User Guide for the SPHysics code

September 2010

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
- ESPHI (An European Smooth Particle Hydrodynamics Initiative) project supported by the Commission of the European Communities (Marie Curie Actions, contract number MTKI-CT-2006-042350).
- Flood Risk Management Research Consortium (FRMRC) Phase 2, EPSRC Grant F020511
- Research Councils UK (RCUK) Research Fellowship

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 the user to test the program and verify that it is installed correctly. For 3-D applications a complex geometry generator is now provided making the creation of new geometries and loading in CAD files more accessible.

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 ρ_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:

$$\mathrm{W}(r,h) = \alpha_D \exp\left(-q^2\right) \tag{1.3}$$

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

2) Quadratic:

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

where α_D is $2/(\pi h^2)$ in 2D and $5/(4\pi h^3)$ in 3D

3) Cubic spline:

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

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

4) Quintic (Wendland, 1995):

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

where α_D is $7/(4\pi h^2)$ in 2D and $21/16\pi h^3)$ in 3D.

The tensile correction (Monaghan, 2000) is automatically activated when using kernels with first derivatives that go to zero with decreasing inter-particle spacing.

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

$$\left(-\frac{1}{\rho}\vec{\nabla}\mathrm{P}\right)_a = -\sum_{\mathrm{b}} \mathrm{m_b}\left(\frac{\mathrm{P_b}}{\rho_{\mathrm{b}}^2}+\frac{\mathrm{P_a}}{\rho_{\mathrm{a}}^2}\right)\vec{\nabla}_{\mathrm{a}}\mathrm{W_{ab}} \tag{1.9}$$

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

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

with $\mu_{ab} = \dfrac{h\vec{v}_{ab}\vec{r}_{ab}}{\vec{r}_{ab}^{\,2}+\eta^2}$; where $\vec{r}_{ab} = \vec{r}_a - \vec{r}_{\mathrm{b}}$, $\vec{v}_{ab} = \vec{v}_a - \vec{v}_b$; being $\vec{r}_k$ and $\vec{v}_k$ the position and the velocity corresponding to particle k ($\underline{a}$ or $\underline{b}$); $\overline{\mathrm{c_{ab}}} = \dfrac{\mathrm{c_a}+\mathrm{c_b}}{2}$. $\eta^2 = 0.01\mathrm{h}^2$, α 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(Lo and Shao 2002) to:

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

where υ_0 is the kinetic viscosity of laminar flow ($10^{-6} m^2/s$).
So, in SPH notation, Eq. 1.11 can be written as:

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

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. See Appendix for a brief description of the theory of Large-Eddy Simulation (LES) and Sub-Grid Scale (SGS) models. 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): $\frac{\tau_{ij}}{\rho} = \nu_t\left(2S_{ij} - \frac{2}{3}k\delta_{ij}\right) - \frac{2}{3}C_I\Delta^2\delta_{ij}\left|S_{ij}\right|^2$, where $\vec{\tau}_{ij}$ is the sub-particle stress tensor, $v_t = \left[(C_s\Delta l)\right]^2|S|$ the turbulence eddy viscosity, k the SPS turbulence kinetic energy, C_s the Smagorinsky constant (0.12), C_I = 0.0066, Δl 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

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

1.4. Continuity equation

Changes in the fluid density are calculated in SPHysics using

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

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)

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

where ε=0.5 and $\overline{\rho_{ab}} = (\rho_a + \rho_b)/2$. This method moves <u>*a*</u> 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 Ψ_{ab} refers to viscosity terms, which can be calculated using the different approaches mentioned above.

1.8 Density Reinitialization

While the dynamics from SPH simulations are generally realistic, the pressure field of the particles exhibits large pressure oscillations. Efforts to overcome this problem have concentrated on several approaches including correcting the kernel (for an overview see Bonet & Lok, 1999) and developing an incompressible solver. One of the most straight forward and computationally least expensive is to perform a filter over the density of the particles and the re-assign a density to each particle (Colagrossi and Landrini, 2003). There are two orders of correction, zeroth order and first order.

Zeroth Order – Shepard Filter

The Shepard filter is a quick and simple correction to the density field, and the following procedure is applied every 30 time steps

$$\rho_a^{new} = \sum_b \rho_b \tilde{W}_{ab} \frac{m_b}{\rho_b} = \sum_b m_b \tilde{W}_{ab} \tag{1.20}$$

where the kernel has been corrected using a zeroth-order correction:

$$\tilde{W}_{ab} = \frac{W_{ab}}{\sum_b \tilde{W}_{ab} \frac{m_b}{\rho_b}} \tag{1.21}$$

First Order – Moving Least Squares (MLS)

The Moving Least Squares (MLS) approach was developed by Dilts (1999) and applied successfully by Colagrossi and Landrini (2003) and by Panizzo (2004). This is a first-order correction so that the linear variation of the density field can be exactly reproduced:

$$\rho_a^{new} = \sum_b \rho_b W_{ab}^{MLS} \frac{m_b}{\rho_b} = \sum_b m_b W_{ab}^{MLS} \tag{1.22}$$

