# User Guide for the SPHysics code



July 2007

M.G. Gesteira (mggesteira@uvigo.es )
B.D. Rogers (benedict.rogers@manchester.ac.uk)
R.A. Dalrymple (rad@jhu.edu)
A.J.C. Crespo (alexbexe@uvigo.es)
M. Narayanaswamy (muthu@jhu.edu)

## Acknowledgements

The development and application of SPHysics were partially supported by:

- Xunta de Galicia under project PGIDIT06PXIB383285PR.
- Office of Naval Research, Geosciences Program
- EPSRC Project Grant GR/S28310

## Abstract

This report documents the computer program SPHysics based on Smoothed Particle Hydrodynamics theory. The documentation provides a brief description of the governing equations and the different numerical schemes used to solve them. FORTRAN code is provided for two and three- dimensional versions of the model. Post-processing tools for MATLAB visualization are also provided. Finally, several working examples are documented to enable to user to test the program and verify that it is installed correctly.

# Contents

# 1. THEORY

## 1.1. The SPH method

The main features of the SPH method, which is based on integral interpolants, are described in detail in the following papers (Monaghan, 1982; Monaghan, 1992; Benz, 1990; Liu, 2003, Monaghan, 2005). Herein we will only refer to the representation of the constitutive equations in SPH notation. In SPH, the fundamental principle is to approximate any function $A(\vec{r})$ by

$$A(\vec{r}) = \int A(\vec{r}')\, W(\vec{r} - \vec{r}', h)\, d\vec{r}' \tag{1.1}$$

where $h$ is called the smoothing length and $W(\vec{r} - \vec{r}', h)$ is the weighting function or kernel. This approximation, in discrete notation, leads to the following approximation of the function at a particle (interpolation point) $\underline{a}$,

$$A(\vec{r}) = \sum_b m_b \frac{A_b}{\rho_b} W_{ab} \tag{1.2}$$

where the summation is over all the particles within the region of compact support of the kernel function., The mass and density are denoted by $m_b$ and $\rho_b$ respectively and $W_{ab} = W(\vec{r}_a - \vec{r}_b, h)$ is the weight function or kernel.

## 1.2. The weighting function or smoothing kernel

The performance of an SPH model is critically dependent on the choice of the weighting functions. They should satisfy several conditions such as positivity, compact support, and normalization. Also, $W_{ab}$ must be monotonically decreasing with increasing distance from particle $\underline{a}$ and behave like a delta function as the smoothing length, $h$, tends to zero (Monaghan, 1992; Benz, 1990; Liu, 2003). Kernels depend on the smoothing length, $h$, and the non-dimensional distance between particles given by $q = r / h$, $r$ being the distance between particles $\underline{a}$ and $\underline{b}$. The parameter $h$, often called influence domain or smoothing domain, controls the size of the area around particle $\underline{a}$ where contribution from the rest of the particles cannot be neglected.

In SPHysics, the user can choose from one of the following four different kernel definitions:

1) Gaussian:

$$\mathrm{W}_{ab} = \frac{1}{\alpha_D} \exp\left(-q^2\right) \tag{1.3}$$

where $\alpha_D$ is $1/(\pi h^2)$ in 2D and $1/(\pi^{3/2} h^3)$ in 3D

2) Quadratic:

$$W(q,h) = \alpha_D \left[ \frac{3}{16} q^2 - \frac{3}{4} q + \frac{3}{4} \right] \qquad 0 \le q \le 2 \tag{1.4}$$

where $\alpha_D$ is $1/(\pi h^2)$ in 2D and $1/(\pi h^3)$ in 3D

3) Cubic spline:

$$W(q,h) = \alpha_D \begin{cases} 1 - \frac{3}{2} q^2 + \frac{3}{4} q^3 & 0 \le q \le 1 \\ \frac{1}{4}(2-q)^3 & 1 \le q \le 2 \\ 0 & q \ge 2 \end{cases} \tag{1.5}$$

where $\alpha_D$ is $10/(7\pi h^2)$ in 2D and $1/(\pi h^3)$ in 3D.

4) Quintic (Wendland, 1995):

$$W(q,h) = \alpha_D \left( 1 - \frac{q}{2} \right)^4 (2q+1) \qquad 0 \le q \le 2 \tag{1.6}$$

where $\alpha_D$ is $7/(4\pi h^2)$ in 2D and $7/(8\pi h^3)$ in 3D.

The tensile correction (Monaghan, 2000) is automatically activated when using kernels whith first derivatives that go to zero with decreasing inter-particle spacing. In addition, the kernel has been modified following the constant correction proposed by Bonet and Kulasegaram (2000) in order to ensure that the normalization condition is satisfied, particularly near the free surface.

### 1.3. Momentum equation

The momentum conservation equation in a continuum field is

$$\frac{D\vec{v}}{Dt} = -\frac{1}{\rho} \vec{\nabla} P + \vec{g} + \vec{\Theta} \tag{1.7}$$

where $\vec{\Theta}$ refers to the diffusion terms.
Different approaches, based on various existing formulations of the diffusive terms, can be considered in the SPH method to describe the momentum equation. Three different options for diffusion can be used in SPHysics: (i) artificial viscosity, (ii) laminar viscosity and (iii) full viscosity (laminar viscosity+ Sub-Particle Scale (SPS) Turbulence):

#### 1.3.1. Artificial viscosity

The artificial viscosity proposed by Monaghan (1992) has been used very often due to its simplicity. In SPH notation, Eq. 1.7 can be written as

$$\frac{d\vec{v}_a}{dt} = -\sum_b m_b \left( \frac{P_b}{\rho_b^2} + \frac{P_a}{\rho_a^2} + \Pi_{ab} \right) \vec{\nabla}_a W_{ab} + \vec{g} \tag{1.8}$$

where $\vec{g}$ = (0, 0, -9.81) ms$^{-2}$ is the gravitational acceleration.
The pressure gradient term in symmetrical form is expressed in SPH notation as

$$\left( -\frac{1}{\rho} \vec{\nabla} \mathrm{P} \right)_a = -\sum_b \mathrm{m_b} \left( \frac{\mathrm{P_b}}{\rho_b^2} + \frac{\mathrm{P_a}}{\rho_a^2} \right) \vec{\nabla}_a \mathrm{W_{ab}} \tag{1.9}$$

with $P_k$ and $\rho_k$ are the pressure and density corresponding to particle $k$ (evaluated at $\underline{a}$ or $\underline{b}$).
$\Pi_{ab}$ is the viscosity term:

$$\Pi_{ab} = \begin{cases} \dfrac{-\alpha \overline{c_{ab}} \mu_{ab}}{\overline{\rho_{ab}}} & \vec{v}_{ab} \vec{r}_{ab} < 0 \\ 0 & \vec{v}_{ab} \vec{r}_{ab} > 0 \end{cases} \tag{1.10}$$

with $\mu_{ab} = \dfrac{h \vec{v}_{ab} \vec{r}_{ab}}{\vec{r}_{ab}^2 + \eta^2}$; where $\vec{r}_{ab} = \vec{r}_a - \vec{r}_b$, $\vec{v}_{ab} = \vec{v}_a - \vec{v}_b$; being $\vec{r}_k$ and $\vec{v}_k$ the position and the velocity corresponding to particle $k$ ($\underline{a}$ or $\underline{b}$); $\overline{c_{ab}} = \dfrac{c_a + c_b}{2}$. $\eta^2 = 0.01h^2$, $\alpha$ is a free parameter that can be changed according to each problem.

### 1.3.2. Laminar viscosity

The momentum conservation equation with laminar viscous stresses is given by

$$\frac{D\vec{v}}{Dt} = -\frac{1}{\rho} \vec{\nabla} P + \vec{g} + \upsilon_O \nabla^2 \vec{v} \tag{1.11}$$

where the laminar stress term simplifies(Morris *et al*., 1997) to:

$$\left( \upsilon_0 \nabla^2 \vec{\mathrm{v}} \right)_a = \sum_b \mathrm{m_b} \left( \frac{4 \upsilon_0 \vec{\mathrm{r}}_{ab} \vec{\nabla}_{ab}}{(\rho_a + \rho_b) \left| \vec{\mathrm{r}}_{ab} \right|^2} \right) \vec{\nabla}_a \mathrm{W_{ab}} \tag{1.12}$$

where $\upsilon_0$ is the kinetic viscosity of laminar flow ($10^{-6} m^2/s$).
So, in SPH notation, Eq. 1.11 can be written as:

$$\frac{\mathrm{d}\vec{v}_a}{\mathrm{dt}} = -\sum_b \mathrm{m_b} \left( \frac{\mathrm{P_b}}{\rho_b^2} + \frac{\mathrm{P_a}}{\rho_a^2} \right) \vec{\nabla}_a \mathrm{W_{ab}} + \vec{g} + \sum_b \mathrm{m_b} \left( \frac{4 \upsilon_0 \vec{\mathrm{r}}_{ab} \vec{\nabla}_{ab}}{(\rho_a + \rho_b) \left| \vec{\mathrm{r}}_{ab} \right|^2} \right) \vec{\nabla}_a \mathrm{W_{ab}} \tag{1.13}$$

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

The Sub-Particle Scale (SPS) approach to modeling turbulence was first described by Gotoh *et al*. (2001) to represent the effects of turbulence in their MPS model. The momentum conservation equation is,

$$\frac{D\vec{v}}{Dt} = -\frac{1}{\rho} \vec{\nabla} P + \vec{g} + \upsilon_0 \nabla^2 \vec{v} + \frac{1}{\rho} \vec{\nabla} \bar{\tau} \tag{1.14}$$

where the laminar term can be treated following Eq. 1.12 and $\bar{\tau}$ represents the SPS stress tensor.

The eddy viscosity assumption (Boussinesq's hypothesis) is often used to model the SPS stress tensor using Favre-averaging (for a compressible fluid): $\frac{\tau_{ij}}{\rho} = 2\nu_t S_{ij} - \frac{2}{3}k\delta_{ij} - \frac{2}{3}C_I\Delta^2\delta_{ij}|S_{ij}|^2$, where $\vec{\tau}_{ij}$ is the sub-particle stress tensor, $\nu_t = [\min(C_s\Delta l)]^2|S|$ the turbulence eddy viscosity, $k$ the SPS turbulence kinetic energy, Cs the Smagorinsky constant (0.12), $C_I$ = 0.0066, $\Delta l$ the particle-particle spacing and $|S| = (2S_{ij}S_{ij})^{1/2}$, $S_{ij}$ the element of SPS strain tensor.

So, following (Dalrymple & Rogers, 2006), Eq. 1.14 can be written in SPH notation as

$$\frac{d\vec{v}_a}{dt} = -\sum_b m_b\left(\frac{P_b}{\rho_b^2} + \frac{P_a}{\rho_a^2}\right)\vec{\nabla}_a W_{ab} + \vec{g}$$
$$+ \sum_b m_b\left(\frac{4\upsilon_0 \vec{r}_{ab}\vec{v}_{ab}}{(\rho_a + \rho_b)|\vec{r}_{ab}|^2}\right)\vec{\nabla}_a W_{ab} + \qquad (1.15)$$
$$+ \sum_b m_b\left(\frac{\tau_b}{\rho_b^2} + \frac{\tau_a}{\rho_a^2}\right)\vec{\nabla}_a W_{ab}$$

### 1.4. Continuity equation

Changes in the fluid density are calculated in SPHysics using

$$\frac{d\rho_a}{dt} = \sum_b m_b \vec{v}_{ab}\vec{\nabla}_a W_{ab} \qquad (1.16)$$

instead of using a weighted summation of mass terms (Monaghan, 1992), since it is known to result in an artificial density decrease near fluid interfaces.

### 1.5. Equation of state

The fluid in the SPH formalism is treated as weakly compressible. This facilitates the use of an equation of state to determine fluid pressure, which is much faster than solving an equation such as the Poisson´s equation. However, the compressibility is adjusted to slow the speed of sound so that the time step in the model (using a Courant condition based the speed of sound) is reasonable. Another limitation on the compressibility is imposed by the fact that the sound speed should be about ten times faster than the maximum fluid velocity, thereby keeping density variations to within less than 1%.

Following (Monaghan *et al*., 1999; Batchelor, 1974), the relationship between pressure and density is assumed to follow the expression

$$P = B\left[\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right] \tag{1.17}$$

where $\gamma = 7$ and, $B = c_0^2 \rho_0 / \gamma$ being $\rho_0 = 1000$ kg m$^{-3}$ the reference density and $c_o = c(\rho_o) = \sqrt{(\partial P / \partial \rho)}\Big|_{\rho_o}$ the speed of sound at the reference density.

### 1.6. Moving the particles

Particles are moved using the XSPH variant (Monaghan, 1989)

$$\frac{d\vec{r}_a}{dt} = \vec{v}_a + \varepsilon \sum_b \frac{m_b}{\overline{\rho_{ab}}} \vec{v}_{ab} W_{ab} \tag{1.18}$$

where $\varepsilon$=0.5 and $\overline{\rho_{ab}} = (\rho_a + \rho_b)/2$. This method moves *a* particle with a velocity that is close to the average velocity in its neighborhood.

### 1.7. Thermal energy

The thermal energy associated to each particle is calculated using the expression given by Monaghan (1994)

$$\frac{de_a}{dt} = \frac{1}{2} \sum_b m_b \left( \frac{P_a}{\rho_a^2} + \frac{P_b}{\rho_b^2} + \Psi_{ab} \right) \vec{v}_{ab} \vec{\nabla}_a W_{ab} \tag{1.19}$$

where $\Psi_{ab}$ refers to viscosity terms, which can be calculated using the different approaches mentioned above.

# 2. IMPLEMENTATION

## 2.1. Time stepping

Two numerical schemes are implemented in SPHysics: (i) the Predictor-Corrector algorithm described by Monaghan (1989); (ii) the Verlet algorithm (Verlet, 1967).
Consider the momentum (1.7), density (1.16), position (1.18) and density of energy (1.19) equations in the following form

$$\frac{\mathrm{d}\vec{v}_a}{\mathrm{dt}} = \vec{F}_a \tag{2.1a}$$

$$\frac{\mathrm{d}\rho_a}{\mathrm{dt}} = D_a \tag{2.1b}$$

$$\frac{\mathrm{d}\vec{r}_a}{\mathrm{dt}} = \vec{V}_a \tag{2.1c}$$

$$\frac{\mathrm{d}e_a}{\mathrm{dt}} = E_a \tag{2.1d}$$

where $\vec{V}_a$ represents the velocity contribution from particle *a* and from neighboring particles (XSPH correction).

### 2.1.1. Predictor-Corrector scheme

This scheme predicts the evolution in time as,

$$\begin{aligned} \vec{v}_a^{n+1/2} &= \vec{v}_a^n + \frac{\Delta t}{2}\vec{F}_a^n ;\ \rho_a^{n+1/2} = \rho_a^n + \frac{\Delta t}{2} D_a^n \\ \vec{r}_a^{n+1/2} &= \vec{r}_a^n + \frac{\Delta t}{2}\vec{V}_a^n ;\ e_a^{n+1/2} = e_a^n + \frac{\Delta t}{2} E_a^n \end{aligned} \tag{2.2}$$

calculating $P_a^{n+1/2} = f(\rho_a^{n+1/2})$ according to Eq. 1.17.
These values are then corrected using forces at the half step

$$\begin{aligned} \vec{v}_a^{n+1/2} &= \vec{v}_a^n + \frac{\Delta t}{2}\vec{F}_a^{n+1/2} ;\ \rho_a^{n+1/2} = \rho_a^n + \frac{\Delta t}{2} D_a^{n+1/2} \\ \vec{r}_a^{n+1/2} &= \vec{r}_a^n + \frac{\Delta t}{2}\vec{V}_a^{n+1/2} ;\ e_a^{n+1/2} = e_a^n + \frac{\Delta t}{2} E_a^{n+1/2} \end{aligned} \tag{2.3}$$

Finally, the values are calculated at the end of the time step following:

$$\begin{aligned} \vec{v}_a^{n+1} &= 2\vec{v}_a^{n+1/2} - \vec{v}_a^n ;\ \rho_a^{n+1} = 2\rho_a^{n+1/2} - \rho_a^n \\ \vec{r}_a^{n+1} &= 2\vec{r}_a^{n+1/2} - \vec{r}_a^n ;\ e_a^{n+1} = 2e_a^{n+1/2} - e_a^n \end{aligned} \tag{2.4}$$

Finally, the pressure is calculated from density using $P_a^{n+1} = f(\rho_a^{n+1})$.

