.info/chronoengine/>).



## 17. References

1. Adami S, Hu XY, Adams NA. 2012. A generalized wall boundary condition for smoothed particle hydrodynamics. *Journal of Computational Physics*, 231 (21), 7057-7075.
2. Antuono M, Colagrossi A, Marrone S, Molteni D. 2010. Free-surface flows solved by means of sph schemes with numerical diffusive terms. *Computer Physics Communications*, 181 (3), 532-549.
3. Antuono M, Colagrossi A, Marrone S. 2012. Numerical diffusive terms in weakly-compressible SPH schemes. *Computer Physics Communications*, 183 (12), 2570-2580.
4. Altomare C, Crespo AJC, Rogers BD, Domínguez JM, Gironella X, Gómez-Gesteira M. 2014. Numerical modelling of armour block sea breakwater with Smoothed Particle Hydrodynamics. *Computers and Structures*, 130, 34-45. doi:10.1016/j.compstruc.2013.10.011.
5. Altomare C, Suzuki T, Domínguez JM, Barreiro A, Crespo AJC, Gómez-Gesteira M. 2015a. Numerical wave dynamics using Lagrangian approach: wave generation and passive & active wave absorption. In: Proceedings of the 10th SPHERIC International Workshop, Parma, Italy.
6. Altomare C, Domínguez JM, Crespo AJC, Suzuki T, Caceres I, Gómez-Gesteira M. 2015. Hybridisation of the wave propagation model SWASH and the meshfree particle method SPH for real coastal applications. *Coastal Engineering Journal*, 57(4): 1550024. doi:10.1142/S0578563415500242.
7. Altomare C, Crespo AJC, Domínguez JM, Gómez-Gesteira M, Suzuki T, Verwaest T. 2015. Applicability of Smoothed Particle Hydrodynamics for estimation of sea wave impact on coastal structures. *Coastal Engineering* 96, 1-12.
8. Barreiro A. 2015. Smoothed Particle Hydrodynamics model for civil and coastal engineering applications, PhD thesis, Universidade de Vigo, Spain.
9. Batchelor G K. 1974. *Introduction to fluid dynamics*. Cambridge University Press.
10. Biesel F, Suquet F. 1951. Etude theorique d'un type d'appareil a la houle. *La Houille Blanche*, 2, 157-160.
11. Bouscasse B, Colagrossi A, Marrone S, Antuono M. 2013. Nonlinear water wave interaction with floating bodies in SPH. *Journal of Fluids and Structures*, 42, 112-129.
12. Canelas RB. 2015. Numerical modeling of fully coupled solid-fluid flows, PhD thesis, Universidade de Lisboa, Instituto Superior Técnico, Portugal.

13. Canelas RB, Domínguez JM, Crespo AJC, Gómez-Gesteira M, Ferreira RML. 2015. A Smooth Particle Hydrodynamics discretization for the modelling of free surface flows and rigid body dynamics. *International Journal for Numerical Methods in Fluids*, 78, 581-593.
14. Canelas RB, Crespo AJC, Domínguez JM, Ferreira RML, Gómez-Gesteira M. 2016. SPH-DEM model for arbitrary geometries in free surface solid-fluid flows. *Computer Physics Communications*.
15. Crespo AJC, Gómez-Gesteira M and Dalrymple RA. 2007. Boundary conditions generated by dynamic particles in SPH methods. *Computers, Materials & Continua*, 5, 173-184.
16. Crespo AJC, Gómez-Gesteira M and Dalrymple RA. 2008. Modeling Dam Break Behavior over a Wet Bed by a SPH Technique. *Journal of Waterway, Port, Coastal, and Ocean Engineering*, 134(6), 313-320.
17. Crespo AJC, Domínguez JM, Barreiro A, Gómez-Gesteira M and Rogers BD. 2011. GPUs, a new tool of acceleration in CFD: Efficiency and reliability on Smoothed Particle Hydrodynamics methods. *PLoS ONE* 6 (6), e20685, doi:10.1371/journal.pone.0020685.
18. Crespo AJC, Domínguez JM, Rogers BD, Gómez-Gesteira M, Longshaw S, Canelas RB, Vacondio R, Barreiro A, García-Feal O. 2015. DualSPHysics: Open-source parallel CFD solver based on Smoothed Particle Hydrodynamics (SPH). *Computer Physics Communications*, 187, 204-216.
19. Dalrymple RA and Rogers BD. 2006. Numerical modeling of water waves with the SPH method. *Coastal Engineering*, 53, 141–147.
20. Domínguez JM, Crespo AJC, Gómez-Gesteira M, Marongiu JC. 2011. Neighbour lists in Smoothed Particle Hydrodynamics. *International Journal for Numerical Methods in Fluids*, 67, 2026-2042, doi: 10.1002/fld.2481
21. Domínguez JM, Crespo AJC, Gómez-Gesteira M. 2013a. Optimization strategies for CPU and GPU implementations of a smoothed particle hydrodynamics method. *Computer Physics Communications*, 184(3), 617-627. doi:10.1016/j.cpc.2012.10.015
22. Domínguez JM, Crespo AJC, Valdez-Balderas D, Rogers BD, Gómez-Gesteira M. 2013b. New multi-GPU implementation for Smoothed Particle Hydrodynamics on heterogeneous clusters. *Computer Physics Communications*, 184, 1848-1860. doi: 10.1016/j.cpc.2013.03.008
23. Domínguez JM, Crespo AJC, Gómez-Gesteira M. 2013c. Simulating more than 1 billion SPH particles using GPU hardware acceleration. In: *Proceedings of the 8th SPHERIC International Workshop*, Trondheim, Norway.