The corrected kernel is evaluated as follows:

$$W_{ab}^{MLS} = W_b^{MLS}(\vec{r}_a) = \beta(\vec{r}_a) \cdot (\vec{r}_a - \vec{r}_b)\, W_{ab} \tag{1.23}$$

so that in 2-D

$$W_{ab}^{MLS} = [\beta_0(\vec{r}_a) + \beta_{1x}(\vec{r}_a)(x_a - x_b) + \beta_{1z}(\vec{r}_a)(z_a - z_b)]W_{ab} \tag{1.24}$$

where the correction vector β is given by

$$\beta(\vec{r}_a) = \begin{bmatrix} \beta_0 \\ \beta_{1x} \\ \beta_{1z} \end{bmatrix} = \mathbf{A}^{-1} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \text{where} \quad \mathbf{A} = \sum_b W_b(\vec{r}_a)\, \tilde{\mathbf{A}} V_b \tag{1.25}$$

with the matrix $\tilde{\mathbf{A}}$ being given by

$$\tilde{\mathbf{A}} = \begin{bmatrix} 1 & (x_a - x_b) & (z_a - z_b) \\ (x_a - x_b) & (x_a - x_b)^2 & (z_a - z_b)(x_a - x_b) \\ (z_a - z_b) & (x_a - x_b)(z_a - z_b) & (z_a - z_b)^2 \end{bmatrix} \tag{1.26}$$

Similar to the Shepard filter this is applied every 30 time steps or similar. The equations are similar in 3-D but just include the *y*-direction.

1.9 Kernel Renormalization

A periodic correction of the kernel function W_{ab} is necessary in SPH hydraulics computations, where a finite domain and a free surface are often part of the computational domain. Particles near boundaries or the free surface have a kernel smoothing function truncated due to the absence of neighboring particles. The

conditions of consistency and normalization fail. However it is still possible to handle these situations by opportunely correcting the kernel function W_{ab} itself or its gradient.
In SPHysics, there are two techniques to avoid errors from a corrupted interpolating function:

1.9.1 Kernel correction.

The method was proposed by Bonet and Lok (1999) and, in an alternative form, by Liu et al., (1997). The kernel is modified to ensure that polynomial functions are exactly interpolated up to a given degree. In spite of the first-order correction (linear correction) is described in detail in Bonet and Lok (1999), the same authors consider that the linear correction is unsuitable for computational purposes. They also propose using constant, rather than linear, correction. So, a vectorial variable ($\vec{f}_a$) can be expressed as

$$\vec{f}_a = \frac{\sum_b \frac{m_b}{\rho_b} \vec{f}_b W_{ab}}{\sum_b \frac{m_b}{\rho_b} W_{ab}} \tag{1.27}$$

1.9.2 Kernel gradient correction.

The corrected kernel gradient $\tilde{\nabla} W_{ab}$ should be used to calculate the forces in the equation of motion instead of the normal kernel gradient ∇W_{ab}, being

$$\tilde{\nabla} W_{ab} = \vec{L}_b \nabla W_{ab} \tag{1.28}$$

$$L_a = M_a^{-1} \tag{1.29}$$

$$M_a = \sum_b^{num} \frac{m_b}{\rho_b} \nabla W_{ab} \otimes (x_a - x_b) \tag{1.30}$$

where *num* is the number of particles interacting with particle *a.*
Considering, for the sake of clarity, a 2D medium, the diagonal elements of M_a are defined positive since

$$\nabla W_{ab} = \frac{dW}{dr} \frac{1}{r_{ab}} (\vec{x}_a - \vec{x}_b) \tag{1.31}$$

with $r_{ab} = \left\| \vec{x}_a - \vec{x}_b \right\|$ resulting in

$$M_a(1,1) = \left(- \sum_{b=1}^{num} \frac{m_b}{\rho_b} \frac{dW}{dr} \frac{1}{r_{ab}} (x_a - x_b)^2 \right) \tag{1.32}$$

with $dW/dr < 0$.
The same can be proved for

$$M_a(2,2) = \left(- \sum_{b=1}^{num} \frac{m_b}{\rho_b} \frac{dW}{dr} \frac{1}{r_{ab}} (z_a - z_b)^2 \right) \tag{1.33}$$

On the other hand, $\vec{M}_a$ is symmetric since

$$M_a(1,2) = M_a(2,1) = -\left(\sum_{b=1}^{num} \frac{m_b}{\rho_b} \frac{dW}{dr} \frac{1}{r_{ab}} (x_a - x_b)(z_a - z_b) \right) \tag{1.34}$$

Note that matrix $\vec{M}$ and its inverse $\vec{L}$ are equal to the identity matrix when the particle a is placed far from the boundaries or the free- surface. In this case, there is no real correction on the kernel gradient (on the force). Nevertheless, when the particle a is placed near the boundaries or the free surface, the distribution of particles around it is not symmetric anymore. Thus, both $\vec{M}$ and $\vec{L}$ are different from the identity matrix and the kernel gradient is corrected following

$$\begin{bmatrix} \tilde{\nabla} W_x \\ \tilde{\nabla} W_z \end{bmatrix} = \begin{bmatrix} L_a(1,1) & L_a(1,2) \\ L_a(2,1) & L_a(2,2) \end{bmatrix} \begin{bmatrix} \nabla W_x \\ \nabla W_z \end{bmatrix} \tag{1.35}$$

where the subscripts x and z represent the spatial coordinates . Note that the correction is anisotropic since the terms $L_a(1,2)$ and $L_a(2,1)$ involve both spatial coordinates.

1.10. Riemann Solver formulation

This section aims to introduce only the basic concepts behind including Riemann solvers into SPH. For a full presentation of the theory underpinning Riemann solvers, MUSCL upwinding and higher-order accurate schemes, the reader is referred to texts such as Toro (2001), etc. The main advantage of introducing Riemann solvers into SPH is that the pressure and velocity fluctuations present in so many of the SPH schemes for water are removed (e.g. Rogers *et al*., 2010).