### 2.1.2. Verlet scheme

This time stepping algorithm, to discretize Equations 3.1a-d, is split into two parts:
In general, variables are calculated according to

$$\vec{v}_a^{\,n+1} = \vec{v}_a^{\,n-1} + 2\Delta t\vec{F}_a^{\,n};\ \rho_a^{n+1} = \rho_a^{n-1} + 2\Delta t D_a^n$$
$$\vec{r}_a^{\,n+1} = \vec{r}_a^{\,n} + \Delta t\vec{V}_a^{\,n} + 0.5\Delta t^2\vec{F}_a^{\,n};\ e_a^{n+1} = e_a^{n-1} + 2\Delta t E_a^n \qquad (2.5)$$

Once every M time steps (M on the order of 50 time steps), variables are calculated according to

$$\vec{v}_a^{\,n+1} = \vec{v}_a^{\,n} + \Delta t\vec{F}_a^{\,n};\ \rho_a^{n+1} = \rho_a^n + \Delta t D_a^n$$
$$\vec{r}_a^{\,n+1} = \vec{r}_a^{\,n} + \Delta t\vec{V}_a^{\,n} + 0.5\Delta t^2\vec{F}_a^{\,n};\ e_a^{n+1} = e_a^n + \Delta t E_a^n \qquad (2.6)$$

This is to stop the time integration diverging since the equations are no longer coupled.

## 2.2. Variable time step

Time-step control is dependant on the CFL condition, the forcing terms and the viscous diffusion term (Monaghan; 1989). A variable time step $\delta t$ is calculated according to Monaghan and Kos (1999):

$$\Delta t = 0.3\cdot\min(\Delta t_f, \Delta t_{cv});\ \Delta t_f = \min_a\left(\sqrt{h/|f_a|}\right);\ \Delta t_{cv} = \min_a \frac{h}{c_s + \max_b\left|\frac{h\vec{v}_{ab}\vec{r}_{ab}}{\vec{r}_{ab}^{\,2}}\right|} \qquad (2.7)$$

Here $\Delta t_f$ is based on the force per unit mass $|f_a|$, and $\Delta t_{cv}$ combines the Courant and the viscous time-step controls.

## 2.3 Computational efficiency: link list.

In the code the computational domain is divided in square cells of side 2h (see Figure 2.1). following Monaghan and Latanzio (1985). Thus, for a particle located inside a cell, only the interactions with the particles of neighboring cells need to be considered. In this way the number of calculations per time step and, therefore, the computational time diminish considerably, from $N^2$ operations to $N\log N$, $N$ being the number of particles.

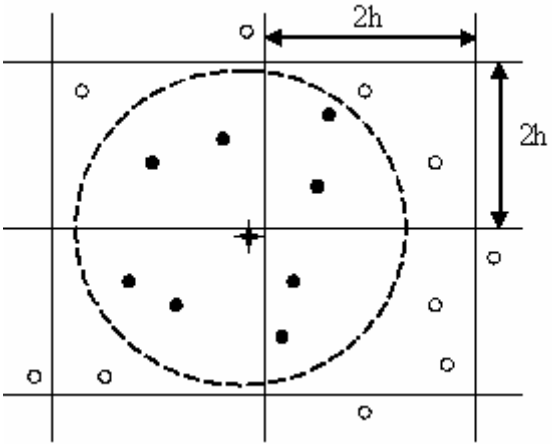


**Figure 2.1:** Set of neighboring particles in 2D. The particle marked with a star corresponds to particle *a*. The possible neighbors in adjacent cells are marked with a dot. Particle *a* only interacts with particles inside the dashed circle (solid dots)

The SPHysics code in 2D sweeps through the grid along the *x*-direction, for each *z*-level. Around each cell, the E, N, NW & NE neighbouring cells are checked to minimise repeating the particle interactions. Thus, for example, when the centre cell is i=5 and k=3 (see scheme in Figure 2.2), the target cells are (5,4), (4,4), (6,4) and (6,3). The rest of the cells were previously considered through the sweeping (e.g. the interaction between cell (5,3) and (5,2) was previously accounted when (5,2) was considered to be the centre cell).

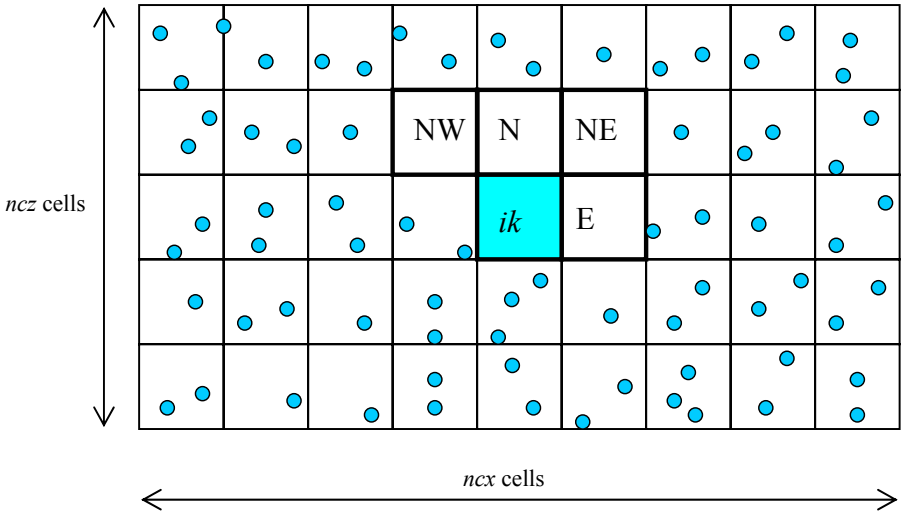


**Figure 2.2:** Sweeping through grid cells in 2D. Starting from the lower left corner, particles inside the center cell *ik* interact with adjacent cells only in E, N, NW and NE directions. The interactions with the rest of the cells W, S, SW & SE directions were previously computed using reverse interactions.

A similar protocol is used in 3D calculations (Figure 2.3).

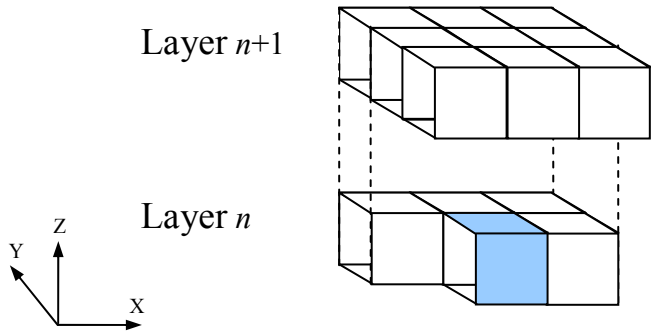


**Figure 2.3:** Sweeping through grid cells in 3D. Only 13 out of 26 possible neighboring cells are considered when centered on a particular *ijk* cell. The rest were previously

considered when centered on adjacent cells using reverse interactions.

Two link lists are considered in SPHysics. The first one tracks the boundary particles and it is partially upgraded every time step. This is due to the fact that the only boundary particles that change their position in time are the ones that describe moving objects such as gates and wavemakers. The second link list corresponds to fluid particles and is completely updated every time step.

## 2.4. Boundary conditions

Three boundary conditions have been implemented in SPHysics: (i) Dynamic Boundary conditions (Crespo et al, 2007; Dalrymple and Knio, 2000); (ii) Repulsive boundary conditions (Monaghan & Kos, 1999) and (iii) Periodic open boundary conditions:

### 2.4.1. Dynamic Boundary conditions

In this method, boundary particles are forced to satisfy the same equations as fluid particles. Thus, they follow the momentum equation (Eq. 1.7), the continuity equation (Eq. 1.16), the equation of state (Eq. 1.17), and the energy equation (Eq. 1.19). However, they do not move according to Eq. 1.18. They remain fixed in position (fixed boundaries) or move according to some externally imposed function (moving objects like gates, wavemakers …).
Boundary particles are organized in a staggered manner (see Fig. 2.4):

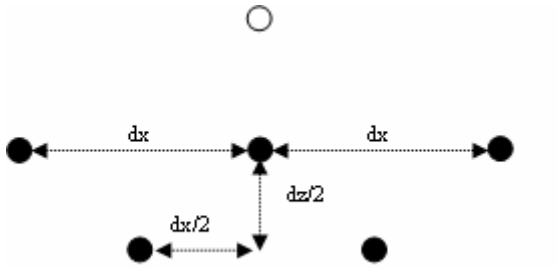


**Figure 2.4:** 2D sketch of the interaction between a fluid particle (empty circle) and a set of boundary particles (full circles). The boundary particles are placed in a staggered manner.

When a fluid particle approaches a boundary the density of the boundary particles increases according to Eq. 1.16 resulting in a pressure increase following Eq. 1.17. Thus, the force exerted on the fluid particle increases due to the pressure term $\left(P/\rho^2\right)$ in momentum equation (see Eq. 1.8, 1.13 or 1.15). This mechanism is depicted in a simple example where a fluid particle approaches the bottom of the tank. When the distance between the boundary particle and the fluid particle becomes smaller than $2h$, the density, pressure and force exerted on the incoming particle increase generating the repulsion mechanism (see Fig. 2.5). The normalized pressure

term, $NPT_z = \left(P/\rho^2\right)_z / \left(P/\rho^2\right)_R$, is represented in Fig. 5c, where $z$ refers to the distance from the incoming fluid particle to the wall and $R$ the minimum distance to the wall attained by the incoming particle. The wall is composed of boundary particles.

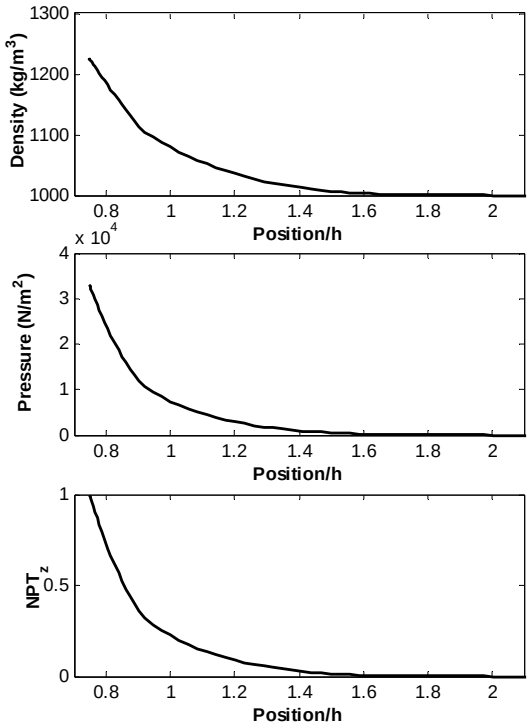


**Figure 2.5:** Variation of density (a), pressure (b) and normalized pressure term (c) for a moving particle approaching a solid boundary. Calculations were run without viscosity.

2.4.2. Repulsive boundary conditions.

This boundary condition was developed by Monaghan (1994) to ensure that a water particle can never cross a solid boundary. In this case, analogous to inter-molecular forces, the particles that constitute the boundary exert central forces on the fluid particles. Thus, for a boundary particle and a fluid particle separated a distance $r$ the force for unit of mass has the form given by the Lennard-Jones potential. In a similar way, other authors (Peskin, 1977) express this force assuming the existence of forces in the boundaries, which can be described by a delta function. This method was refined in Monaghan and Kos (1999) by means of an interpolation process, minimizing the inter-spacing effect of the boundary particles on the repulsion force of the wall.

Following this approach, the force experienced by a water particle, $\vec{f}$, acting normal to the wall, is given by

$$\vec{f} = \vec{n} R(\psi) P(\xi) \varepsilon\left(z, u_{\perp}\right) \tag{2.8}$$

where $\vec{n}$ is the normal of the solid wall. The distance $\psi$ is the perpendicular distance of the particle from the wall, while $\xi$ is the projection of interpolation location $\xi_i$ onto the

chord joining the two adjacent boundary particles and $u_\perp$ is the velocity of the water particle projected onto the normal.. The repulsion function, $R(\psi)$, is evaluated in terms of the normalized distance from the wall, $q = \psi/2h$, as

$$R(\psi) = A\frac{1}{\sqrt{q}}(1-q) \tag{2.9}$$

where the coefficient $A$ is

$$A = \frac{1}{h}0.01c_i^2 \tag{2.10}$$

$c_i$ being the speed of sound corresponding to particle $i$.

The function $P(\xi)$ is chosen so that a water particle experiences a constant repulsive force as it travels parallel to the wall

$$P(\xi) = \frac{1}{2}\left(1+\cos\left(\frac{2\pi\xi}{\Delta b}\right)\right) \tag{2.11}$$

where $\Delta b$ is the distance between any two adjacent boundary particles. Finally, the function $\varepsilon(z,u_\perp)$ is a modification to Monaghan and Kos's original suggestion and adjusts the magnitude of the force according to the local water depth and velocity of the water particle normal to the boundary

$$\varepsilon(z,u_\perp) = \varepsilon(z)+\varepsilon(u_\perp) \tag{2.12}$$

where

$$\varepsilon(z) = \begin{cases} 0.02 & z \geq 0 \\ |z/h_o| + 0.02 & 0 > z \geq -h_o \\ 1 & |z/h_o| > 1 \end{cases} \tag{2.13}$$

and

$$\varepsilon(u_\perp) = \begin{cases} 0 & u_\perp > 0 \\ |20u_\perp|/c_o & |20u_\perp| < c_o \\ 1 & |20u_\perp| > c_o \end{cases} \tag{2.14}$$

In Equations 2.12-2.14, $z$ is the elevation above the local still-water level $h_o$, $u_\perp = (\bar{v}_{WP} - \bar{v}_{BP})\cdot\bar{n}$, where the subscripts WP and BP refer to water and boundary particles respectively, and $c_o = \sqrt{B\gamma/\rho_o}$ the speed of sound at the reference density.

The system of normals requires each boundary particle (BP) to know the coordinates of its adjacent BPs. In a two-dimensional situation as shown in Figure 2.6a, the boundary particle $i$ is surrounded by BPs $i$-1 and $i$+1 so that the tangential vector is given by $\vec{t} = (\vec{r}_{i+1} - \vec{r}_{i-1})/|\vec{r}_{i+1} - \vec{r}_{i-1}|$ so that the normal is then found from $\vec{n}\vec{t} = 0$. The three-dimensional situation is shown in Figure 2.6b where BP $i$ also has adjacent neighbours $j$-1 and $j$+1. The coordinates of these adjacent BPs are required to calculate the tangents and normal: $\vec{t} = (\vec{r}_{i+1} - \vec{r}_{i-1})/|\vec{r}_{i+1} - \vec{r}_{i-1}|$, $\vec{s} = (\vec{r}_{j+1} - \vec{r}_{j-1})/|\vec{r}_{j+1} - \vec{r}_{j-1}|$ and $\vec{n} = \vec{t}\times\vec{s}$.

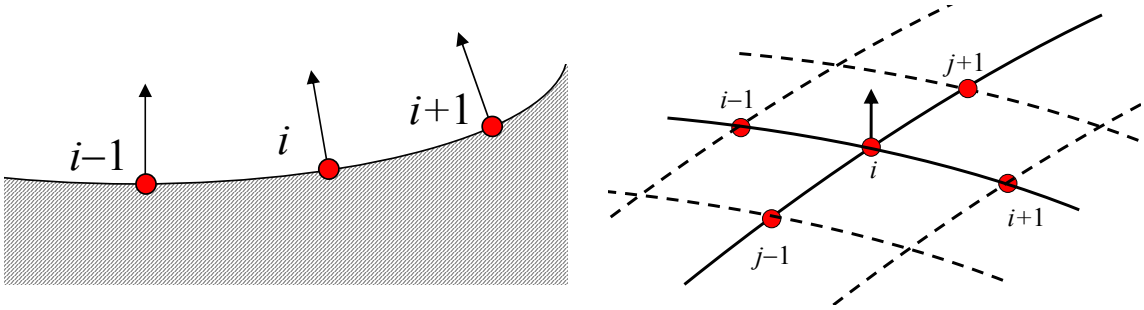


**Figure 2.6:** Location of adjacent boundary particles. (a) 2-D boundary particles and adjacent neighbours; (b) 3-D boundary particles and adjacent neighbours

### 2.4.3 Periodic Open Boundary Conditions

In the present release version of the code, open boundaries are implemented using periodic boundary conditions. Essentially this means that particles near an open lateral boundary interact with the particles near the complementary open lateral boundary on the other side of the domain. This situation is shown in Figure 2.7 where water particle *i* lies near the top boundary and therefore its area of influence (or kernel support) extends beyond the lateral boundary. With periodic boundaries, this area of influence is continued through the bottom boundary so that particles interact near the bottom boundary within the extended support interact with particle *i*.