24. Domínguez JM, Crespo AJC, Cercós-Pita JL, Fourtakas G, Rogers BD, Vacondio R. 2015. Evaluation of reliability and efficiency of different boundary conditions in an SPH code. In: Proceedings of the 10th SPHERIC International Workshop, Parma, Italy.
25. Fekken G. 2004. Numerical simulation of free surface flow with moving rigid bodies. Ph. D. thesis, University of Groningen.
26. Fourtakas G. 2014. Modelling multi-phase flows in Nuclear Decommissioning using SPH, PhD thesis, University of Manchester, United Kingdom.
27. Fourtakas G, Rogers BD. 2016. Modelling multi-phase liquid-sediment scour and resuspension induced by rapid flows using Smoothed Particle Hydrodynamics (SPH) accelerated with a graphics processing unit (GPU). *Advances in Water Resources*. doi:10.1016/j.advwatres.2016.04.009.
28. Fourtakas G, Domínguez JM, Vacondio R, Nasar A, Rogers BD. 2014. Local Uniform STencil (LUST) Boundary Conditions for 3-D Irregular Boundaries in DualSPHysics. In: Proceedings of the 9th SPHERIC International Workshop, Paris, France.
29. Frigaard P, Hgedal M, Christensen M. 1993, Wave generation theory. Hydraulics and Coastal Engineering Laboratory, Dept. of Civil Engineering, Aalborg University, Aalborg (Denmark).
30. Gómez-Gesteira M and Dalrymple R. 2004. Using a 3D SPH method for wave impact on a tall structure. *Journal of Waterway, Port, Coastal, and Ocean Engineering*, 130(2), 63-69.
31. Gómez-Gesteira M, Rogers BD, Dalrymple RA, Crespo AJC. 2010. State-of-the-art of classical SPH for free-surface flows. *Journal of Hydraulic Research* 48 Extra Issue, 6–27, doi:10.3826/jhr.2010.0012.
32. Gómez-Gesteira M, Rogers BD, Crespo AJC, Dalrymple RA, Narayanaswamy M, Domínguez JM. 2012a. SPHysics - development of a free-surface fluid solver- Part 1: Theory and Formulations. *Computers & Geosciences*, 48, 289-299. doi:10.1016/j.cageo.2012.02.029.
33. Gómez-Gesteira M, Crespo AJC, Rogers BD, Dalrymple RA, Domínguez JM, Barreiro A. 2012b. SPHysics - development of a free-surface fluid solver- Part 2: Efficiency and test cases. *Computers & Geosciences*, 48, 300-307. doi:10.1016/j.cageo.2012.02.028.
34. Gotoh H, Shao S, Memita T. 2001. SPH-LES model for numerical investigation of wave interaction with partially immersed breakwater. *Coastal Engineering Journal*, 46(1), 39–63.
35. Hădžić I, Hennig J, Peric M, Xing-Kaeding Y. 2005. Computation of flow-induced motion of floating bodies. *Applied Mathematical Modelling*, 29, 1196–1210.