1.10.1 Definition of the Riemann Problem.

The Riemann problem is defined as a discontinuity located at location x_0 in space:

$$f(x,t) = \begin{cases} f_L & x \le x_0 \\ f_R & x > x_0 \end{cases} \tag{1.36}$$

where the subscripts L and R denote left and right states respectively. For example, consider the simple case of the simple density discontinuity below:

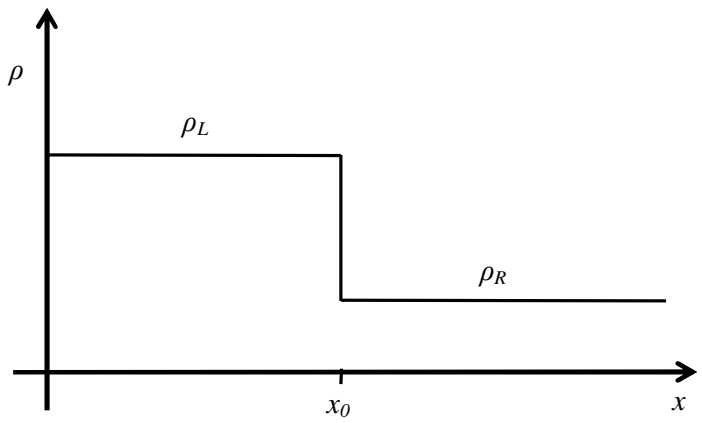
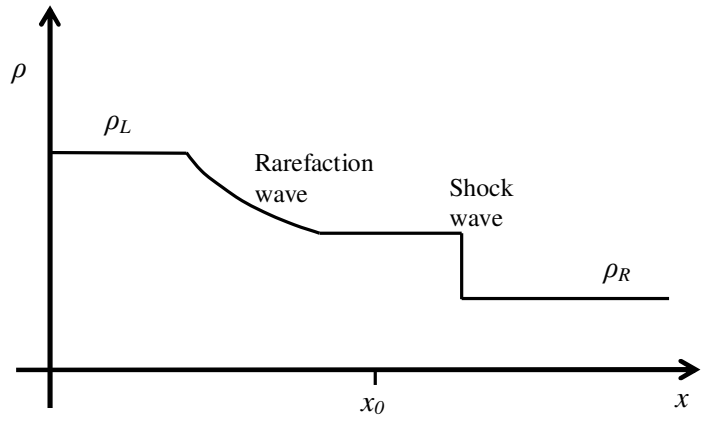
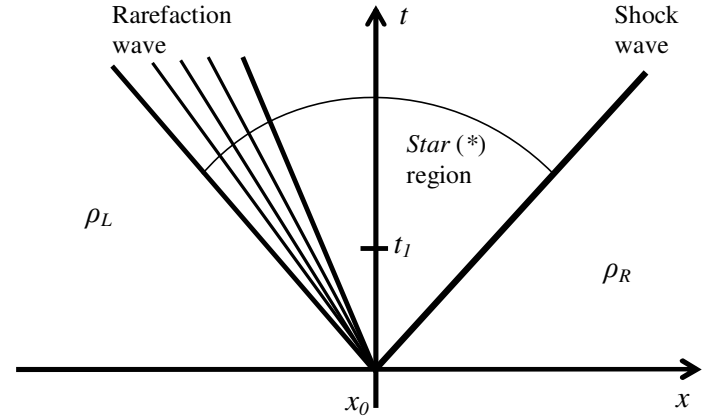


Figure 1.1 - Initial discontinuity in density

When this initial condition is evolved in time, a shock wave propagates to the right while a rarefaction wave propagates to the left as shown in Figure 1.2a below. This can be depicted in an *x-t* diagram which displays the shock propagating to the right as a single line and the rarefaction wave spreading out to the right separating the left and right regions. The region between the left and right regions is normally referred to as the star region.



(a) Propagation of initial discontinuity at time $t = t_1$



(b) *x-t* diagram

Figure 1.2 – Evolution of discontinuity in density

The solution to the Riemann problem is therefore the defined as all the states from the left to right regions. Numerous solvers have since been proposed to solve for the variables within the star region along with the speeds of the shock and rarefaction waves. If the problem is one dimensional (or reducible to 1-D), it is possible to use an exact Riemann solver, however, this is computationally expensive and so is generally avoided. To circumvent this, a range of approximate Riemann solvers have been proposed including the Osher, Random choice, Roe, HLLC, WAF approximate Riemann solvers (see Toro 2001). The problem can then be solved in terms of the primitive variables [ρ, u, v, w, e] (herein referred to as non-conservative), or the conserved variables [ρ, ρu, ρv, ρw, E].

In SPHysics we have implemented both primitive and conservative variable Riemann solvers.