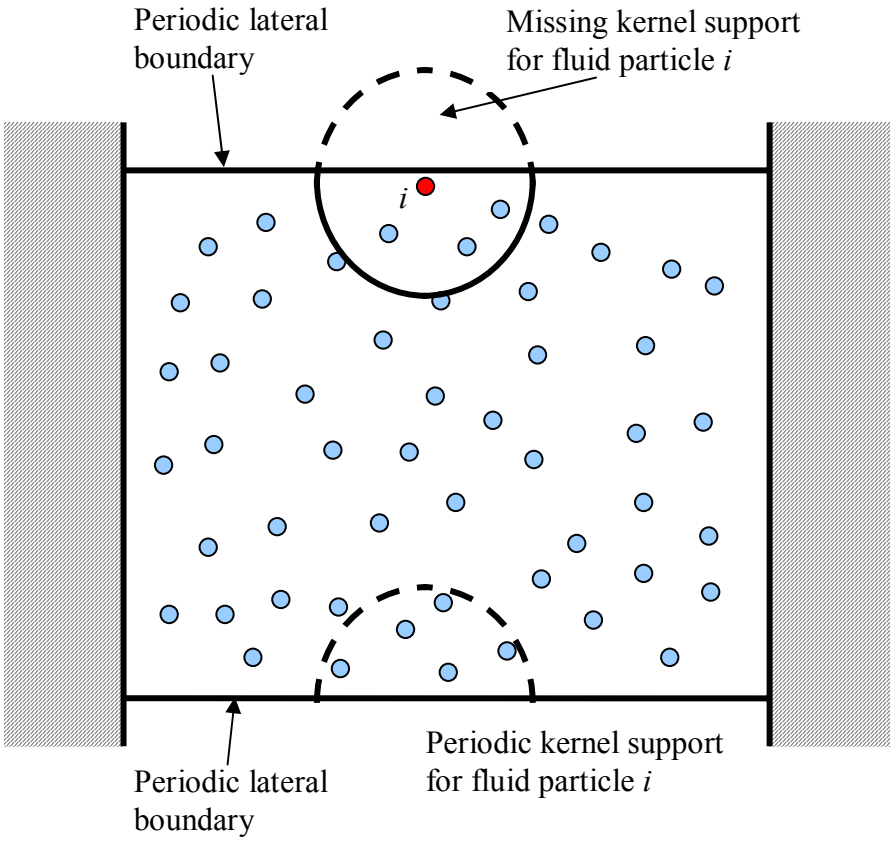

**Figure 2.7:** Periodic lateral boundaries: area of influence (kernel support) for particle *i* extend beyond top lateral boundary and is continued through periodic bottom boundary.

## 2.5. Checking limits

In SPH, fluid particles can leave the computational domain in different ways, both physically and non-physically. Once the particle is outside the domain, it is continuously accelerated under the effect of gravity. These particles must be identified and removed from the run to avoid spurious effects. The treatment of these particles depends on the way they leave the computational domain.

### 2.5.1. Fixing the limits

Limits of the computational domain are fixed at the beginning of the run depending on the initial position of the particles. In each direction: $\Lambda_k^{\max} = \max(\Lambda_k(i, t=0)) + h$ and $\Lambda_k^{\min} = \min(\Lambda_k(i, t=0)) - h$, where $\Lambda_k$ refers to the direction (X, Y or Z) and $i \in [1, N]$ refers to all particles. These limits fix the number of cells of dimensions *2h×2h×2h* (in 3D) used to cover the computational domain.
Limits in X, Y and $Z^-$ directions remain unchanged during the run. The limit in $Z^+$ is allowed to vary in time, since fluid can splash and surpass the upper limit of the container. All limits are checked at every time step.

### 2.5.2. Changing the limits in $Z^+$

When a fluid particle surpasses the upper limit in the vertical Z direction, the computational domain is extended and new cells are created. The number of boundary particles inside these new cells is immediately set to zero. Fluid particles can then occupy these cells depending on their position. The number of cells in the vertical is thus dynamically modified depending on the position of the highest fluid particle. This generates important savings in execution time, since no redundant cell computations are performed.

When a part of a moving object surpasses the initial upper limit that part of the object it is stopped at that upper limit for the rest of the run.

### 2.5.3. Limits in X, Y or $Z^-$ directions

A fluid particle can surpass the initial limits in X, Y or $Z^-$ direction due to several reasons. Dynamic boundary particles are not completely impermeable. Hence a single particle, accelerated by collision in the proximity of a boundary, can possibly penetrate the boundary. On the other hand, the fluid can collide with the container overtopping

the lateral walls. Once the fluid leaves the container, fluid particles are continuously accelerated by gravity away from the domain of interest, giving rise to very small time steps according to Eq. 3.7 slowing down the calculations.

The position of particles is checked every time step, in such a way that when a particle is found outside the container, the particle is replaced at a previously defined position outside the container and marked with a *flag*. Thus, although the particle is not eliminated from the list (the number of particles, N, remains constant) the particle is not allowed to move with time.

# 3. USER'S MANUAL

## 3.1. Installation

Two versions of SPHysics are available in this release:

- SPHysics_2D. The computational domain is considered to be 2D, where *x* corresponds to the horizontal direction and *z* to the vertical direction.
- SPHysics_3D. The computational domain is fully 3D. *x* and *y* are the horizontal directions and *z* the vertical direction.

SPHysics is distributed in a compressed file (gz or zip). The directory tree shown in Figure 3.1 can be observed after uncompressing the package

In that figure, the following directories can be observed both in 2D and in 3D.

***source*** contains the FORTRAN codes. This directory contains two subdirectories:

SPHysicsgen: contains the FORTRAN codes to create the initial conditions of the run.

SPHysics: contains the FORTRAN source codes of SPH.

***execs*** contains all executable codes.

***run_directory*** is the directory created to run the model. The different subdirectories *Case1, …, CaseN* placed in this directory correspond to the different working cases to be created by the user. Input and output files are written in these directories

***Post-Processing*** this directory contains MATLAB codes to visualize results.

## 3.2. Program Outline

Both the 2D and 3D version consist in two programs, which are run separately and in the following order.

2D Code:

SPHysicsgen_2D: Creates the initial conditions and files for a given case.

SPHysics_2D: Runs the selected case with the initial conditions created by SPHysicsgen_2D code.

3D Code:

SPHysicsgen_3D: Creates the initial conditions and files for a given case.

SPHysics_3D: Runs the selected case with the initial conditions created by SPHysicsgen_3D code.

In general, 2D or 3D appended to the source file name means that the source is suited for 2D or 3D calculations.

In the remainder of this document, SPHysicsgen and SPHysics, when used, refer to both the aforementioned 2D and 3D programs for convenience. For example, SPHysicsgen will refer to both SPHysicsgen_2D and SPHysicsgen_3D.

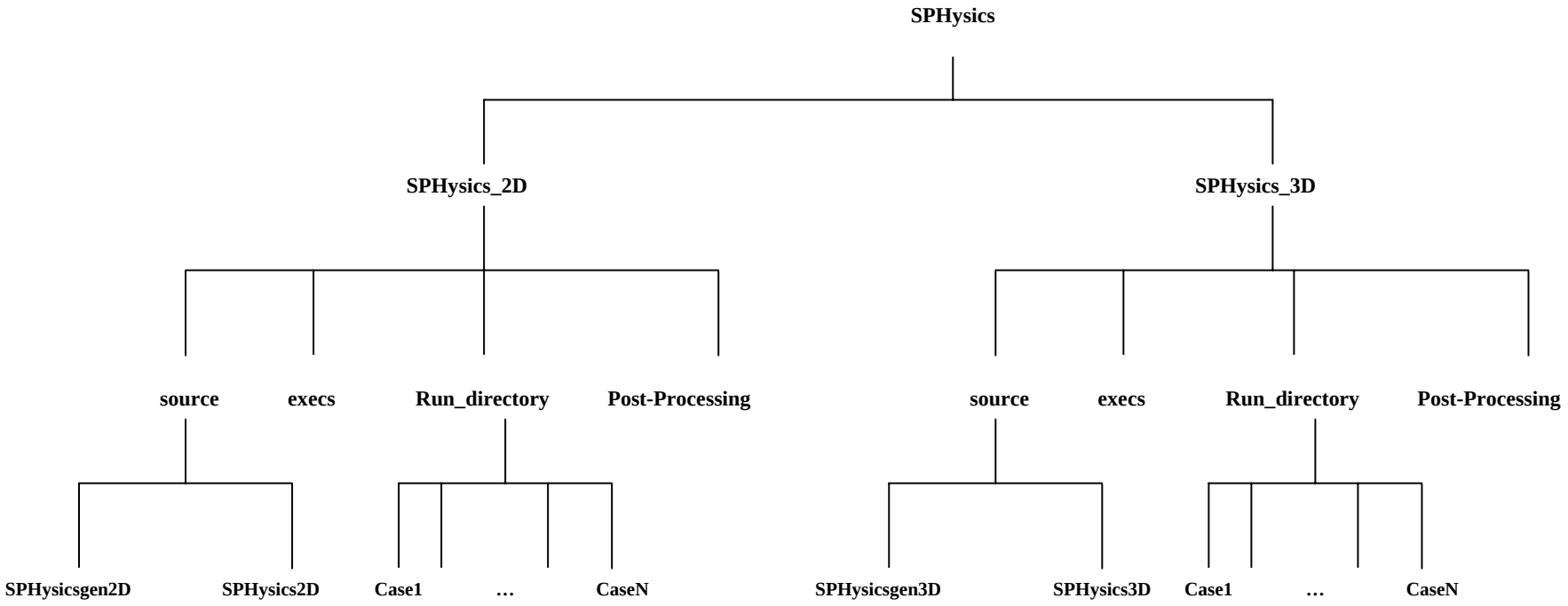


**Figure 3.1.** Directory tree.

3.2.1. SPHysicsgen

All subroutines are included in two source files (SPHysicsgen_2D.f or SPHysicsgen_3D.f), depending on the nature two or three- dimensional of the calculation. Each source uses a different common file, where most of the variables are stored. The common files are common.gen2D (in 2D) and common.gen3D (in 3D). Both versions (2D and 3D) can be compiled by the user with any FORTRAN compiler and the resulting executable file is placed in subdirectory *execs*.

SPHysicsgen plays a dual role: (i) Creating the MAKEFILE to compile SPHysics; and (ii) Creating the output files that will be the input files to be read by SPHysics. These files contain information about the geometry of the domain, the distribution of particles and the different running options.

In Windows for example, *SPHysicsgen.exe* can be executed using one of the following two commands,

**1**. *SPHysicsgen.exe <input_file >output_file*

*input_file* is the general name (any name can be used) of the file containing the running options. Different examples of *input_file* will be shown in next section.

*output_file* is the general name (any name can be used) of the file containing general information about the run. This file is never read by the rest of the code and only serves to provide information to the user.

**2**. *SPHysicsgen.exe*

In this case, data about the run must then be provided by the user by means of the keyboard and the information about the run appears on the screen. This option can be used by beginners to get familiarized with the different options.

3.2.1.1. Creating compiling options

The compilation of SPHysics code depends on the nature of the problem under consideration and on the particular features of the run. Thus, the user can chose the options that are better suited to any particular problem and only those options will be included in the executable versions of *SPHysics*. This protocol speeds up calculations since the model is not forced to make time consuming logical decisions.

Both in 2D and 3D the following compiling options can be considered:

i) Kind of kernel: (1=Gaussian; 2= Quadratic; 3= Cubic Spline; 5= Quintic).
ii) Viscosity treatment: (1= Artificial viscosity; 2= Laminar viscosity; 3=Laminar viscosity +SPS).
iii) Time stepping: (1= Predictor- Corrector algorithm; 2= Verlet algorithm).
iv) Boundary conditions: (1= Repulsive BCs; 2= Dynamic Bcs).
v) Choice of compilers: (1=gfortran; 2=ifort; 3= HP 6.5). Please refer to section 4.1.1 for details on running the code on Windows using HP 6.5, and section

4.1.2 to use gfortran and ifort compilers. The gfortran and ifort compilers have only been tested on Linux and Mac OSX platforms.

3.2.1.2. Input files

Different examples of input files will be shown in next section, where several test cases will be described.

3.2.1.3. Output files

As we mentioned above, different output files are created by *SPHysicsgen*. These files can be used either by the *SPHysics executable* as input files or by MATLAB codes to visualize results (different MATLAB codes are provided in */Post-processing* subdirectory.

SPHysics.mak

Compiling file created by the executable *SPHysicsgen*. It depends on the running options defined by *input_file*. It was prepared for COMPAQ VISUAL FORTRAN, IFORT and GFORTRAN although it can be adapted to other compilers.

INDAT

Created by *SPHysicsgen*

Read by SPHysics code at GETDATA (see subsection 3.2.2.3).

UNIT=11

The file contains the following variables:

| | |
|---|---|
| i_kernel | vlx |
| i_algorithm | vly |
| i_densityFilter | vlz |
| i_viscos | dx |
| iBC | dy |
| i_periodicOBs(1) | dz |
| i_periodicOBs(2) | h |
| i_periodicOBs(3) | np |
| lattice | nb |
| h_SWL | nbf |
| B | ivar_dt |
| gamma | dt |
| coef | tmax |
| eps | out |
| rho0 | trec_ini |
| viscos_val | dtrec_det |
| visc_wall | t_sta_det |
| | t_end_det |

Description:

i_kernel: Kind of kernel (1=Gaussian; 2= Quadratic; 3= Cubic Spline; 5= Quintic).

i_algorithm: Kind of algorithm (1= Predictor- Corrector algorithm; 2= Verlet algorithm).

i_densityFilter: Use of a Shepard filter (1=Yes).

i_viscos: Viscosity definition 1= Artificial; 2= Laminar; 3= Laminar + SPS

IBC: Boundary conditions. 1=Monaghan repulsive forces; 2= Dynamic boundary conditions.

i_periodicOBs(1): Periodic Lateral boundaries in *x* direction? (1=yes)

i_periodicOBs(2): Periodic Lateral boundaries in *y* direction? (1=yes)

i_periodicOBs(3): Periodic Lateral boundaries in *z* direction? (1=yes)

lattice: Lattice: (1) SC; (2) BCC

h_SWL: Still water level (m).

B: Parameter in Equation of State (Monaghan and Koss, 1999).

gamma: Parameter in Equation of State (Monaghan and Koss, 1999) (Default value 7).

Coef: Coeffficient to calculate the smoothing length (h) in terms of dx,dy,dz

h=coefficient*sqrt(dx*dx+dy*dy+dz*dz)

eps: Epsilon parameter in XSPH approach (Default value 0.5).

rho0: Reference density (Default value 1000 $kgm^{-3}$).

viscos_val: Viscosity parameter, it corresponds to α (Monaghan and Koss, 1999) if *i_viscos* = 1 and to ν (kinematical viscosity) if *i_viscos* = 2 or 3.

visc_wall: Wall viscosity value for Repulsive Force BC

vlx medium extent in X direction.

vly medium extent in Y direction. It is set to zero when *IDIM*= 2.

vlz medium extent in Z direction.

dx: Initial interparticle spacing in *x* direction.

dy: Initial interparticle spacing in *y* direction.

dz: Initial interparticle spacing in *z* direction.

h: Smoothing length.

np: Total number of particles.

nb: Number of boundary particles.

nbf: Number of fixed boundary particles. Note that boundary particles can be fixed or move according to some external dependence (e.g. gates, wavemakers).

ivar_dt: Variable time step calculated when *ivar_dt*=1.

dt: Initial time step. It is kept throughout the run when *ivar_dt*=0.

tmax: RUN duration (in seconds)

out: Recording time step (in seconds). The position, velocity, density, pressure and mass of every particle is recorded in PART file every *out* seconds.

trec_ini: Initial recording time.

dtrec_det: Detailed recording step.

t_sta_det: Start time in detailed recording.

t_end_det: End Time in detailed recording.

IPART

Created by *SPHysicsgen*.

Read by SPHysics code at GETDATA (see subsection 3.2.2.3).

UNIT=13

The file contains the following variables recorded at *time=0*:

In 2D version

xp(1) zp(1) up(1) wp(1) rhop(1) p(1) pm(1) vorty(1)
xp(2) zp(2) up(2) wp(2) rhop(2) p(2) pm(2) vorty(2)
…….
…….
xp(np) zp(np) up(np) wp(np) rhop(np) p(np) pm(np) vorty(np)

In 3D version

xp(1) yp(1) zp(1) up(1) vp(1) wp(1) rhop(1) p(1) pm(1) vortx(1) vorty(1) vortz(1)
xp(2) yp(2) zp(2) up(2) vp(2) wp(2) rhop(2) p(2) pm(2) vortx(2) vorty(2) vortz(2)
…… ……..
…… ……..
xp(np) yp(np) zp(np) up(np) vp(np) wp(np) rhop(np) p(np) pm(np) vortx(np) vorty(np) vortz(np)

Description:

xp(i) Position in *x* direction of particle i.
yp(i) Position in *y* direction of particle i.
zp(i) Position in *z* direction of particle i.
up(i) Velocity in *x* direction of particle i.
vp(i) Velocity in *y* direction of particle i.
wp(i) Velocity in *z* direction of particle i.
rhop(i) Density of particle i.
p(i) Pressure at particle i.
pm(i) Mass of particle i.
vortx(i),vorty(i),vortz(i) correspond to vorticity in *x*,*y* and *z* constant planes

MATLABIN

Created by *SPHysicsgen*.

To be used by MATLAB codes for graphical representation.