36. Koshizuka S and Oka Y. 1996. Moving particle semi-implicit method for fragmentation of compressible fluid. *Nuclear Science Engineering*, 123, 421-434.
37. Lee E-S, Moulinec C, Xu R, Violeau D, Laurence D, Stansby P. 2008. Comparisons of weakly compressible and truly incompressible algorithms for the SPH mesh free particle method. *Journal of Computational Physics*, 227(18), 8417-8436.
38. Leimkuhler BJ, Reich S, Skeel RD. 1996. Integration Methods for Molecular dynamic IMA Volume in Mathematics and its application. Springer.
39. Lind SJ, Xu R, Stansby PK, Rogers BD. 2012. Incompressible smoothed particle hydrodynamics for free-surface flows: A generalised diffusion-based algorithm for stability and validations for impulsive flows and propagating waves. *Journal of Computational Physics*, 231(4), 1499-1523.
40. Liu GR. 2003. Mesh Free methods: Moving beyond the finite element method, CRC Press.
41. Liu Z, Frigaard P. 2001. Generation and Analysis of Random Waves. Laboratoriet for Hydraulik og Havnebygning, Instituttet for Vand, Jord og Miljøteknik, Aalborg Universitet.
42. Lo EYM and Shao S. 2002. Simulation of near-shore solitary wave mechanics by an incompressible SPH method. *Applied Ocean Research*, 24, 275-286.
43. Longshaw, SM, Rogers, BD. 2015. Automotive fuel cell sloshing under temporally and spatially varying high acceleration using GPU-based Smoothed Particle Hydrodynamics (SPH). *Advances in Engineering Software*, 83, 31-44.
44. Lorensen WE, Cline HE. 1987. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. IN: Computer Graphics SIGGRAPH 87 Proceedings, 21(4), 163-170.
45. Madsen OS. 1971. On the generation of long waves. *Journal of Geophysical Research*, 76(36), 8672–8683,
46. Mokos A. 2013. Multi-phase Modelling of Violent Hydrodynamics Using Smoothed Particle Hydrodynamics (SPH) on Graphics Processing Units (GPUs, PhD thesis, University of Manchester, United Kingdom.
47. Mokos A, Rogers BD, Stansby PK, Domínguez JM. 2015. Multi-phase SPH modelling of violent hydrodynamics on GPUs. *Computer Physics Communications*, 196, 304-316.
48. Molteni, D and Colagrossi A. 2009. A simple procedure to improve the pressure evaluation in hydrodynamic context using the SPH, *Computer Physics Communications*, 180 (6), 861–872.

49. Monaghan JJ. 1992. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30, 543- 574.
50. Monaghan JJ. 1994. Simulating free surface flows with SPH. *Journal of Computational Physics*, 110, 399–406.
51. Monaghan JJ and Lattanzio JC. 1985. A refined method for astrophysical problems. *Astron. Astrophys*, 149, 135–143.
52. Monaghan JJ and Kos A. 1999. Solitary waves on a Cretan beach. *Journal of Waterway, Port, Coastal and Ocean Engineering*, 125, 145-154.
53. Monaghan JJ, Cas, RAF, Kos AM, Hallworth M, 1999. Gravity currents descending a ramp in a stratified tank. *Journal of Fluid Mechanics*, 379, 39–70.
54. Monaghan, J.J., 2000. SPH without tensile instability. *Journal of Computational Physics*, 159, 290-311.
55. Monaghan JJ, Kos A, Issa N. 2003. Fluid motion generated by impact. *Journal of Waterway, Port, Coastal and Ocean Engineering* 129, 250-259.
56. Monaghan JJ. 2005. Smoothed Particle Hydrodynamics. *Rep. Prog. Phys*, 68, 1703-1759
57. Moyo S and Greenhow M. 2000. Free motion of a cylinder moving below and through a free surface. *Applied Ocean Research*, 22, 31–44.
58. Papanastasiou TC. 1987. Flows of Materials with Yield. *Journal of Rheology (1978-present)*, 31(5), 385-404.
59. Rogers BD, Dalrymple RA, Stansby PK. 2010. Simulation of caisson breakwater movement using SPH. *Journal of Hydraulic Research*, 48, 135-141, doi:10.3826/jhr.2010.0013.
60. Skillen A, Lind S, Stansby PK, Rogers BD. 2013. Incompressible Smoothed Particle Hydrodynamics (SPH) with reduced temporal noise and generalised Fickian smoothing applied to body-water slam and efficient wave-body interaction. *Computer Methods in Applied Mechanics and Engineering*, 256, 163-173.
61. Vacondio R, Rogers BD, Stansby PK, Mignosa P, Feldman J. 2013. Variable resolution for SPH: a dynamic particle coalescing and splitting scheme. *Computer Methods in Applied Mechanics and Engineering*, 256, 132-148.
62. Vacondio R, Crespo AJC, Domínguez JM, Rogers BD, Gómez-Gesteira M. 2015. DualSPHysics with adaptivity: towards the simulation of real engineering problems

with variable resolution. In: Proceedings of the 10th SPHERIC International Workshop, Parma, Italy.