1.10.2 Non-conservative Riemann Formulation.

In this formulation first proposed by Parshikov *et al*. (1999 & 2001) and later by Cha and Whitworth (2003) as Godunov Particle Hydrodynamics (GPH), the main change takes place with the pressure gradient term within the momentum equation. The reader is referred to these papers for a more detailed description. Essentially, the sum of the left and right pressures is replaced by the pressure within the star region, i.e. $\left(P_a + P_b\right) \rightarrow 2P^*_{ab}$. Hence the pressure gradient in the momentum equation is changed from:

$$\frac{\mathrm{d}\,\mathbf{v}_a}{\mathrm{d}\,t} = -\sum_b m_b \left(\frac{P_a}{\rho_a^2} + \frac{P_b}{\rho_b^2} \right) \nabla_a W_{ab} \tag{1.37}$$

to:

$$\frac{\mathrm{d}\,\mathbf{v}_a}{\mathrm{d}\,t} = -\sum_b m_b 2P^*_{ab} \left(\frac{1}{\rho_a^2} + \frac{1}{\rho_b^2} \right) \nabla_a W_{ab} \tag{1.38}$$

Alternatively, we can change the variationally consistent form (Vila, 1999; Bonet and Lok 1999; Colagrossi *et al*. 2003)

$$\frac{\mathrm{d}\,\mathbf{v}_a}{\mathrm{d}\,t} = -\sum_b m_b \left(\frac{P_a + P_b}{\rho_a \rho_b} \right) \nabla_a W_{ab} \qquad \text{to} \qquad \frac{\mathrm{d}\,\mathbf{v}_a}{\mathrm{d}\,t} = -\sum_b m_b \frac{2P^*_{ab}}{\rho_a \rho_b} \nabla_a W_{ab} \tag{1.39}$$

A similar operation concerns the velocities resolved onto the line joining two particle centres $\left(U_a^R + U_b^R\right) \rightarrow 2U^*_{ab}$. Parshikov *et al*. define $U^R = \mathbf{u} \cdot \left(\frac{\mathbf{r}_{ba}}{r_{ba}} \right)$ so that the continuity equation changes from

$$\frac{\mathrm{d}\,\rho_a}{\mathrm{d}\,t} = \sum_b m_b \left(\mathbf{u}_a - \mathbf{u}_b \right) \nabla_a W_{ab} \tag{1.40}$$

to:

$$\frac{\mathrm{d}\,\rho_a}{\mathrm{d}\,t} = -2\sum_b m_b \left(\mathbf{u}^*_{ab} - \mathbf{u}_a \right) \nabla_a W_{ab} \tag{1.41}$$

The intermediate star values are approximated initially using an acoustic-based solver. Denoting the values on the particles *a* and *b* as the left and right states, *L* and *R*, respectively, the star regions are approximated as:

$$U^*_{ab} = \frac{\rho_a c_a U_b^R + \rho_b c_b U_a^R + P_a - P_b}{\rho_a c_a + \rho_b c_b} \tag{1.42a}$$

$$P^*_{ab} = \frac{\rho_a c_a P_b + \rho_b c_b P_a - \rho_a c_a \rho_b c_b \left(U_b^R - U_a^R \right)}{\rho_a c_a + \rho_b c_b} \tag{1.42b}$$

Hence, the SPH equations become:

$$\frac{\mathrm{d}\rho_a}{\mathrm{d}t} = -2\sum_b m_b\left(\mathbf{u}^*_{ab} - \mathbf{u}_a\right)\nabla_a W_{ab} \tag{1.43a}$$

$$\frac{\mathrm{d}\mathbf{v}_a}{\mathrm{d}t} = -\sum_b m_b \frac{2P^*_{ab}}{\rho_a\rho_b}\nabla_a W_{ab} + \mathbf{g} + \textit{viscous terms} \tag{1.43b}$$

$$\frac{\mathrm{d}e_a}{\mathrm{d}t} = -\sum_b m_b \frac{2P^*_{ab}}{\rho_a\rho_b}\left(\mathbf{u}^*_{ab} - \mathbf{u}_a\right)\nabla_a W_{ab} \tag{1.43c}$$

1.10.3 Conservative Riemann Formulation.

In this formulation, we use the work of Vila (1999) whereby the conventional SPH formulation is replaced by one that solves a Riemann problem between each particle pair with the equations expressed in terms of the conserved variables [ρ, ρu, ρv, ρw, E]. The inviscid Navier-Stokes equations are solved so that no viscous terms are required to keep the scheme stable in contrast to our previous SPH schemes. We have chosen this formulation since the pressure fields and wave propagation are more accurate than the previous formulations used by the present authors (Dalrymple and Rogers, 2006). An accurate wave profile and pressure field are essential for predicting the forces on objects. More information on this formulation can be found in the papers of Vila (1999) and Guilcher *et al*. (2007). The governing equations when expressed in SPH form are given as:

$$\frac{\mathrm{d}\mathbf{x}_a}{\mathrm{d}t} = \mathbf{v}\left(\mathbf{x}_a, t\right) \tag{1.44a}$$

$$\frac{\mathrm{d}\omega_a}{\mathrm{d}t} = \omega_a \nabla\cdot \underline{v}_a \tag{1.44b}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\omega_a\rho_a\right) + \omega_a\sum_{b\in P}\omega_b 2\rho_A\left(\mathbf{v}_{A,ab} - \mathbf{v}^0\left(\mathbf{x}_{ab}, t\right)\right)\cdot\nabla_a W_{ab} = 0 \tag{1.44c}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\omega_a\rho_a\mathbf{v}_a\right) + \omega_a\sum_{b\in P}\omega_b 2\left[p_A + \rho_A\mathbf{v}_{A,ab}\otimes\left(\mathbf{v}^0_{A,ab} - \mathbf{v}^0\left(x_{ab}, t\right)\right)\right]\nabla_a W_{ab} = \omega_a f_a \tag{1.44d}$$

where subscript A denotes the result from the approximate Riemann solver, superscript 0 denotes the field value (i.e. the value at the particle itself), ω_a is the volume of particle a. This Riemann problem is solved using an HLLC approximate Riemann solver using MUSCL-based upwinding (Toro, 2001). f_a is the vector of external forces, for example, gravity.