UNIT=8

The file contains the following variables:

np
vlx
vly
vlz
out
nb

Description:

vlx medium extent in *x* direction.

vly medium extent in *y* direction. It is set to zero when *IDIM*= 2.

vlz medium extent in *z* direction.

The rest of the variables were previously described.

NORMALS

Created by *SPHysicsgen*.

To be used by SPHysics code when *IBC*=1. It contents the normal and tangent vectors to each boundary particle.

UNIT=21

The file contains the following variables:

In 2D version

xnb(i),znb(i),
iBP_Pointer_Info(i,1), iBP_Pointer_Info(i,2), iBP_Pointer_Info(i,3),iBP_Pointer_Info(i,4),
BP_xz_Data(i,1), BP_xz_Data(i,2)

In 3D version

xnb(i),ynb(i),znb(i),xtb(i),ytb(i),ztb(i),xsb(i),ysb(i),zsb(i),
iBP_Pointer_Info(i,1), iBP_Pointer_Info(i,2), iBP_Pointer_Info(i,3),
iBP_Pointer_Info(i,4), iBP_Pointer_Info(i,5), iBP_Pointer_Info(i,6),
BP_xyz_Data(i,1), BP_xyz_Data(i,2), BP_xyz_Data(i,3)

Description:

xnb(i),ynb(i),znb(i): Components of the unitary vector normal to the boundary at point i.

xtb(i),ytb(i),ztb(i): Components of the unitary vector tangential to the boundary at that point.

xsb(i), ysb(i), zsb(i) Components of the unitary vector tangential to the boundary at point i and perpendicular to the previous one.

iBP_Pointer_Info(i,1): Absolute index BP

iBP_Pointer_Info(i,2): Rank of BP (default=0, reserved for MPI)

iBP_Pointer_Info(i,3): Absolute index of i-1 neighbour BP

iBP_Pointer_Info(i,4): Absolute index of i+1 neighbour BP

iBP_Pointer_Info(i,5): Absolute index of j-1 neighbour BP

iBP_Pointer_Info(i,6): Absolute index of j+1 neighbour BP

BP_xyz_Data(i,1), BP_xyz_Data(i,2), BP_xyz_Data(i,3): xp(BP), yp(BP), zp(BP)
needed for the future release of a MPI version of the code.

OBSTACLE

Created by *SPHysicsgen*

To be used by MATLAB codes for graphical representation

UNIT=55

The file contains the following variables:

iopt_obst
XXmin
XXmax
YYmin
YYmax
ZZinf
ZZmax
slope
iopt_obst
XXmin
XXmax
YYmin
YYmax
ZZinf
ZZmax
slope
……
iopt_obst

Description:

iopt_obst Conditional variable (1= obstacle exists; 0= it does not exist). The last one is always zero.
XXmin Minimum value of the obstacle in *x* direction.
XXmax Minimum value of the obstacle in *x* direction.
YYmin Minimum value of the obstacle in y direction.
YYmax Minimum value of the obstacle in y direction.
ZZmin Minimum value of the obstacle in z direction.
ZZmax Minimum value of the obstacle in z direction.
Slope Obstacle slope in *x* direction.

## WAVEMAKER

Created by *SPHysicsgen*

To be used by *SPHysics* code. Parameters fix the wavemaker extent and movement. It will only move in *x* direction.

UNIT=66

The file contains the following variables:

iopt_wavemaker
i_paddleType
nwavemaker_ini
nwavemaker_end
X_PaddleCentre
X_PaddleStart
paddle_SWL
flap_length

stroke
twavemaker
Nfreq
A_wavemaker(n)
Period(n)
phase(n)
twinitial(n)

Description:
iopt_wavemaker: Conditional variable (1= Wavemaker exists; 0= it does nor exist).
i_paddleType: Enter Paddle-Type (1: Piston, 2: Piston-flap)
nwavemaker_ini: First wavemaker particle.
nwavemaker_end: Last wavemaker particle.
X_PaddleCentre: Wavemaker Centre position in X coordinates
X_PaddleStart: X_PaddleStart = 0.5*stroke
paddle_SWL: Enter paddle Still Water Level (SWL)
flap_length: Enter piston-flap flap_length
stroke: Wavemaker Stroke = 2*Amplitude
twavemaker: Initial time of wavemaker
Nfreq: Number of frequencies
A_wavemaker:Amplitude of wavemaker movement.
Period: Period of wavemaker movement.
phase(n): Phase of wavemaker movement.
twinitial(n): Start of wavemaker movement (seconds).

GATE
Created by *SPHysicsgen*
To be used by *SPHysics* code. Parameters fix the gate extent and movement.
UNIT=77
The file contains the following variables:
iopt_gate
ngate_ini
ngate_end
VXgate,VYgate,VZgate
tgate

Description:
iopt_gate Conditional variable (1= gate exists; 0= it does nor exist).
ngate_ini First gate particle
ngate_end Last gate particle
VXgate,VYgate,VZgate Gate velocity in coordinates
tgate Start of gate movement (seconds).

3.2.1.4. Subroutines

All subroutines in SPHysicsgen are inside a single source file SPHysicsgen_2D.f or SPHysicsgen_3D.f

SPHysicsgen Main program.

Depending on the subroutine, different container geometries can be used.

BOX Subroutine to build a box in 2D or 3D.

BEACH Subroutine to build a beach in 2D or 3D. The beach consists in a flat area followed by a tilted region. The tilted area always has a slope in *x*- direction and a possible slope in *y*- direction.

Each subroutine calls new subroutines to generate the walls of the container and the different obstacles placed inside it.

BOUNDARIES_LEFT Subroutine to generate the left boundary of the container both in 2D and 3D.

BOUNDARIES_RIGHT Subroutine to generate the right boundary of the container both in 2D and 3D.

BOUNDARIES_BOTTOM Subroutine to generate the bottom boundary of the container both in 2D and 3D.

BOUNDARIES_FRONT Subroutine to generate the front of the container in 3D.

BOUNDARIES_BACK Subroutine to generate the back of the container in 3D.

WALL Subroutine to generate a wall with an arbitrary slope in *x*- direction inside the container.

WALL_HOLE Subroutine to generate a wall with a round shaped hole inside the container (Only in 3D version).

WALL_SLOT Subroutine to generate a wall with a slot inside the container (Only in 3D version).

OBSTACLE Subroutine to generate an obstacle inside the container.

WAVEMAKER Subroutine to generate a piston that can move in *x*- direction.

GATE Subroutine to generate gate that can move in any direction.

EXTERNAL_GEOMETRY This subroutine, which only works in 2D, reads the container and the initial fluid distribution from a file previously generated. The MATLAB software to generate the pre-processing will be provided in next release.

Apart from previous subroutines, which control the shape and dimensions of the container, other subroutines are responsible of the fluid properties inside that container.

FLUID_PARTICLES Subroutine to choose between different initial distributions of the fluid.

DROP Subroutine used to generate a round shaped area (2D or 3D) as initial position. The velocity of the particles inside the region can be fixed by the user (all particles share the same velocity).

SET Subroutine used to generate a set of particle as initial condition. The number of particle and the initial position and velocity of each particle can be decided by the user. This configuration is particularly useful when checking changes in the code since it permit runs with a small number of moving particles.

FILL PART Subroutine used to generate a cubic area as initial position (2D or 3D). Different cubes can placed at different position inside the computational domain.

WAVE Subroutine used to generate a wave (2D or 3D) advancing in *x*- direction as initial position.

POS_VELOC Subroutine used to determine the initial position and velocity of particles.

PRESSURE Subroutine used to determine the initial pressure of particles.

P_BOUNDARIES Subroutine to assign density equal to the reference density to the boundary particles and gage pressure equal to zero.

CORRECT_P_BOUNDARIES Subroutine to correct pressure at boundaries. It considers the density to be equal to the reference density plus a hydrostatic correction. Pressure is then calculated according to Batchelor equation (see Eq. 1.17).

PERIODICITYCHECK Subroutine to determine the limits in periodic boundary conditions. These BC are only available in 3D and in *y*- direction.

NORMALS_CALC_2D and NORMALS_CALC_3D Subroutines to calculate the normals to be used in repulsive boundary conditions.

NORMALS_FILEWRITE_2D and NORMALS_FILEWRITE_3D Subroutines to write the normals to be used in repulsive boundary conditions.

TOCOMPILE_IFORT Subroutine to create the MAKEFILE, SPHysics.mak, used to compile SPHysics using a IFORT compiler. The source files to be included in SPHysics.mak depend on the particular conditions of the run fixed by the input files.

TOCOMPILE_GFORTRAN Subroutine to create the MAKEFILE, SPHysics.mak, used to compile SPHysics using a GFORTRAN compiler. The source files to be included in the MAKEFILE depend on the particular conditions of the run fixed by the input files.

TOCOMPILE_CVF Subroutine to create the MAKEFILE necessary to compile SPHysics using a Compaq Visual Fortran compiler. The source files to be included in the MAKEFILE depend on the particular conditions of the run fixed by the input files.

### 3.2.2. SPHysics

SPHysics nature depends on the compiling option determined by *SPHysicsgen*

#### 3.2.2.1. Input files

The input files correspond to the output files generated by *SPHysicsgen* and described in section 3.2.1.3.

3.2.2.2. Output files

PART_klmn

Created by *SPHysics* at POUTE_3D.f or POUTE_2D.f with a periodicity in seconds fixed by the *input_file* used to run *SPHysicsgen*.

UNIT=23

The structure of PART_klmn is the same as that of IPART previously described. The indices *k*, *m*, *n* and *l* can take any integer value from 0 to 9, in such a way that the maximum number of images is 9999.

Each PART_klmn file is opened, recorded and closed in each call to POUTE_3D.f or POUTE_2D.f subroutines, so, a single UNIT=23 is assigned to all PART_klmn files.

SCAL

Created by *SPHysics* at POUTE_3D.f or POUTE_2D.f with the same periodicity as PART_klmn.

UNIT=22

The following variables are recorded:

itime time np nb nbf h

Description:

itime: Number of iterations since the beginning of the run.

time: Time instant (in seconds).

np: Total number of particles.

nb: Number of boundary particles.

nbf: Number of fixed boundary particles. Note that boundary particles can be fixed or move according to some external dependence (e.g. gates, wavemakers).

h: Smoothing length.

DT

Created by *SPHysics* at POUTE_3D.f or POUTE_2D.f

UNIT=19

The following variables are recorded:

time dt1 dt2 dtnew

Description:

time: Time instant (in seconds)

dt1: Time step based on the force per unit mass (see section 2.2).

dt2: Time step combining the Courant and the viscous conditions (see section 2.2).

dtnew: Time step corresponding to next step using *dt1* and *dt2*.

DETPART_klmn

Created by *SPHysics* at POUTE_3D.f or POUTE_2D.f

UNIT=53
The same as PART_klmn but with a shorter periodicity during a certain interval of the run. Details about periodicity, starting and end of this recording can be seen in section 4.

EPART
Created by *SPHysics* at POUTE_3D.f or POUTE_2D.f at the end of the run.
UNIT=33
This file contains the same information as IPART but corresponding to the end of the run.

ESCAL
Created by *SPHysics* at POUTE_3D.f or POUTE_2D.f at the end of the run
UNIT=32
It contains the same information as SCAL.

ENERGY
Created by *SPHysics* at ENERGY_2D.f or ENERGY_3D.f.
UNIT=50
The file contains the following variables recorded with the same periodicity as PART_kmnl.

time Eki_p Epo_p TE_p Eki_b Epo_b TE_b

Description:
time Time instant (in seconds)
Eki_p Kinetical energy summation (for fluid particles)
Epo_p Potential energy summation (for fluid particles)
TE_p Thermal energy summation (for fluid particles)
Eki_b Kinetical energy summation (for boundary particles)
Epo_b Potential energy summation (for boundary particles)
TE_b Thermal energy summation (for boundary particles)

NOTE: Boundary particle energies only make sense when using Dynamic Boundary Conditions.

3.2.2.3. Subroutines
All subroutines in SPHysicsgen are placed in the same source file, however SPHysics ones are placed in different source files. A short description of each possible subroutine follows.

SPHysics (Source file: SPHYSIC_2D.f or SPHYSIC_3D.f). Main program containing the main loop.

GETDATA (Source file: GETDATA_2D.f or GETDATA_3D.f). Subroutine called from SPHysics at the beginning of the run. It provides data about the run (scales, kernel parameters, steps, use of gates and/or wavemakers…).

ENERGY (Source file: ENERGY_2D.f or ENERGY_2D.f). Subroutine called from SPH to record information about energy (kinematical, potential and thermal). This subroutine is called at the beginning and end of the run and also every *out* seconds (variable provided by INDAT file). It creates the file ENERGY described in previous section.

INI_DIVIDE (Source file: INI_DIVIDE.f). Subroutine called from SPH at the beginning of the run (just for fixed boundary particles) and from subroutine STEP during the run (every time step for moving objects and fluid particles). It initializes the link list.

DIVIDE (Source file: DIVIDE_2D.f and DIVIDE_3D.f). Subroutine called from SPHysics at the beginning of the run and from subroutine STEP during the run (every time step). The first time (when called from SPHysics) creates the link list corresponding to the fixed boundary particles. The rest of the calls the subroutine allocates the fluid particles and the moving boundary particles into the link list.

KEEP_LIST (Source file: KEEP_LIST.f). Subroutine called from SPHysics at the beginning of the run just after calling DIVIDE. It keeps the list of fixed boundary particles, which is never recalculated again.

CHECK_LIMITS (Source files: CHECK_LIMITS_2D.f and CHECK_LIMITS_3D.f). Subroutine called from SPHysics every time step. The subroutines detect the position of particles outside the computational domain and relocate them (see section 2.5).

POUTE (Source files: POUTE_2D.f and POUTE_3D.f). Subroutine called from SPHysics to record information about particles (position, velocity, density, pressure and mass). This subroutine is called at the beginning and end of the run and also every *out* seconds. It creates the SCAL, PART, ESCAL and EPART files previously described.

SHEPARD (Source file: SHEPARD_2D.f and SHEPARD_3D.f). Subroutine called from SPH every *out* seconds. It uses a Shepard filter when selected in initial conditions.

STEP (Source files: STEP_PREDICTOR_CORRECTOR_2D.f, STEP_PREDICTOR_CORRECTOR_3D.f, STEP_VERLET_2D.f and STEP_VERLET_3D.f). Subroutine called from SPHysics. It basically manages the marching procedure, depending on the computational algorithm (Predictor-Corrector or Verlet).

CORRECT (Source files: CORRECT_2D.f, CORRECT_3D.f, CORRECT_SPS_2D.f and CORRECT_SPS_3D.f). This subroutine is called by STEP every time step. It

basically accounts for the body forces and XSPH correction (and SPS terms are calculated if *i_visos* = 3).

RECOVER_LIST (Source file: RECOVER_LIST.f). This subroutine is called from STEP every time step. It recovers the list corresponding to the fixed boundary particles created by KEEP_LIST.

VARIABLE_TIME_STEP (Source files: VARIABLE_TIME_STEP_2D.f and VARIABLE_TIME_STEP_3D.f). This subroutine is called from STEP every time step. It calculates the time step considering maximum inter-particle forces, the speed of sound and the viscosity.

AC (Source files: AC_2D.f and AC_3D.f). This subroutine is called from STEP every time step. It controls the boundary particles movement (gates and wavemakers) and calls the subroutines SELF and CELIJ.

SELF (Source files: SELF_BC_DALRYMPLE_2D.f, SELF_BC_DALRYMPLE_3D.f, SELF_BC_MONAGHAN_2D.f, SELF_BC_MONAGHAN_3D.f). This subroutine is called from AC every time step. It controls the interaction between particles inside the same *"cell"* determined by the link list.

CELIJ (Source files: CELIJ_BC_DALRYMPLE_2D.f, CELIJ_BC_DALRYMPLE_3D.f, CELIJ_BC_MONAGHAN_2D.f, CELIJ_BC_MONAGHAN_3D.f). This subroutine is called from AC every time step. It controls the interaction between particles inside adjacent *"cells"* determined by the link list.

KERNEL (Source files: KERNEL_GAUSSIAN_2D.f, KERNEL_GAUSSIAN_2D.f, KERNEL_QUADRATIC_2D.f, KERNEL_QUADRATIC_3D.f, KERNEL_CUBIC_2D.f, KERNEL_CUBIC_3D.f, KERNEL_WENDLAND5_2D.f, KERNEL_ WENDLAND5_3D.f). This subroutine is called from SELF and CELIJ every time step. It calculates the particle-particle interaction according to kernel definition (1=gaussian, 2=quadratic; 3=cubic; 5=wendland) and dimensionality of the problem (2D or 3D).