63. Valdez-Balderas D, Domínguez JM, Rogers BD, Crespo AJC. 2013. Towards accelerating smoothed particle hydrodynamics simulations for free-surface flows on multi-GPU clusters. *Journal of Parallel and Distributed Computing*, 73(11), 1483-1493. doi:10.1016/j.jpdc.2012.07.010.
64. Vand V. 1948. Viscosity of Solutions and Suspensions. I. Theory. *The Journal of Physical and Colloid Chemistry*, 52(2), 277-299.
65. Verlet L. 1967. Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. *Physical Review*, 159, 98-103.
66. Violeau D. 2012. Fluid Mechanics and the SPH Method: Theory and Applications. Oxford University Press.
67. Wendland H. 1995. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in Computational Mathematics* 4, 389-396.
68. Xu R, Stansby P, Laurence D. 2009. Accuracy and stability in incompressible SPH (ISPH) based on the projection method and a new approach. *Journal of Computational Physics*, 228(18), 6703-6725.

## 18. Licenses

### GPL License

The source code for DualSPHysics is freely redistributable under the terms of the GNU General Public License (GPL) as published by the Free Software Foundation.

Simply put, the GPL says that anyone who redistributes the software, with or without changes, must pass along the freedom to further copy and change it. By distributing the complete source code for GNU DualSPHysics under the terms of the GPL, we guarantee that you and all other users will have the freedom to redistribute and change DualSPHysics.

Releasing the source code for DualSPHysics has another benefit as well. By having access to all of the source code for a mathematical system like SPHysics, you have the ability to see exactly how each and every computation is performed.

Although enhancements to DualSPHysics are not required to be redistributed under the terms of the GPL, we encourage you to release your enhancements to SPHysics under the same terms for the benefit of all users. We also encourage developers to submit your changes for inclusion in future versions of DualSPHysics.

Copyright (C) 2016 by

Dr Jose M. Dominguez, Dr Alejandro Crespo,

Prof. Moncho Gomez Gesteira, Dr Anxo Barreiro,

Dr Benedict Rogers, Dr Georgios Fourtakas, Dr Athanasios Mokos,

Dr Renato Vacondio, Dr Ricardo Canelas,

Dr Stephen Longshaw, Dr Corrado Altomare.

EPHYSLAB Environmental Physics Laboratory, Universidade de Vigo

School of Mechanical, Aerospace and Civil Engineering, University of Manchester

DualSPHysics is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

DualSPHysics is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License, along with DualSPHysics. If not, see <http://www.gnu.org/licenses/>.

## **BSD License**

Copyright (C) 2016 by Dr Jose M. Dominguez.  
All rights reserved.

DualSPHysics is an international collaboration between University of Vigo (Spain) and University of Manchester (UK).

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of the DualSPHysics nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## **TinyXml License**

[www.sourceforge.net/projects/tinyxml](http://www.sourceforge.net/projects/tinyxml)

Original code (2.0 and earlier) copyright (c) 2000-2006 Lee Thomason  
([www.grinninglizard.com](http://www.grinninglizard.com))

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

## **Random generator License**

<http://www.agner.org/random>

Author: Agner Fog

Date created: 2001

Last modified: 2008-11-16

Project: randomc.h

Platform: Any C++

Description: Random Number generator of type 'Mersenne Twister'

This random number generator is described in the article by M. Matsumoto & T. Nishimura, in: ACM Transactions on Modeling and Computer Simulation, vol. 8, no. 1, 1998, pp. 3-30.

Details on the initialization scheme can be found at <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html>

Further documentation: The file ran-instructions.pdf contains further documentation and instructions.

Copyright 2001-2008 by Agner Fog.

GNU General Public License <http://www.gnu.org/licenses/gpl.html>