1.10.4 The HLLC Approximate Riemann Solver.

The HLLC Riemann solver used in SPHysics provides an approximate solution for *Star Region* where the contact surface is reinstated (Toro *et al*. 1994). Now in the addition to the left and right going waves, there is a contact surface within the *Star Region* so that we also have *star* left and right regions *L and *R as shown in Figure 1.3. As we are solving compressible flow equations where a contact discontinuity might exist, it is important, therefore, to use a solver that captures this effect.

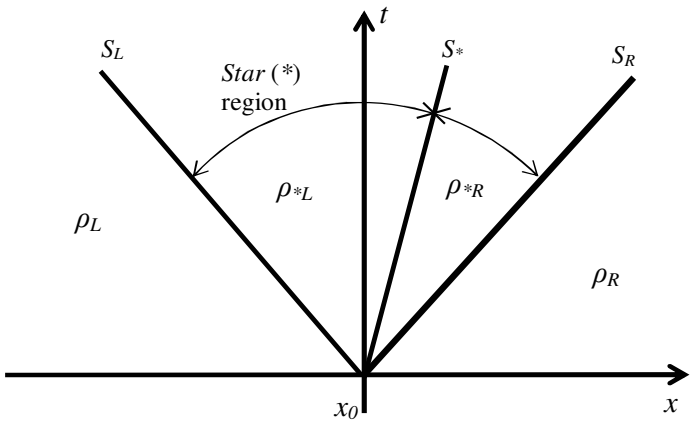


Figure 1.3 - Wave solution for HLLC Riemann solver - *x*-*t* diagram

Defining $\mathbf{Q} = [\rho, \rho u, \rho v, \rho w, E]$, and the wave speeds in the Left, Star and Right regions as S_L, S_* and S_R respectively, the solution to this is given as:

$$\mathbf{Q}_{HLLC} = \begin{cases} \mathbf{Q}_L & if \quad x/t \le S_L \\ \mathbf{Q}_{*L} & if \quad S_L \le x/t \le S_* \\ \mathbf{Q}_{*R} & if \quad S_* \le x/t \le S_R \\ \mathbf{Q}_R & if \quad S_R \le x/t \end{cases} \tag{1.45}$$

where

$$\mathbf{Q}_* = \rho_K \left(\frac{S_K - u_K}{S_K - S_*} \right) \begin{bmatrix} 1 \\ S_* \\ v_K \\ w_K \\ \frac{E_K}{\rho_K} + (S_* - u_K)\left[S_* + \frac{p_K}{\rho_K (S_K - u_K)} \right] \end{bmatrix}. \tag{1.46}$$

with the subscript K denoting left (K=L) or right (K=R), respectively. The wave speeds are defined as follows:

$$S_L = u_L - c_L q_L, \quad S_* = u_*, \quad S_R = u_R + c_R q_R, \tag{1.47}$$

where for Tait's or Morris' equations of state

$$q_K = \begin{cases} 1 & if \quad p_* \le p_K \\ \left[\frac{\left(\frac{\rho_K}{\rho_{*K}} \right)(p_K - p_*)}{(\rho_K - \rho_{*K}) c_K^2} \right]^{\frac{1}{2}} & if \quad p_* > p_K \end{cases} \tag{1.48}$$

1.10.5 Higher-order Reconstruction and MUSCL-based schemes.

It is well known in finite volume Godunov-type solvers that the first-order schemes are dissipative. However, attempting to obtain a higher-order scheme just using simple extrapolation leads to unphysical oscillation in the solution since this can lead to overestimation of the variables between each computation point. Using an approach similar to Monotone Upwind-centred Scheme for Conservation Laws (MUSCL) allows us to extrapolate the data to be second order in space.

In SPHysics, this Riemann problem is solved using an HLLC approximate Riemann solver using MUSCL-based upwinding Toro (2001) with a general β-limiter (Hirsch, 1998). For an arbitrary function Φ between two particles a and b, we construct the Riemann problem left and right states either side of the midpoint using the gradient of the constructed variable. Defining the gradient constructed variable differences as

$$\begin{aligned} \Delta\phi_a &= \nabla\phi_a \cdot \tfrac{1}{2}\mathbf{r}_{ba} \\ \Delta\phi_b &= \nabla\phi_b \cdot \tfrac{1}{2}\mathbf{r}_{ab} \end{aligned}, \tag{1.49}$$

the left and right Riemann states either side of the midpoint are then defined by

$$\begin{aligned} \phi_a^L &= \phi_a + \overline{\Delta\phi_a} \\ \phi_b^R &= \phi_b - \overline{\Delta\phi_b} \end{aligned}, \tag{1.50}$$

where $\overline{\Delta\phi_a}$ and $\overline{\Delta\phi_b}$ are given by

$$\overline{\Delta\phi_a}, \overline{\Delta\phi_b} = \begin{cases} \max[0, \min(\beta\Delta\phi_a, \Delta\phi_b), \min(\Delta\phi_a, \beta\Delta\phi_b)] & \text{if } \Delta\phi_b > 0 \\ \min[0, \max(\beta\Delta\phi_a, \Delta\phi_b), \max(\Delta\phi_a, \beta\Delta\phi_b)] & \text{if } \Delta\phi_b < 0 \end{cases}. \tag{1.51}$$

If either $\overline{\Delta\phi_a}$ or $\overline{\Delta\phi_b}$ is found to have a value greater than $\frac{1}{2}|\phi_a - \phi_b|$, then $\overline{\Delta\phi_a}$ and $\overline{\Delta\phi_b}$ are further limited to $\frac{1}{2}|\phi_a - \phi_b|$. This is done to ensure that the values of ϕ_a^L and ϕ_b^R always lie between the values of ϕ_a and ϕ_b. The left and right states are used as the discontinuous states for the individual Riemann problem between particles a and b. For a full description an understanding of Riemann solvers, the reader is referred to Toro (2001). As is well known for Riemann solvers and higher-order schemes, the choice of limiters can be sensitive.