VISCOSITY (Source files: VISCOSITY_ARTIFICIAL_2D.f, VISCOSITY_ARTIFICIAL_3D.f, VISCOSITY_LAMINAR_2D.f, VISCOSITY_LAMINAR_3D.f, VISCOSITY_LAMINAR+SPS_2D.f and VISCOSITY_LAMINAR+SPS_3D.for). This subroutine is called from SELF and CELIJ every time step. It calculates viscosity terms depending on the chosen option ((1) Artificial (2) Laminar (3) Laminar +SPS) and dimensionality of the problem (2D or 3D).

MONAGHANBC (Source file: MONAGHANBC_2D.f and MONAGHANBC_3D.f). This subroutine is called from CELIJ and SELF (only when considering SELF_BC_MONAGHAN_3D.f and CELIJ_BC_MONAGHAN_3D.f sources). It accounts for Monaghan's repulsive force between fluid and boundary particles.

MOVINGOBJECTS (Source file: MOVINGOBJECTS_2D.f and MOVINGOBJECTS _3D.f). This subroutine is called from STEP.

# 4. TEST CASES

## 4.1. Running the model

Creating and running executable files can be done step by step by the user (compiling the different source files, putting them in a certain directory and executing the codes while typing the values of the different variables and options when prompted). Nevertheless, this process can become tedious, especially when running different realizations of the same case with small differences in a small number of parameters. The entire process can be automatically done, although with some differences on different computer systems. Here we will show two examples for WINDOWS and LINUX.

### 4.1.1. Compiling and executing on Linux

SPHysics also currently supports following fortran compilers that have been tested on Linux platforms,

1. gfortran, a free Fortran 95/2003 compiler that can be downloaded from http://gcc.gnu.org/wiki/GFortran.
2. The non-commercial Intel ® Fortran Compiler 10.0 that can be downloaded from http://www.intel.com/cd/software/products/asmo-na/eng/compilers/282048.htm. In this document, this compiler will be referred to as ifort.

In order to run SPHysics on Linux, gfortran, ifort and the GNU make utility need to be installed and available in the default search path (typically /usr/bin or /usr/local/bin). The f0llowing paragraphs explain the procedure to compile and run the 2D version of SPHysics. The procedure is exactly the same for the 3D version.

Compiling SPHysicsgen_2D

In the SPHysics_2D/source/SPHysicsgen2D directory there are two Makefiles named SPHysicsgen_gfortran.mak and SPHysicsgen_ifort.mak. As their names suggest, they are used to compile SPHysicsgen_2D using the gfortran and ifort compilers respectively. The gfortran Makefile can be executed using the command 'make -f Makefile_gfortran.mak'. The Makefile,

1. compiles SPHysicsgen_2D
2. checks for existence of SPHysics_2D/execs and SPHysics_2D/execs.bak directories. If non-existent these directories are created.
3. moves the previous version of the SPHysicsgen_2D executable, if available, from the execs directory to execs.bak directory
4. moves the latest compiled version of SPHysicsgen_2D to the execs directory.

Running SPHysicsgen_2D and SPHysics_2D

As mentioned before, SPHysicsgen_2D, based on the options chose by the user, generates the Makefile, SPHysics.mak, to compile the main program SPHysics. The subroutines tocompile_gfortran and tocompile_ifort, in SPHysicsgen_2D, write out SPHysics.mak for gfortran and ifort compilers respectively.

There are linux batch files located in the four 2D example directories, run_directory/CaseN, where N=1,2,3,4. These batch files are named CaseN_linux.bat (N=1,2,3,4) . Similar linux batch files are located in the 3D example directories.

The following table gives a detailed description of the commands used in the script file *Case1_linux.bat* which is located in SPHysics_2D/run_directory/Case1. This batch file can be executed, while in the Case1 directory, by typing ./Case1_linux.bat at the command prompt.

| **COMMAND** | **COMMENTS** |
|---|---|
| ../../execs/SPHysicsgen_2D < Case1.txt > Case1.out | Run SPHysicsgen_2D with Case1.txt as the input file instead of command line input. The output from the execution is redirected in Case1.out |
| cp SPHysics.mak ../../source/SPHysics2D | Copy the generated Makefile to the SPHysics2D source directory. |
| cd ../../source/SPHysics2D | Change to source directory in order to compile SPHysics using SPHysics.mak |
| make -f SPHysics.mak clean | Remove any prexisting object files |
| make -f SPHysics.mak | Compile and generate SPHysics_2D using SPHysics.make. Similar to the Makefiles for SPHysicsgen_2D, this Makefile compiles and places the SPHysics_2D executable in the execs directory and moves the older executable to the execs.bak directory |
| rm SPHysics.mak | Remove the Makefile from the source/SPHysics2D directory. |
| cd ../../run_directory/Case1 | Change to the Case1 example directory. |
| ../../execs/SPHysics_2D > sph.out | Execute SPHysics_2D and redirect the output from the run to sph.out |
| | |

4.1.2. Compiling and executing on Windows.

In the SPHysics_2D/source/SPHysicsgen2D directory there is a Makefile named SPHysicsgen_cvf.mak. It is used to compile SPHysicsgen_2D using the CVF compiler. The cvf Makefile can be executed using the command NMAKE /f "SPHysicsgen_cvf.mak"

As mentioned before, SPHysicsgen_2D, based on the options chose by the user, generates the Makefile, SPHysics.mak, to compile the main program SPHysics. The subroutine tocompile_windows, in SPHysicsgen_2D, write out SPHysics.mak for cvf compiler

There are windows batch files located in the example directories, The batch file *Case1_cvf.bat* located in .*SPHysics**SPHysics_2D**run_directory**Case1* (see Fig. 3.1) is used. Similar batch files correspond to other 2D examples. Examples corresponding to 3D calculations can be found in .*SPHysics**SPHysics_3D**run_directory**Case1*
The user should, while in the Case1 directory, write *Case1_cvf.bat* on a command window. The content of this file is briefly describe in next table

| **COMMAND** | **COMMENTS** |
|---|---|
| del *.exe | Remove previous executable files. |
| copy..\..\execs\SPHysicsgen_2D.exe SPHysicsgen_2D.exe | Copy SPHysicsgen_2D.exe file to the working directory. |
| SPHysicsgen_2D.exe <Case1.txt > Case1.out | Run SPHysicsgen_2D.exe.<br>This program creates the initial conditions and select the options of the run. In addition, it also creates a file SPHysics.mak that can be used to compile the SPHysics_2D code with the right options.<br>Any name can be used for the input and output files |
| copy SPHysics.mak ..\..\source\SPHysics_2D\SPHysics.mak | Copy the SPHysics.mak file to the place where the SPHysics_2D source files are located. |
| cd ..\..\execs\ | Change to the directory where the executable file will be created. |
| del *.obj | Remove previous object files |
| del SPHysics_2D.exe | Remove previous executable versions of SPHysics_2D.exe |

| | |
|---|---|
| cd..\source\SPHysics_2D | Change to the directory containing the SPHysics_2D source files |
| NMAKE /f "SPHysics.mak" | *NMAKE /f "SPHysics.mak"* is used to compile SPHysics_2D.exe. There are multiple options to compile SPHysics_2D.exe. They are automatically selected depending on the initial conditions provided by the input file (Case1.txt in this example). The file *SPHysics.mak*, which is automatically created by SPHysicsgen_2D.exe, contains information about those options. |
| cd ..\..\ run_directory\Case1 | Change directory |
| del PART* | Remove previous PART* files |
| del DETPART* | Remove previous DETPART* files |
| copy ..\..\execs\SPHysics_2D.exe SPHysics_2D.exe | Copy SPHysics_2D.exe file to the working directory |
| SPHysics_2D.exe >sph.out | Run the case.<br>Any name can be used for the output file sph.out |

### 4.2. Test case 1: 2D Dam break in a box

The case can be run using *Case1.bat* (*Case1_cvf.bat* or *Case1_linux.bat*) whose output directory is *Case1*. The input file *Case1.txt* is located in the output directory. The information contained in that file can be summarized as follows:

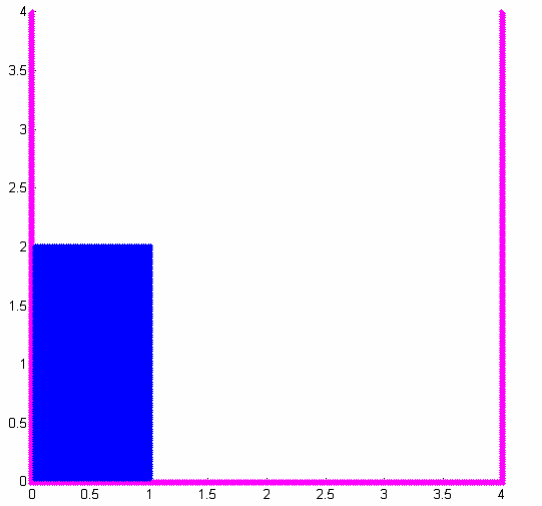


**Figure 4.1:** Initial configuration of Case1.

| Variable Name | Value | Comments |
|---|---|---|
| i_kernel | 5 | Kind of kernel<br>5=Wendland kernel |
| i_algorithm | 1 | Kind of algorithm<br>1=predictor corrector |
| i_shepard | 0 | Possibility to apply a Shepard filter (1=yes) |
| i_viscos | 1 | Viscosity:<br>(1) Artificial<br>(2) Laminar<br>(3) Laminar +SPS |
| Viscos_val | 0.1 | If *i_viscos* = 1 it corresponds to α (Monaghan and Koss, 1999)<br>If *i_viscos* = 2 or 3 it corresponds to ν (kinematical viscosity on the order of $10^{-6}$). |
| h_SWL | 2.0 | Still water level (m).<br>Used to calculate B and the speed of sound[1]. |
| coef | 25 | Coefficient to calculate B.<br>Recommended values( 10 , 40 )[1] |
| IBC | 2 | Boundary Conditions<br>1= Monaghan & Kos (1999) |

| | | |
|---|---|---|
| | | 2= Dalrymple & Knio (2000) |
| i_geometry | 1 | Geometry of the medium<br>1= BOX whose bottom can be tilted in X and Y<br>2= Beach |
| lattice | 2 | Lattice: (1) SC; (2) BCC |
| vlx,vlz | 4.0,4.0 | Dimension of the medium |
| dx,dz | 0.03,0.03 | Initial interparticle spacing |
| beta_deg | 0 | inclination of floor (X direction) |
| iopt_addwall | 0 | Add wall (1=y) |
| iopt_obst | 0 | Add obstacle (1=y) |
| iopt_wavemaker | 0 | Add wavemaker (1=y) |
| iopt_gate | 0 | Add gate (1=y) |
| initial | 2 | Initial configuration of the fluid<br>1= Set of particles without grid<br>2= Particles. on a grid filling the box till a certain height<br>3= Particles on a grid filling a part of the box till a certain height<br>4=Round shaped drop |
| i_correct_pb | 0 | Initial pressure is corrected at fluid particles (hydrostatic). If *i_correct_pb* =1 its also done for boundary particles. |
| XXmin,XXmax | 0.03,1.0 | Extent of fluid in X |
| ZZmin,ZZmax | 0.03,2.0 | Extent of fluid in Z |
| iopt_fill_part | 0 | If *iopt_fill_part* =1 a new region will be filled<br>The previous parameters (since i_correct_pb) will be asked. |
| tmax,out | 3,0.02 | tmax: Run duration (seconds)<br>out: Recording time step (seconds) |
| trec_ini | 0.0 | trec_ini: initial time of outputting general data (seconds) |
| dtrec_det,t_sta_det,t_end_det | 0,1,-1 | Detailed recording<br>step, Start time, End Time<br>A wrong sequence where t_sta_det > t_end_det disables this option |
| dt,ivar_dt | 0.0001, 1 | dt: Initial time step (seconds) |

| | | |
|---|---|---|
| | | ivar_dt: Controls variable time step (If *ivar_dt* = 1 ⇒ variable time step)[2] |
| coefficient | 0.92 | Coeffficient to calculate the smoothing length (h) in terms of dx,dz, h=coefficient*sqrt(dx*dx+dz*dz) |
| i_compile_opt | 3 | Compiler : 1=gfortran, 2=ifort, 3=HP 6.5 |

[1]Following (Monaghan and Koss, 1999), the speed of sound is imposed depending on the fastest waves in the medium which themselves depend on h_SWL

[2] The variable time step is calculated following Monaghan (1989) and Monaghan & Koss (1999).

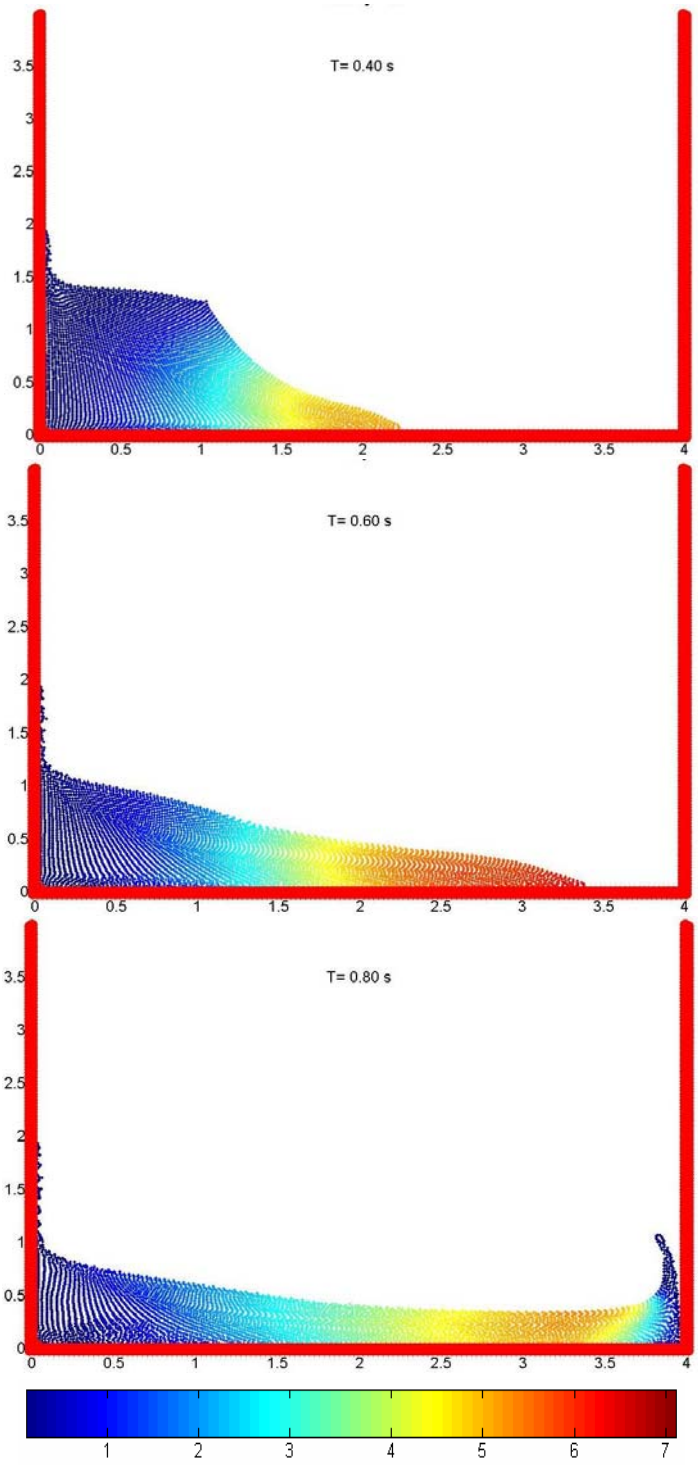


**Figure 4.2:** X-Velocity plot in Case1.

### 4.3. Test case 2: 2D Dam break evolution over a wet bottom in a box.

The case can be run using *Case2.bat* whose output directory is *Case2*. The input file *Case2.txt* is located in the output directory. The information contained in that file can be summarized as follows:

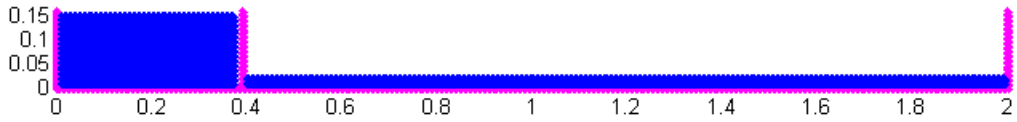


**Figure 4.3:** Initial configuration of Case2

| Variable Name | Value | Comments |
|---|---|---|
| i_kernel | 3 | Kind of kernel<br>3= Cubic Spline kernel |
| i_algorithm | 2 | Kind of algorithm<br>2=verlet |
| i_shepard | 0 | Possibility to apply a Shepard filter (1=yes) |
| i_viscos | 1 | Viscosity:<br>(1) Artificial<br>(2) Laminar<br>(3) Laminar +SPS |
| Viscos_val | 0.08 | If *i_viscos* = 1 it corresponds to α (Monaghan and Koss, 1999)<br>*If i_viscos* = 2 or 3 it corresponds to ν (kinematical viscosity on the order of $10^{-6}$). |
| h_SWL | 0.15 | Still water level (m).<br>Used to calculate B and the speed of sound[1]. |
| coef | 25 | Coefficient to calculate B.<br>Recommended values( 10 , 40 )[1] |
| IBC | 2 | Boundary Conditions<br>1= Monaghan & Kos (1999)<br>2= Dalrymple & Knio (2000) |

| | | |
|---|---|---|
| i_geometry | 1 | Geometry of the medium<br>1= BOX whose bottom can be tilted in X and Y<br>2= Beach |
| lattice | 2 | Lattice: (1) SC; (2) BCC |
| vlx,vlz | 2,0.16 | Dimension of the medium |
| dx,dz | 0.005,0.005 | Initial interparticle spacing |
| beta_deg | 0 | inclination of floor (X direction) |
| iopt_addwall | 0 | Add wall (1=y) |
| iopt_obst | 0 | Add obstacle (1=y) |
| iopt_wavemaker | 0 | Add wavemaker (1=y) |
| iopt_gate | 1 | Add gate (1=y) |
| Xwall | 0.38 | Gate position in X |
| ZZmin,ZZmax | 0.,0.16 | Gate height ?? |
| VXgate,VZgate | 0.,1.5 | GATE Velocity ?? |
| tgate | 0 | Initial time to start gate movement |
| iopt_gate | 0 | Add new gate (1=y) |
| initial | 2 | Initial configuration of the fluid<br>1= Set of particles without grid<br>2= Particles. on a grid filling the box till a certain height<br>3= Particles on a grid filling a part of the box till a certain height<br>4=Round shaped drop |
| i_correct_pb | 0 | Initial pressure is corrected at fluid particles (hydrostatic). If *i_correct_pb* =1 its also done for boundary particles. |
| XXmin,XXmax | 0.01,0.35 | Extent of fluid in X |
| ZZmin,ZZmax | 0.005,0.15 | Extent of fluid in Z |
| iopt_fill_part | 1 | If *iopt_fill_part* =1 a new region will be filled<br>The previous parameters (since i_correct_pb) will be asked. |
| XXmin,XXmax | 0.4,1.995 | Extent of fluid in X |
| ZZmin,ZZmax | 0.005,0.018 | Extent of fluid in Z |

| | | |
|---|---|---|
| iopt_fill_part | 0 | |
| tmax,out | 1.2,0.01 | tmax: Run duration (seconds)<br>out: Recording time step (seconds) |
| trec_ini | 0.0 | trec_ini: initial time of outputting general data (seconds) |
| dtrec_det,t_sta_det,t_end_det | 0,1,-1 | Detailed recording<br>step, Start time, End Time<br>A wrong sequence where<br>t_sta_det > t_end_det disables this option |
| dt,ivar_dt | 0.0001, 1 | dt: Initial time step (seconds)<br>ivar_dt: Controls variable time step (If *ivar_dt* = 1 $\Rightarrow$ variable time step)[2] |
| coefficient | 0.92 | Coeffficient to calculate the smoothing length (h) in terms of dx,dz,<br>h=coefficient*sqrt(dx*dx+dz*dz) |
| i_compile_opt | 3 | Compiler : 1=gfortran, 2=ifort, 3=HP 6.5 |

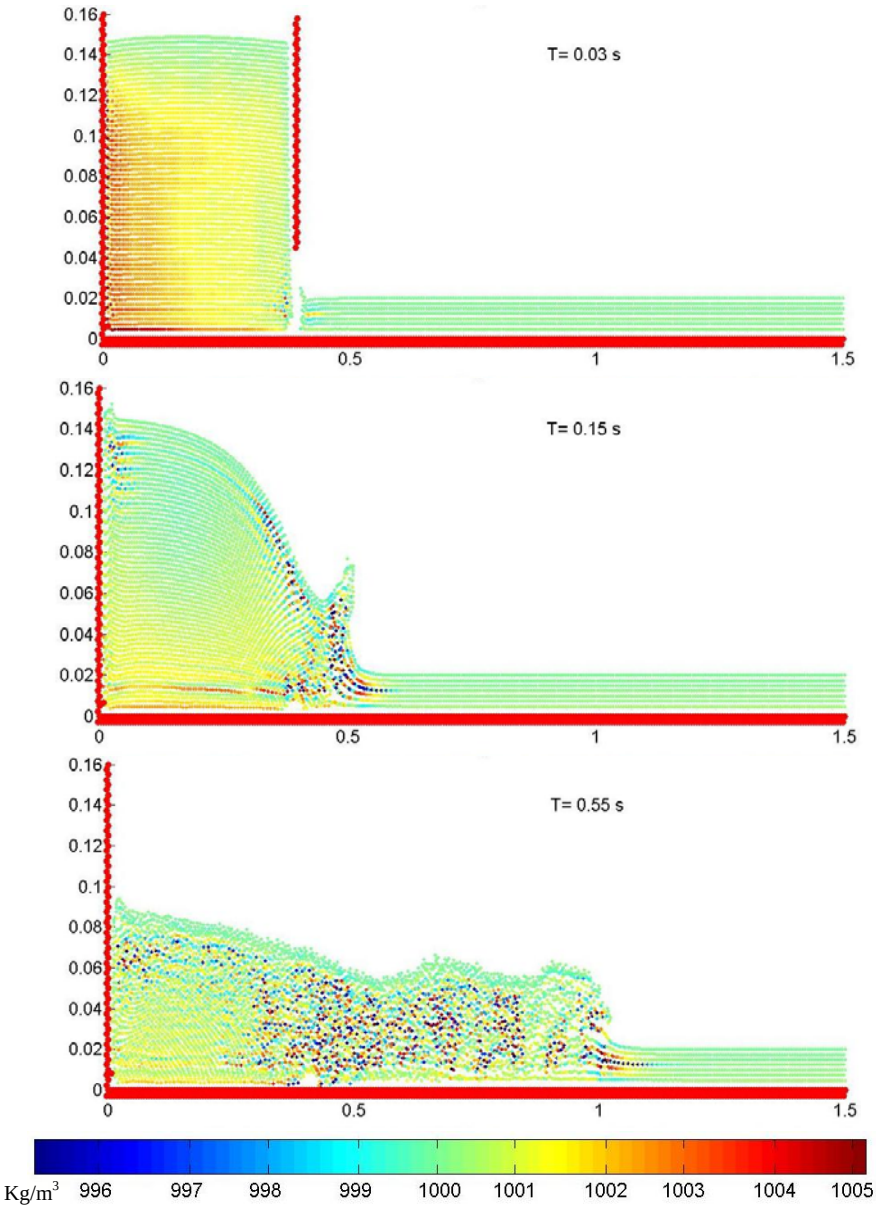


**Figure 4.4:** Density plot in Case2

### 4.4. Test case 3: Waves generated by a paddle in a beach

The case can be run using *Case3.bat* whose output directory is *Case3*. The input file *Case3.txt* is located in the output directory. The information contained in that file can be summarized as follows:

4.4.1 Case 2D

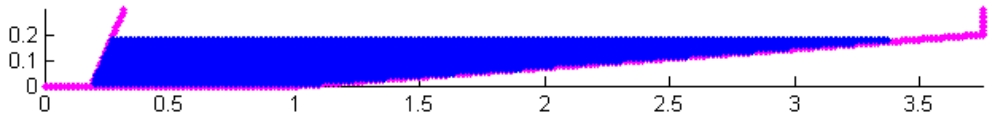


**Figure 4.5:** Initial configuration of Case3 in 2D.

| Variable Name | Value | Comments |
|---|---|---|
| i_kernel | 2 | Kind of kernel<br>2= Quadratic |
| i_algorithm | 1 | Kind of algorithm<br>1=Predictor-Corrector |
| i_shepard | 1 | Possibility to apply a Shepard filter (1=yes) |
| i_viscos | 3 | Viscosity:<br>(1) Artificial<br>(2) Laminar<br>(3) Laminar +SPS |
| Viscos_val | 1.0e-6 | If *i_viscos* = 1 it corresponds to α (Monaghan and Koss, 1999)<br>*If i_viscos* = 2 or 3 it corresponds to ν (kinematical viscosity on the order of $10^{-6}$). |
| h_SWL | 0.2 | Still water level (m).<br>Used to calculate B and the speed of sound[1]. |
| coef | 16 | Coefficient to calculate B.<br>Recommended values( 10 , 40 )[1] |
| IBC | 1 | Boundary Conditions<br>1= Monaghan & Kos (1999)<br>2= Dalrymple & Knio (2000) |
| visc_wall | 8.0e-3 | Wall viscosity value for Repulsive Force BC |

| | | |
|---|---|---|
| i_geometry | 2 | Geometry of the medium<br>1= BOX whose bottom can be tilted in X and Y<br>2= Beach |
| lattice | 1 | Lattice: (1) SC; (2) BCC |
| vlx,vly,vlz | 3.75,0.30 | Dimension of the medium |
| dx,dy,dz | 0.01,0.01 | Initial interparticle spacing |
| vlx1 | 1.0 | Length of Flat Domain |
| beta_deg | 4.2364 | Slope (deg) of the inclined plane ( beta ) ?? |
| iopt_wavemaker | 1 | If wavemaker will be added, left wall is not needed (1=yes) |
| iopt_obstacle | 0 | Add obstacle (1=yes) |
| iopt_wavemaker | 1 | Add wavemaker (1=yes) |
| i_paddleType | 2 | Enter Paddle-Type (1: Piston, 2: Piston-flap) |
| X_PaddleCentre | 0.13 | Wavemaker Centre position in X coordinates |
| paddle_SWL | 0.15 | Enter paddle Still Water Level (SWL) |
| flap_length | 0.1344 | Enter piston-flap flap_length |
| ZZMin, ZZmax | 0.0,0.3 | Start and end of the wavemaker in Z |
| twavemaker | 0.0 | Initial time of wavemaker |
| Nfreq | 1 | Number of frequencies |
| A_wavemaker | 0.2442 | Wavemaker Stroke = 2*Amplitude |
| Period | 1.4 | Period |
| Phase | 0 | Phase |
| twinitial | 0 | twinitial |
| iopt_wavemaker | 0 | Add another wavemaker inside the beach (1=yes) |
| iopt_gate | 0 | Add gate (1=y) |
| i_water | 1 | Add water in the flat region ?? (1=yes) |
| XXmin,XXmax | 0, 1.0 | Extent of fluid in X |
| ZZmin,ZZmax | 0.025, 0.18 | Extent of fluid in Z |
| i_water2 | 1 | Add water in the inclined region ?? (1=yes) |
| XXmin,XXmax | 1.0, 3.75 | Extent of fluid in X |
| ZZmin,ZZmax | 0.025, 0.18 | Extent of fluid in Z |
| i_wave | 0 | Add a solitary wave ?? (1=yes) |

| | | |
|---|---|---|
| tmax,out | 5.0,0.05 | tmax: Run duration (seconds)<br>out: Recording time step (seconds) |
| trec_ini | 0.0 | trec_ini: initial time of outputting general data (seconds) |
| dtrec_det,t_sta_det,t_end_det | 0,1,-1 | Detailed recording<br>step, Start time, End Time<br>A wrong sequence where<br>t_sta_det > t_end_det disables this option |
| dt,ivar_dt | 0.000045,0 | dt: Initial time step (seconds)<br>ivar_dt: Controls variable time step (If *ivar_dt* = 1 $\Rightarrow$ variable time step)[2] |
| coefficient | 0.92 | Coeffficient to calculate the smoothing length (h) in terms of dx,dy,dz<br>h=coefficient*sqrt(dx*dx+dy*dy+dz*dz) |
| i_compile_opt | 3 | Compiler : 1=gfortran, 2=ifort, 3=HP 6.5 |

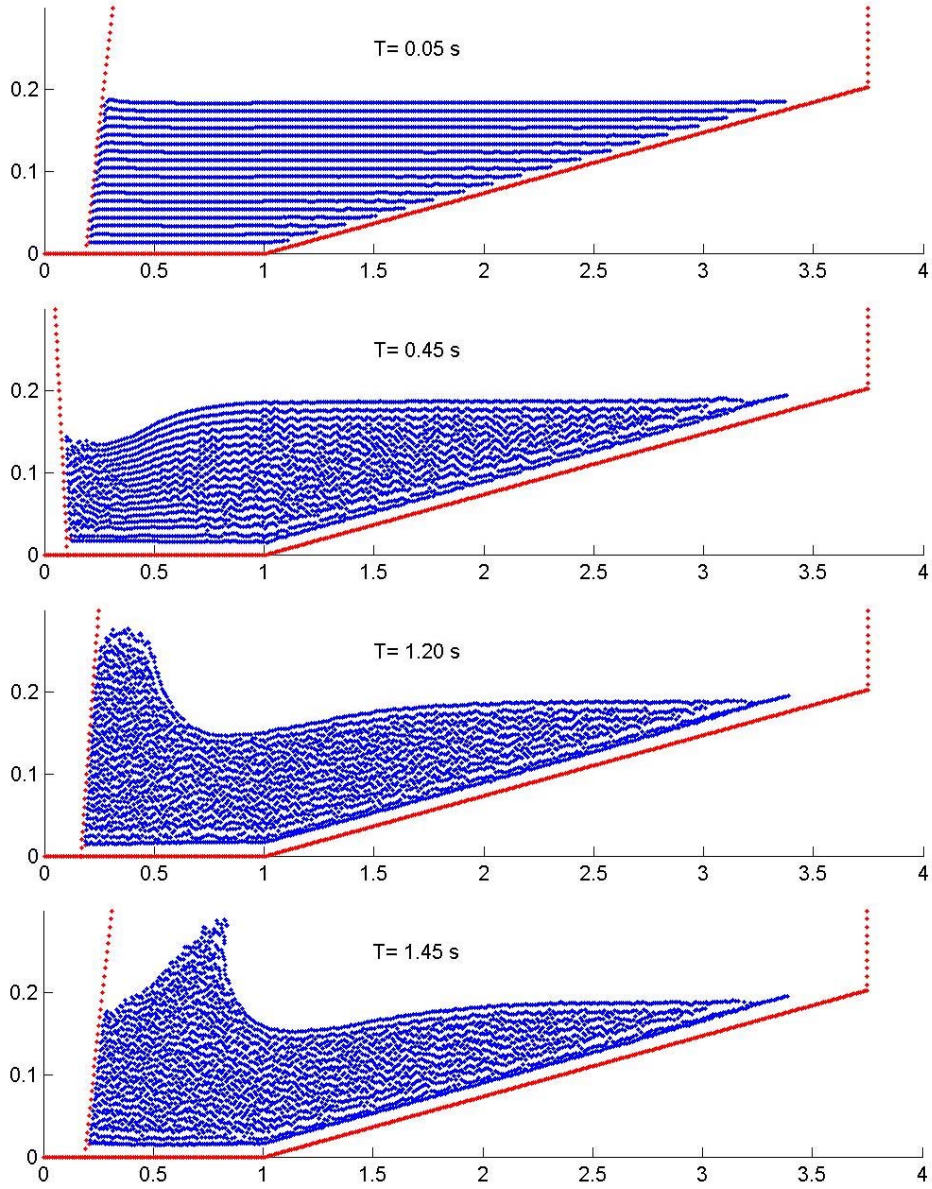


**Figure 4.6:** Wave formation in Case3 (for 2D).

4.4.2 Case 3D

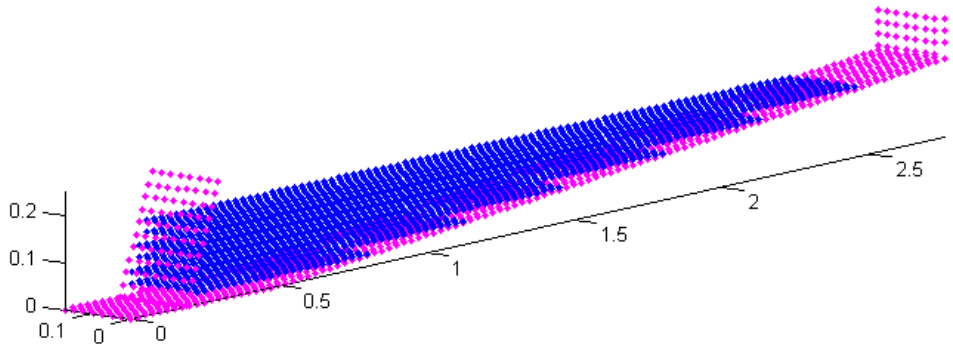


**Figure 4.7:** Initial configuration of Case3 in 3D.

| Variable Name | Value | Comments |
|---|---|---|
| i_kernel | 2 | Kind of kernel<br>2= Quadratic |
| i_algorithm | 1 | Kind of algorithm<br>1=Predictor-Corrector |
| i_shepard | 1 | Possibility to apply a Shepard filter (1=yes) |
| i_viscos | 3 | Viscosity:<br>(1) Artificial<br>(2) Laminar<br>(3) Laminar +SPS |
| Viscos_val | 1.0e-6 | If *i_viscos* = 1 it corresponds to α (Monaghan and Koss, 1999)<br>*If i_viscos* = 2 or 3 it corresponds to ν (kinematical viscosity on the order of $10^{-6}$). |
| h_SWL | 0.15 | Still water level (m).<br>Used to calculate B and the speed of sound[1]. |
| coef | 16 | Coefficient to calculate B.<br>Recommended values( 10 , 40 )[1] |
| IBC | 1 | Boundary Conditions<br>1= Monaghan & Kos (1999)<br>2= Dalrymple & Knio (2000) |
| visc_wall | 8.0e-1 | Wall viscosity value for Repulsive Force BC |