2. IMPLEMENTATION

2.1. Time stepping

Four numerical schemes are implemented in SPHysics: (i) the Predictor-Corrector algorithm described by Monaghan (1989); (ii) the Verlet algorithm (Verlet, 1967); (iii) the Symplectic algorithm (Leimkhuler, 1997) and (iv) Beeman algorithm (Beeman, 1976).
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

$$\begin{aligned} \vec{v}_a^{\,n+1} &= \vec{v}_a^{\,n-1} + 2\Delta t \vec{F}_a^{\,n};\ \rho_a^{n+1} = \rho_a^{n-1} + 2\Delta t D_a^n \\ \vec{r}_a^{\,n+1} &= \vec{r}_a^{\,n} + \Delta t \vec{V}_a^{\,n} + 0.5\Delta t^2 \vec{F}_a^{\,n};\ e_a^{n+1} = e_a^{n-1} + 2\Delta t E_a^n \end{aligned} \tag{2.5}$$

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

$$\begin{aligned} \vec{v}_a^{\,n+1} &= \vec{v}_a^{\,n} + \Delta t \vec{F}_a^{\,n};\ \rho_a^{n+1} = \rho_a^{n} + \Delta t D_a^n \\ \vec{r}_a^{\,n+1} &= \vec{r}_a^{\,n} + \Delta t \vec{V}_a^{\,n} + 0.5\Delta t^2 \vec{F}_a^{\,n};\ \ e_a^{n+1} = e_a^{n} + \Delta t E_a^n \end{aligned} \tag{2.6}$$

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

2.1.3. Symplectic scheme

Symplectic time integration algorithms are time reversible in the absence of friction or viscous effects (Leimkhuler 1997) and hence represent a very attractive option for meshless particle schemes. In this case, first, the values of density and acceleration are calculated at the middle of the time step as:

$$\begin{aligned} \rho_a^{n+\frac{1}{2}} &= \rho_a^n + \frac{\Delta t}{2}\frac{\mathrm{d}\rho_a^n}{\mathrm{d}t}, \\ \underline{r}_a^{\,n+\frac{1}{2}} &= \underline{r}_a^{\,n} + \frac{\Delta t}{2}\frac{\mathrm{d}\underline{r}_a^{\,n}}{\mathrm{d}t}, \end{aligned} \tag{2.7}$$

where the superscript n denotes time step and $t = n\Delta t$. Pressure, $p_a^{n+\frac{1}{2}}$, is calculated using the equation of state. In the second stage $\mathrm{d}\left(\omega_i \rho_i \underline{v}_i\right)^{n+\frac{1}{2}} \Big/ \mathrm{d}t$ gives the velocity and hence position of particles at the end of the time step

$$\begin{aligned} \left(\omega_a \rho_a \underline{v}_a\right)^{n+1} &= \left(\omega_a \rho_a \underline{v}_a\right)^{n+\frac{1}{2}} + \frac{\Delta t}{2}\frac{\mathrm{d}\left(\omega_a \rho_a \underline{v}_a\right)^{n+\frac{1}{2}}}{\mathrm{d}t}, \\ \underline{r}_a^{\,n+1} &= \underline{r}_a^{\,n+\frac{1}{2}} + \frac{\Delta t}{2}\underline{v}_a^{\,n+1}. \end{aligned} \tag{2.8}$$

At the end of the timestep $\mathrm{d}\rho_a^{n+1}/\mathrm{d}t$ is calculated using the updated values of $\underline{v}_a^{\,n+1}$ and $\underline{r}_a^{\,n+1}$ (Monaghan 2005).

2.1.4. Beeman scheme

Beeman algorithm uses a Beeman predictor step and an Adams-Bashforth-Moulton corrector step. This method is accurate to $O(\Delta t^4)$.

The predictor step is fulfilled using Beeman's method as Capone *et al.* 2007:

$$\begin{aligned}
\vec{v}_a^{\,n+1/2} &= \vec{v}_a^{\,n} + 1.5\Delta t\vec{F}_a^{\,n} - 0.5\Delta t\vec{F}_a^{\,n-1} \\
\rho_a^{n+1/2} &= \rho_a^{n} + 1.5\Delta t D_a^{n} - 0.5\Delta t D_a^{n-1} \\
\vec{r}_a^{\,n+1/2} &= \vec{r}_a^{\,n} + \Delta t\vec{V}_a^{\,n} + \tfrac{2}{3}\Delta t^2\vec{F}_a^{\,n} - \tfrac{1}{6}\Delta t^2\vec{F}_a^{\,n-1} \\
e_a^{n+1/2} &= e_a^{n} + 1.5\Delta t E_a^{n} - 0.5\Delta t E_a^{n-1}
\end{aligned} \tag{2.9}$$

calculating $P_a^{n+1/2} = f(\rho_a^{n+1/2})$.

The corrector step is given by

$$\begin{aligned}
\vec{v}_a^{\,n+1} &= \vec{v}_a^{\,n} + \tfrac{5}{12}\Delta t\vec{F}_a^{\,n+1/2} + \tfrac{8}{12}\Delta t\vec{F}_a^{\,n} - \tfrac{1}{12}\Delta t\vec{F}_a^{\,n-1} \\
\rho_a^{n+1} &= \rho_a^{n} + \tfrac{5}{12}\Delta t D_a^{n+1/2} + \tfrac{8}{12}\Delta t D_a^{n} - \tfrac{1}{12}\Delta t D_a^{n-1} \\
\vec{r}_a^{\,n+1} &= \vec{r}_a^{\,n} + \Delta t\vec{V}_a^{\,n} + \tfrac{1}{6}\Delta t^2\vec{F}_a^{\,n+1/2} + \tfrac{1}{3}\Delta t^2\vec{F}_a^{\,n} \\
e_a^{n+1} &= e_a^{n} + \tfrac{5}{12}\Delta t E_a^{n+1/2} + \tfrac{8}{12}\Delta t E_a^{n} - \tfrac{1}{12}\Delta t E_a^{n-1}
\end{aligned} \tag{2.10}$$

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

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 δt is calculated according to Monaghan and Kos (1999):

$$\Delta t = 0.3\cdot\min(\Delta t_f, \Delta t_{cv})\ ;\ \Delta t_f = \min_a\left(\sqrt{h/|f_a|}\right)\ ;\ \Delta t_{cv} = \min_a \frac{h}{c_s + \max_b\left|\frac{h\vec{v}_{ab}\vec{r}_{ab}}{\vec{r}_{ab}^{\,2}}\right|} \tag{2.11}$$

Here Δt_f is based on the force per unit mass $|f_a|$, and Δt_{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.

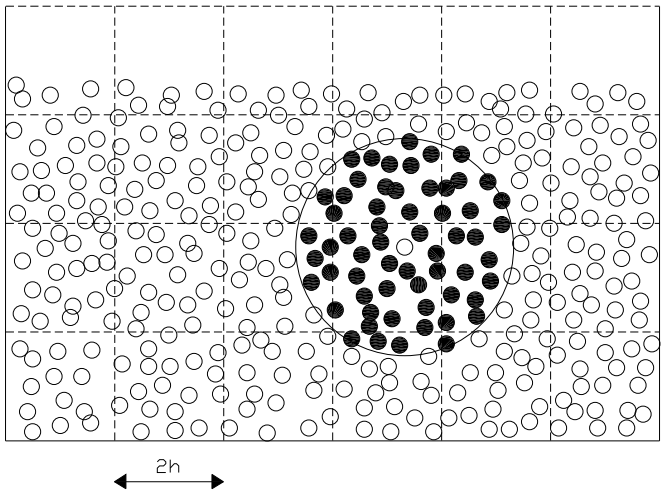


Figure 2.1: Set of neighboring particles in 2D. The possible neighbors of a fluid particle are in the adjacent cells but this only interacts with particles marked by black dots.

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).

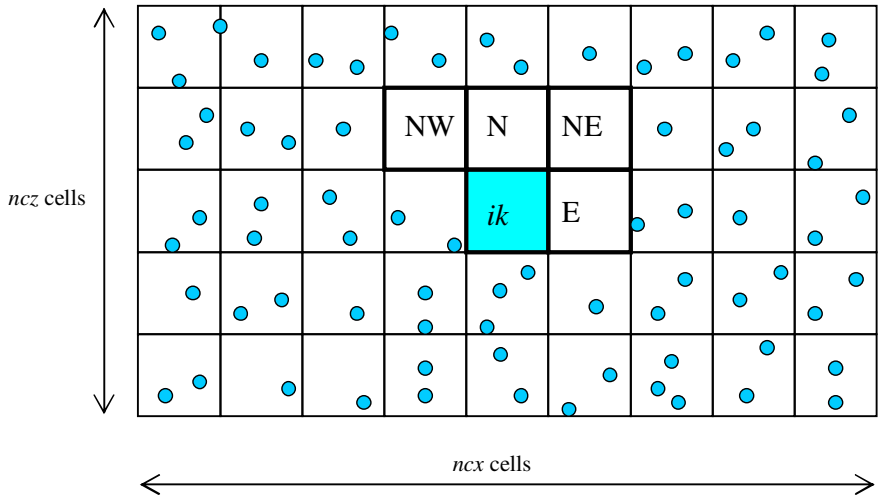


Figure 2.2: Sweeping through grid cells in 2D. Starting from the lower left corner, particles inside the center cell *ik* interact with adjacent cells only in E, N, NW and NE directions. The interactions with the rest of the cells W, S, SW & SE directions were previously computed using reverse interactions.

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

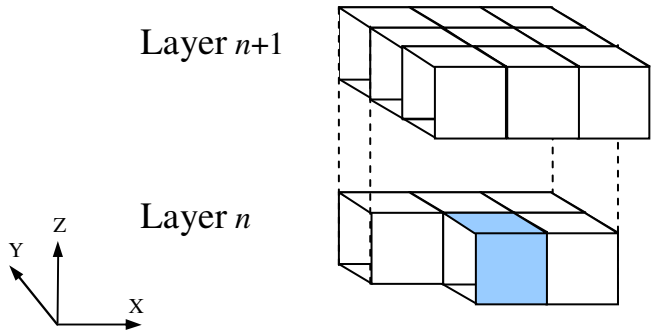


Figure 2.3: Sweeping through grid cells in 3D. Only 13 out of 26 possible neighboring cells are considered when centered on a particular *ijk* cell. The rest were previously considered when centered on adjacent cells using reverse interactions.