| | | |
|---|---|---|
| i_geometry | 2 | Geometry of the medium<br>1= BOX whose bottom can be tilted in X and Y<br>2= Beach |
| lattice | 2 | Lattice: (1) SC; (2) BCC |
| vlx,vly,vlz | 2.75,0.20,0.25 | Dimension of the medium |
| dx,dy,dz | 0.02,0.02,0.02 | Initial interparticle spacing |
| vlx1 | 0.5 | Length of Flat Domain |
| beta_deg | 4.2364 | Slope (deg) of the inclined plane ( beta ) ?? |
| i_periodicOBs(1),i_periodicOBs(2),i_periodicOBs(3) | 0,1,0 | Periodic Lateral boundaries in X, Y, & Z-Directions ? (1=yes) |
| iopt_wavemaker | 1 | If wavemaker will be added, left wall is not needed (1=yes) |
| iopt_obstacle | 0 | Add obstacle (1=yes) |
| iopt_wavemaker | 1 | Add wavemaker (1=yes) |
| i_paddleType | 2 | Enter Paddle-Type (1: Piston, 2: Piston-flap) |
| X_PaddleCentre | 0.13 | Wavemaker Centre position in X coordinates |
| paddle_SWL | 0.15 | Enter paddle Still Water Level (SWL) |
| flap_length | 0.1344 | Enter piston-flap flap_length |
| YYMin, YYmax | 0.0,0.2 | Start and end of the wavemaker in Y |
| ZZMin, ZZmax | 0.0,0.25 | Start and end of the wavemaker in Z |
| twavemaker | 0.0 | Initial time of wavemaker |
| Nfreq | 1 | Number of frequencies |
| A_wavemaker | 0.2442 | Wavemaker Stroke = 2*Amplitude |
| Period | 1.4 | Period |
| Phase | 0 | Phase |
| twinitial | 0 | twinitial |
| iopt_wavemaker | 0 | Add another wavemaker inside the beach (1=yes) |
| iopt_gate | 0 | Add gate (1=y) |
| i_water | 1 | Add water in the flat region ?? (1=yes) |
| XXmin,XXmax | 0, 0.515 | Extent of fluid in X |
| YYmin,YYmax | 0.025, 0.1975 | Extent of fluid in Y |
| ZZmin,ZZmax | 0.025, 0.15 | Extent of fluid in Z |
| i_water2 | 1 | Add water in the inclined region ?? |

| | | |
|---|---|---|
| | | (1=yes) |
| XXmin,XXmax | 0.52, 2.5 | Extent of fluid in X |
| YYmin,YYmax | 0.025, 0.1975 | Extent of fluid in Y |
| ZZmin,ZZmax | 0.025, 0.15 | Extent of fluid in Z |
| i_wave | 0 | Add a solitary wave ?? (1=yes) |
| tmax,out | 5.0,0.05 | tmax: Run duration (seconds)<br>out: Recording time step (seconds) |
| trec_ini | 0.0 | trec_ini: initial time of outputting general data (seconds) |
| dtrec_det,t_sta_det,t_end_det | 0,1,-1 | Detailed recording<br>step, Start time, End Time<br>A wrong sequence where<br>t_sta_det > t_end_det disables this option |
| dt,ivar_dt | 0.00005, 1 | dt: Initial time step (seconds)<br>ivar_dt: Controls variable time step (If *ivar_dt* = 1 $\Rightarrow$ variable time step)[2] |
| coefficient | 0.866025 | Coeffficient to calculate the smoothing length (h) in terms of dx,dy,dz<br>h=coefficient*sqrt(dx*dx+dy*dy+dz*dz) |
| i_compile_opt | 3 | Compiler : 1=gfortran, 2=ifort, 3=HP 6.5 |

**4.5. Test case 4:** Tsunami generated by a sliding Wedge

The case can be run using *Case4.bat* whose output directory is *Case4*. The input file *Case4.txt* is located in the output directory. The information contained in that file can be summarized as follows:

4.5.1 Case 2D

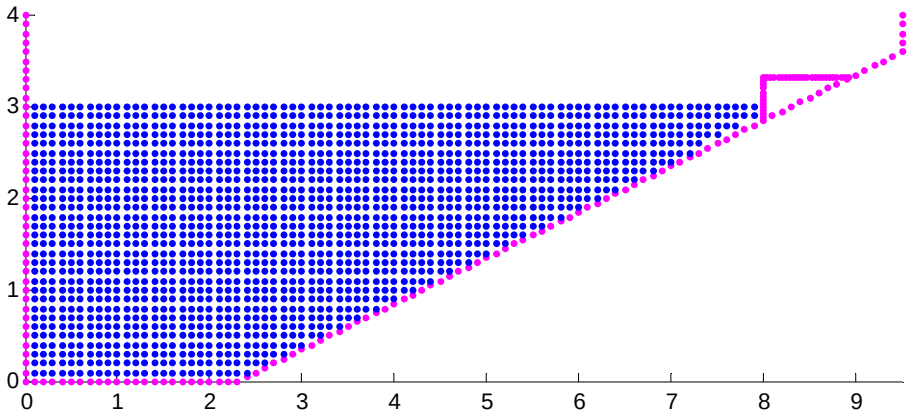


**Figure 4.8:** Initial configuration of Case4 in 2D.

| Variable Name | Value | Comments |
|---|---|---|
| i_kernel | 2 | Kind of kernel<br>2= Quadratic |
| i_algorithm | 1 | Kind of algorithm<br>1=Predictor-Corrector |
| i_shepard | 1 | Possibility to apply a Shepard filter (1=yes) |
| i_viscos | 3 | Viscosity:<br>(1) Artificial<br>(2) Laminar<br>(3) Laminar +SPS |
| Viscos_val | 1.0e-6 | If *i_viscos* = 1 it corresponds to α (Monaghan and Koss, 1999)<br>*If i_viscos* = 2 or 3 it corresponds to ν (kinematical viscosity on the order of $10^{-6}$). |
| h_SWL | 3.0 | Still water level (m).<br>Used to calculate B and the speed of sound[1]. |
| coef | 16 | Coefficient to calculate B.<br>Recommended values( 10 , 40 )[1] |

| | | |
|---|---|---|
| IBC | 1 | Boundary Conditions<br>1= Monaghan & Kos (1999)<br>2= Dalrymple & Knio (2000) |
| visc_wall | 2.0e-4 | Wall viscosity value for Repulsive Force BC |
| i_geometry | 2 | Geometry of the medium<br>1= BOX whose bottom can be tilted in X and Y<br>2= Beach |
| lattice | 1 | Lattice: (1) SC; (2) BCC |
| vlx,vlz | 9.5,4.0 | Dimension of the medium |
| dx,dz | 0.1,0.1 | Initial interparticle spacing |
| vlx1 | 2.25 | Length of Flat Domain |
| beta_deg | 26.565051 | Slope (deg) of the inclined plane ( beta ) ?? |
| iopt_wavemaker | 0 | If wavemaker will be added, left wall is not needed (1=yes) |
| iopt_obstacle | 0 | Add obstacle (1=yes) |
| iopt_wavemaker | 0 | Add wavemaker (1=yes) |
| iopt_RaichlenWedge | 1 | Add Raichlen wedge (1=yes) |
| bigDelta | 0.3 | Enter block-top elevation above SWL |
| specificWeight | 2.14 | Enter specific Weight |
| block_length, block_height, block_width | 0.91, 0.455, 0.61 | Enter block_length, block_height, block_width |
| dxW, dzW | 0.04, 0.04 | Enter block discretization size |
| i_water | 1 | Add water in the flat region ?? (1=yes) |
| XXmin,XXmax | 0.05, 2.20 | Extent of fluid in X |
| ZZmin,ZZmax | 0.05, 3.5 | Extent of fluid in Z |
| i_water2 | 1 | Add water in the inclined region ?? (1=yes) |
| XXmin,XXmax | 2.30, 9.5 | Extent of fluid in X |
| ZZmin,ZZmax | 0.05, 3.5 | Extent of fluid in Z |
| i_wave | 0 | Add a solitary wave ?? (1=yes) |
| tmax,out | 3.0,0.05 | tmax: Run duration (seconds)<br>out: Recording time step (seconds) |
| trec_ini | 0.0 | trec_ini: initial time of outputting general data (seconds) |
| dtrec_det,t_sta_det,t_end_det | 0,1,-1 | Detailed recording |

| | | |
|---|---|---|
| | | step, Start time, End Time<br>A wrong sequence where<br>t_sta_det > t_end_det disables this option |
| dt,ivar_dt | 0.00011, 0 | dt: Initial time step (seconds)<br>ivar_dt: Controls variable time step (If *ivar_dt* = 1 $\Rightarrow$ variable time step)[2] |
| coefficient | 0.92 | Coeffficient to calculate the smoothing length (h) in terms of dx,dz<br>h=coefficient*sqrt(dx*dx+dz*dz) |
| i_compile_opt | 3 | Compiler : 1=gfortran, 2=ifort, 3=HP 6.5 |

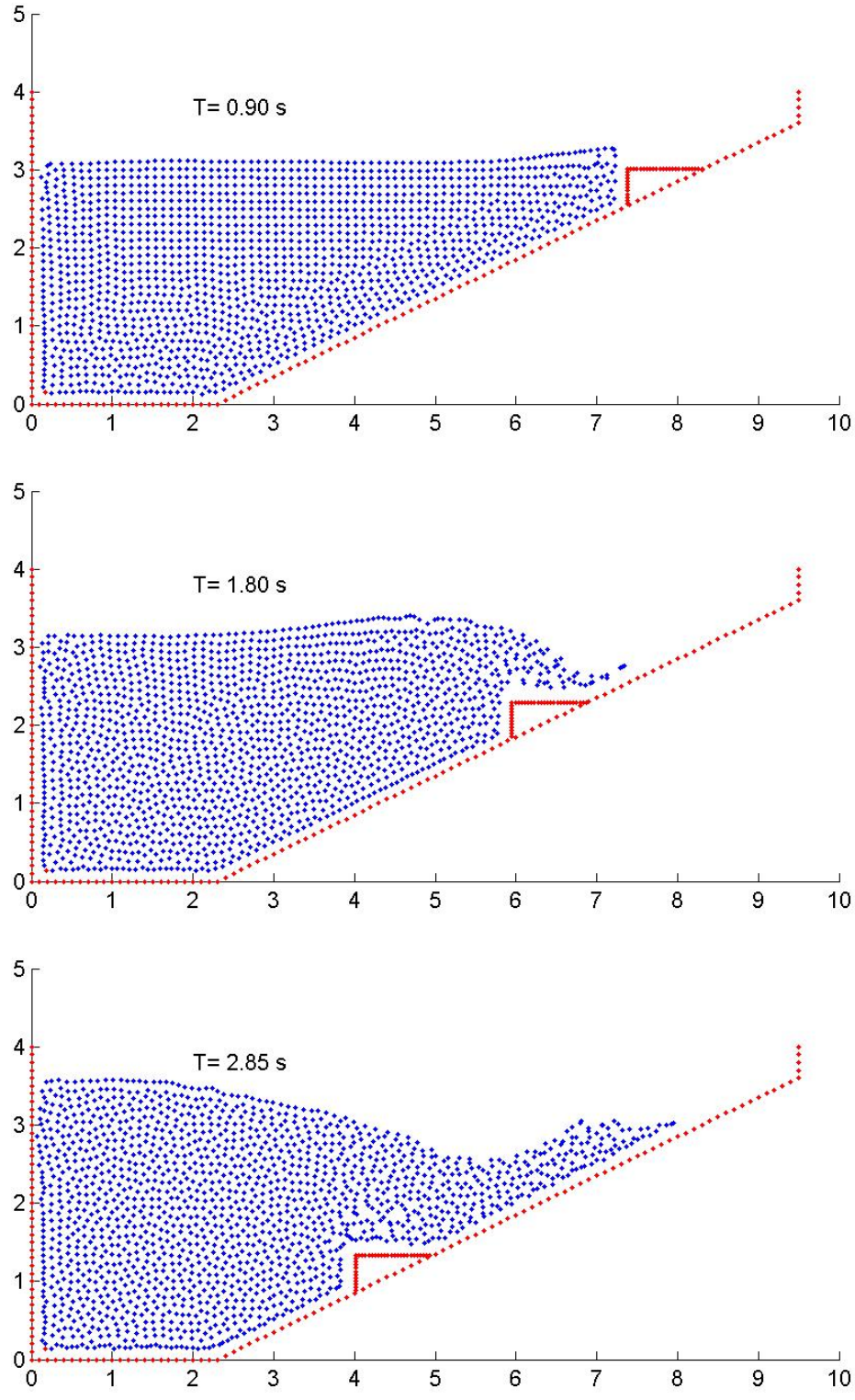


**Figure 4.9:** Tsunami generation using sliding Wedge (for 2D).

4.5.2. Case 3D

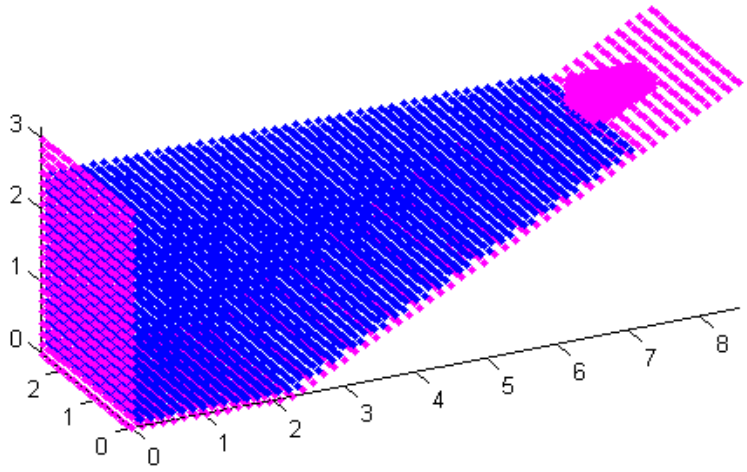


**Figure 4.10:** Initial configuration of Case4 in 3D.

| Variable Name | Value | Comments |
|---|---|---|
| i_kernel | 2 | Kind of kernel<br>2= Quadratic |
| i_algorithm | 1 | Kind of algorithm<br>1=Predictor-Corrector |
| i_shepard | 1 | Possibility to apply a Shepard filter (1=yes) |
| i_viscos | 3 | Viscosity:<br>(1) Artificial<br>(2) Laminar<br>(3) Laminar +SPS |
| Viscos_val | 1.0e-6 | If *i_viscos* = 1 it corresponds to α (Monaghan and Koss, 1999)<br>*If i_viscos* = 2 or 3 it corresponds to ν (kinematical viscosity on the order of $10^{-6}$). |
| h_SWL | 2.5 | Still water level (m).<br>Used to calculate B and the speed of sound[1]. |
| coef | 16 | Coefficient to calculate B.<br>Recommended values( 10 , 40 )[1] |
| IBC | 1 | Boundary Conditions<br>1= Monaghan & Kos (1999)<br>2= Dalrymple & Knio (2000) |
| visc_wall | 8.0e-3 | Wall viscosity value for Repulsive Force BC |

| | | |
|---|---|---|
| i_geometry | 2 | Geometry of the medium<br>1= BOX whose bottom can be tilted in X and Y<br>2= Beach |
| lattice | 2 | Lattice: (1) SC; (2) BCC |
| vlx,vly,vlz | 8.5,2.7,3.0 | Dimension of the medium |
| dx,dy,dz | 0.15,0.15,0.15 | Initial interparticle spacing |
| vlx1 | 2.25 | Length of Flat Domain |
| beta_deg | 26.565051 | Slope (deg) of the inclined plane ( beta ) ?? |
| i_periodicOBs(1),i_periodicOBs(2),i_periodicOBs(3) | 0,1,0 | Periodic Lateral boundaries in X, Y, & Z-Directions ? (1=yes) |
| iopt_wavemaker | 0 | If wavemaker will be added, left wall is not needed (1=yes) |
| iopt_obstacle | 0 | Add obstacle (1=yes) |
| iopt_wavemaker | 0 | Add wavemaker (1=yes) |
| iopt_RaichlenWedge | 1 | Add Raichlen wedge (1=yes) |
| bigDelta | 0.3 | Enter block-top elevation above SWL |
| specificWeight | 2.14 | Enter specific Weight |
| block_length, block_height, block_width | 0.91, 0.455, 0.61 | Enter block_length, block_height, block_width |
| dxW, dyW, dzW | 0.04, 0.04, 0.04 | Enter block discretization size |
| i_water | 1 | Add water in the flat region ?? (1=yes) |
| XXmin,XXmax | 0.075, 2.21 | Extent of fluid in X |
| YYmin,YYmax | 0.0751, 2.651 | Extent of fluid in Y |
| ZZmin,ZZmax | 0.075, 2.5 | Extent of fluid in Z |
| i_water2 | 1 | Add water in the inclined region ?? (1=yes) |
| XXmin,XXmax | 2.32, 8.5 | Extent of fluid in X |
| YYmin,YYmax | 0.0751, 2.651 | Extent of fluid in Y |
| ZZmin,ZZmax | 0.075, 2.5 | Extent of fluid in Z |
| i_wave | 0 | Add a solitary wave ?? (1=yes) |
| tmax,out | 3.0,0.011 | tmax: Run duration (seconds)<br>out: Recording time step (seconds) |
| trec_ini | 0.0 | trec_ini: initial time of outputting general data (seconds) |
| dtrec_det,t_sta_det,t_end_det | 0,1,-1 | Detailed recording |

| | | |
|---|---|---|
| | | step, Start time, End Time<br>A wrong sequence where<br>t_sta_det > t_end_det disables this option |
| dt,ivar_dt | 0.0001729, 0 | dt: Initial time step (seconds)<br>ivar_dt: Controls variable time step (If *ivar_dt* = 1 $\Rightarrow$ variable time step)[2] |
| coefficient | 0.866025 | Coeffficient to calculate the smoothing length (h) in terms of dx,dy,dz<br>h=coefficient*sqrt(dx*dx+dy*dy+dz*dz) |
| i_compile_opt | 3 | Compiler : 1=gfortran, 2=ifort, 3=HP 6.5 |

**4.6. Test case 5:** 3D dam-break interaction with a structure

The case can be run using *Case5.bat* whose output directory is *Case5*. The input file *Case5.txt* is located in the output directory. The information contained in that file can be summarized as follows:

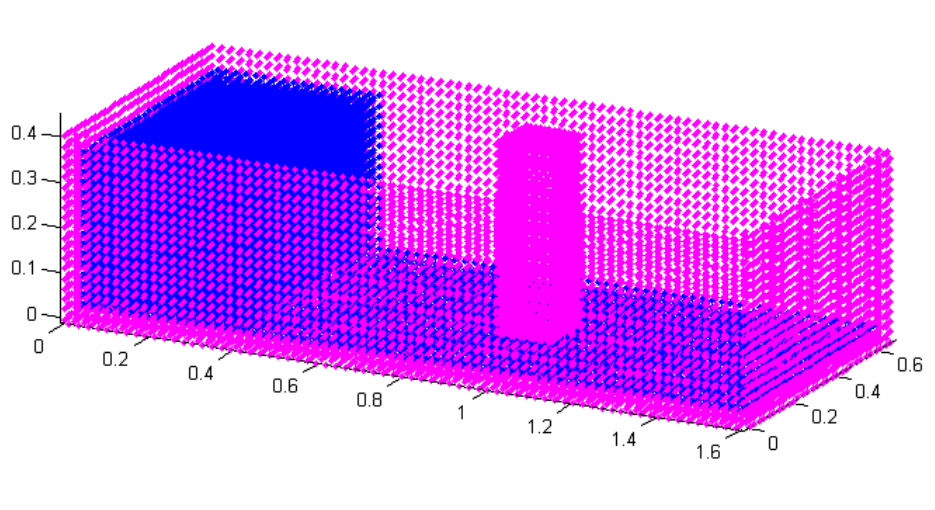


**Figure 4.11:** Initial configuration of Case5.

| Variable Name | Value | Comments |
|---|---|---|
| i_kernel | 3 | Kind of kernel<br>3= Cubic Spline |
| i_algorithm | 1 | Kind of algorithm<br>1=predictor corrector |
| i_shepard | 0 | Possibility to apply a Shepard filter (1=yes) |
| i_viscos | 1 | Viscosity:<br>(1) Artificial<br>(2) Laminar<br>(3) Laminar +SPS |
| Viscos_val | 0.1 | If *i_viscos* = 1 it corresponds to α (Monaghan and Koss, 1999)<br>*If i_viscos* = 2 or 3 it corresponds to ν (kinematical viscosity on the order of $10^{-6}$). |
| h_SWL | 0.30 | Still water level (m).<br>Used to calculate B and the speed of sound[1]. |
| coef | 25 | Coefficient to calculate B.<br>Recommended values( 10 , 40 )[1] |

| | | |
|---|---|---|
| IBC | 2 | Boundary Conditions<br>1= Monaghan & Kos (1999)<br>2= Dalrymple & Knio (2000) |
| i_geometry | 1 | Geometry of the medium<br>1= BOX whose bottom can be tilted in X and Y<br>2= Beach |
| lattice | 2 | Lattice: (1) SC; (2) BCC |
| vlx,vly,vlz | 1.6,0.67,0.4 | Dimension of the medium |
| dx,dy,dz | 0.0225,0.0225, 0.0225 | Initial interparticle spacing |
| beta_deg | 0 | inclination of floor (X direction) |
| Theta_deg | 0 | inclination of floor (Y direction) |
| iopt_addwall | 0 | Add wall (1=y) |
| iopt_addwallslot | 0 | Add wall with slot (1=y) |
| iopt_addwallhole | 0 | Add wall with hole (1=y) |
| iopt_obst | 1 | Add obstacle (1=y) |
| iopt_kind | 2 | Kind of obstacle (1) Solid, (2) Solid Walls |
| ndes | 2 | Density of points(ndens): dxi_new=dxi_old/ndens (ndens >1 increases density) |
| XMin, Xmax | 0.9,1.02 | Cube containing particles in X |
| YMin, Ymax | 0.24,0.36 | Cube containing particles in Y |
| ZMin, Zmax | 0.,0.45 | Cube containing particles in Z |
| slope | 90 | slope in X direction |
| iopt_obst | 0 | Add new obstacle (1=y) |
| iopt_wavemaker | 0 | Add wavemaker (1=y) |
| iopt_gate | 0 | Add gate (1=y) |
| initial | 2 | Initial configuration of the fluid<br>1= Set of particles without grid<br>2= Particles. on a grid filling the box till a certain height<br>3= Particles on a grid filling a part of the box till a certain height<br>4=Round shaped drop |

| | | |
|---|---|---|
| i_correct_pb | 0 | Initial pressure is corrected at fluid particles (hydrostatic). If *i_correct_pb* =1 it's also done for boundary particles. |
| XXmin,XXmax | 0.0225,0.4 | Extent of fluid in X |
| YYMin, YYmax | 0.0225,0.36 | Extent of fluid in Y |
| ZZmin,ZZmax | 0.0225,0.6475 | Extent of fluid in Z |
| iopt_fill_part | 1 | If *iopt_fill_part* =1 a new region will be filled<br>The previous parameters (since i_correct_pb) will be asked. |
| XXmin,XXmax | 0.415,0.875 | Extent of fluid in X |
| YYMin, YYmax | 0.0225,0.6475 | Extent of fluid in Y |
| ZZmin,ZZmax | 0.0225,0.03 | Extent of fluid in Z |
| iopt_fill_part | 1 | If *iopt_fill_part* =1 a new region will be filled<br>The previous parameters (since i_correct_pb) will be asked. |
| XXmin,XXmax | 0.9,1.025 | Extent of fluid in X |
| YYMin, YYmax | 0.0225,0.2175 | Extent of fluid in Y |
| ZZmin,ZZmax | 0.0225,0.03 | Extent of fluid in Z |
| iopt_fill_part | 1 | If *iopt_fill_part* =1 a new region will be filled<br>The previous parameters (since i_correct_pb) will be asked. |
| XXmin,XXmax | 0.9,1.025 | Extent of fluid in X |
| YYMin, YYmax | 0.3825,0.6475 | Extent of fluid in Y |
| ZZmin,ZZmax | 0.0225,0.03 | Extent of fluid in Z |
| iopt_fill_part | 1 | If *iopt_fill_part* =1 a new region will be filled<br>The previous parameters (since i_correct_pb) will be asked. |
| XXmin,XXmax | 1.0425,1.5775 | Extent of fluid in X |
| YYMin, YYmax | 0.0225,0.6475 | Extent of fluid in Y |
| ZZmin,ZZmax | 0.0225,0.03 | Extent of fluid in Z |
| iopt_fill_part | 0 | |
| tmax,out | 2,0.01 | tmax: Run duration (seconds)<br>out: Recording time step (seconds) |
| trec_ini | 0.0 | trec_ini: initial time of outputting general data (seconds) |

| | | |
|---|---|---|
| dtrec_det,t_sta_det,t_end_det | 0,1,-1 | Detailed recording<br>step, Start time, End Time<br>A wrong sequence where<br>t_sta_det > t_end_det disables this option |
| dt,ivar_dt | 0.0001, 1 | dt: Initial time step (seconds)<br>ivar_dt: Controls variable time step<br>(If *ivar_dt* = 1 ⇒ variable time step)[2] |
| coefficient | 0.85 | Coeffficient to calculate the smoothing length (h) in terms of dx,dy,dz<br>h=coefficient*sqrt(dx*dx+dy*dy+dz*dz) |
| i_compile_opt | 3 | Compiler : 1=gfortran, 2=ifort, 3=HP 6.5 |

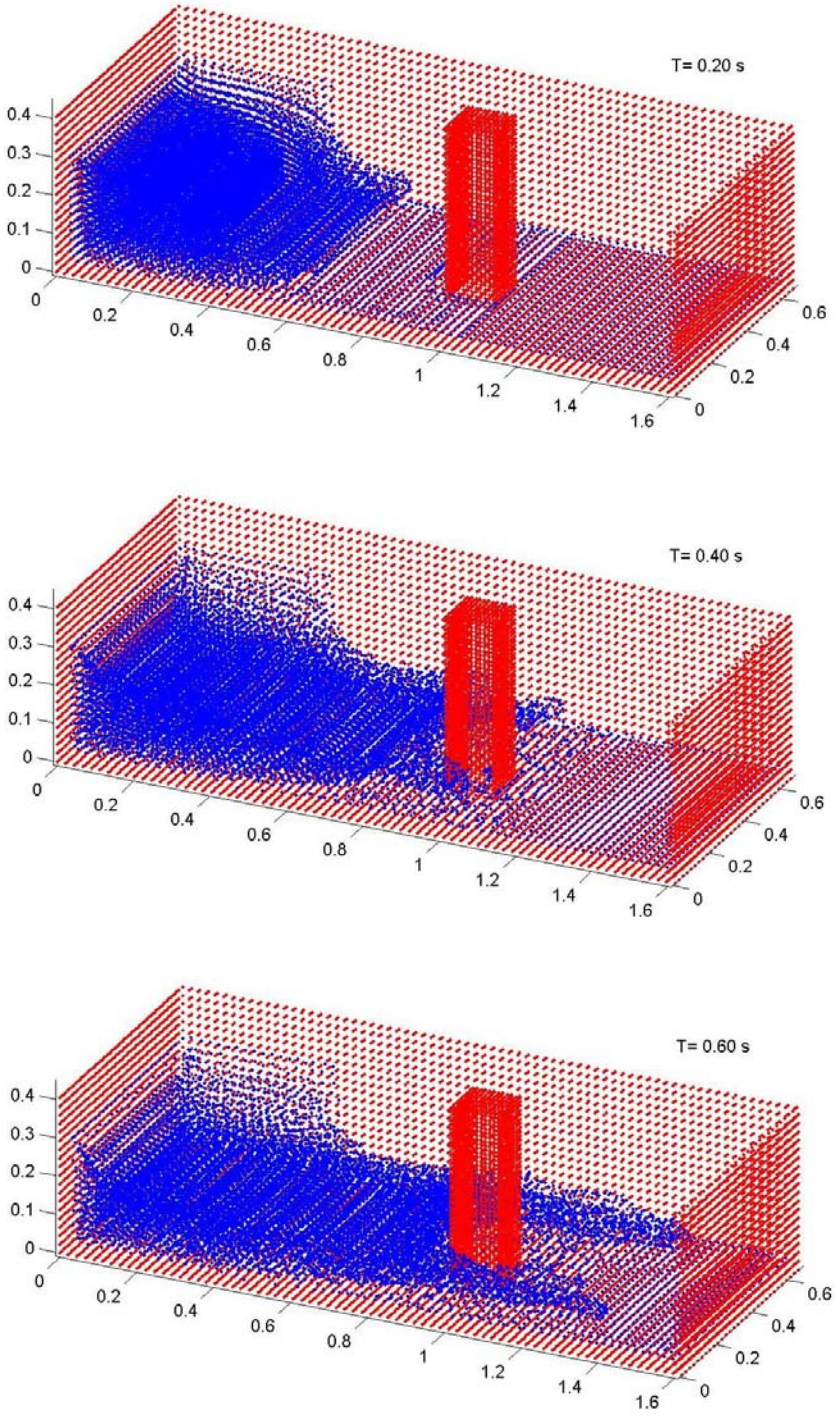


**Figure 4.12:** Interaction wave-structure in Case4

## 5. VISUALIZATION

To visualize the results obtained from SPHysics simulations, some basic post-processing programs have been provided in the SPHysics_2D/Post-Processing and SPHysics-3D/Post-Processing directories.

Detailed README files, explaining the procedure to view the results using Matlab and Paraview, are available in those directories . The user is encouraged to read these README files prior to using the visualization programs.

## 6. REFERENCES


Batchelor, G. K. 1974. *Introduction to fluid dynamics*. Cambridge University Press. U.K.

Benz W. 1990. Smoothed Particle Hydrodynamics: A review in *The numerical Modelling of Nonlinear Stellar Pulsations: Problems and Prospects*, J.R. Butchler ed., Kluwer Acad. Publ. 269-288

Bonet, J. and Kulasegaram, S. 2000. Corrections and stabilization of smooth particle hydrodynamics methods with applications in metal forming simulations. *International Journal for Numerical Methods In Engineering*, 47: 1189-1214.

Crespo, A.J.C., Gómez- Gesteira, M and Dalrymple, R.A. 2007. Boundary conditions generated by dynamic particles in SPH methods. *Computers, materials & continua,* 5(3): 173-184.

Gómez-Gesteira, M. and Dalrymple, R. 2004. Using a 3D SPH method for wave impact on a tall structure. *J. Wtrwy. Port, Coastal and Ocean Engrg.* 130(2): 63-69.

Gómez-Gesteira, M., Cerqueiro, D., Crespo, C., and Dalrymple, R. 2005. Green water overtopping analyzed with a SPH model, *Ocean Engineering.* 32: 223-238.

Gotoh, H., Shao S., and Memita, T. 2004. SPH-LES model for numerical investigation of wave interaction with partially immersed breakwater. *Coastal Engineering Journal,* 46(1): 39-63.

Dalrymple, R.A. and Knio, O. 2000. SPH modelling of water waves," *Proc. Coastal Dynamics,* Lund.

Dalrymple, R.A. and Rogers, B.D. 2006. Numerical modeling of water waves with the SPH method. *Coastal Engineering* 53: 141 – 147

Liu, G.R. 2003. Mesh Free methods: Moving beyond the finite element method. CRC Press, pp. 692.

Monaghan, J. J. 1982 Why particle methods work. *Siam J. Sci. Stat. Comput*. 3: 422-433.

Monaghan, J. J. 1989. On the problem of penetration in particle methods. *Journal Computational Physics*, 82: 1-15.

Monaghan, J. J. 1992. Smoothed particle hydrodynamics. *Annual Rev. Astron. Appl.*, 30: 543- 574.

Monaghan, J. J. 1994. Simulating free surface flows with SPH. *Journal Computational Physics*, 110: 399- 406.

Monaghan, J. J. 2000. SPH without tensile instability. *Journal Computational Physics,* 159: 290-311.

Monaghan, J. J. 2005. Smoothed Particle Hydrodynamics. *Rep. Prog. Phys.* 68: 1703-1759.

Monaghan, J. J. and Kos, A. 1999. Solitary waves on a Cretan beach. *J. Wtrwy. Port, Coastal and Ocean Engrg.*, 125: 145-154.

Monaghan, J.J., and Lattanzio, J.C., 1985. A refined method for astrophysical problems. *Astron. Astrophys*. 149: 135–143.

Morris, J.P., Fox, P.J. and Shu, Y. 1997. Modeling lower Reynolds number incompressible flows using SPH. *Journal Computational Physics,* 136: 214-226.

Peskin, C. S. 1977. Numerical analysis of blood flow in the heart. *Journal Computational Physics* 25: 220- 252.

Verlet, L. 1967. Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. *Phys. Rev*. 159: 98-103.

Wendland, H. 1995. Piecewiese polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in computational Mathematics* 4(1): 389- 396.