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; Rogers & Dalrymple 2008) 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):

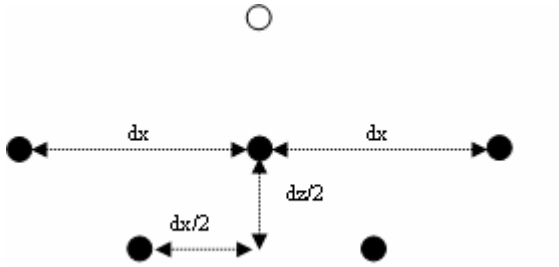


Figure 2.4: 2D sketch of the interaction between a fluid particle (empty circle) and a set of boundary particles (full circles). The boundary particles are placed in a staggered manner.

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 2*h*, 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.

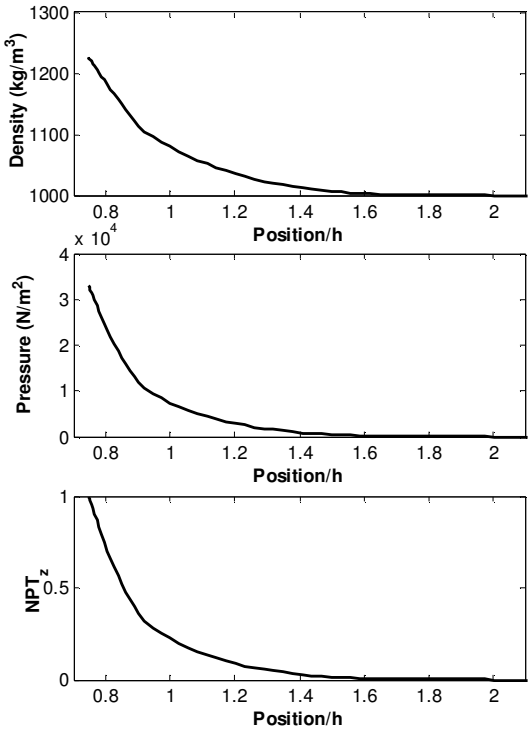


Figure 2.5: Variation of density (a), pressure (b) and normalized pressure term (c) for a moving particle approaching a solid boundary. Calculations were run without viscosity.

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 (Rogers & Dalrymple 2008)

$$\vec{f} = \vec{n} R(\psi) P(\xi) \varepsilon(z, u_\perp) \tag{2.12}$$

where $\vec{n}$ is the normal of the solid wall. The distance ψ is the perpendicular distance of the particle from the wall, while ξ is the projection of interpolation location ξ_i onto the chord joining the two adjacent boundary particles (i.e. the distance along the boundary of a fluid particle between two 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

$$R(\psi) = A \frac{1}{\sqrt{q}} (1 - q) \tag{2.13}$$

where the coefficient *A* is

$$A = \frac{1}{h} 0.01 c_i^2 \tag{2.14}$$

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

$$P(\xi) = \frac{1}{2}\left(1 + \cos\left(\frac{2\pi\xi}{\Delta b}\right)\right) \tag{2.15}$$

where Δb 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

$$\varepsilon(z, u_\perp) = \varepsilon(z) + \varepsilon(u_\perp) \tag{2.16}$$

where

$$\varepsilon(z) = \begin{cases} 0.02 & z \geq 0 \\ |z/h_o| + 0.02 & 0 > z \geq -h_o \\ 1 & |z/h_o| > 1 \end{cases} \tag{2.17}$$

and

$$\varepsilon(u_\perp) = \begin{cases} 0 & u_\perp > 0 \\ |20u_\perp|/c_o & |20u_\perp| < c_o \\ 1 & |20u_\perp| > c_o \end{cases} \tag{2.18}$$

In Equations 2.12-2.14, z is the elevation above the local still-water level h_o, $u_\perp = (\vec{v}_{WP} - \vec{v}_{BP}) \cdot \vec{n}$, 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 neighbors 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 $\vec{n} = \vec{t} \times \vec{s}$. Hence for a fluid particle, a, the distance $\xi = (\vec{r}_a - \vec{r}_{BP}) \cdot \vec{t}_{BP}$.

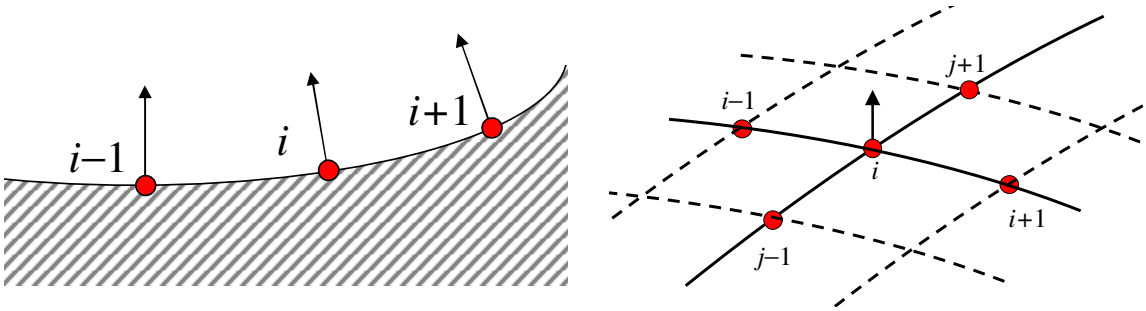


Figure 2.6: Location of adjacent boundary particles. (a) 2-D boundary particles and adjacent neighbors; (b) 3-D boundary particles and adjacent neighbors

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.

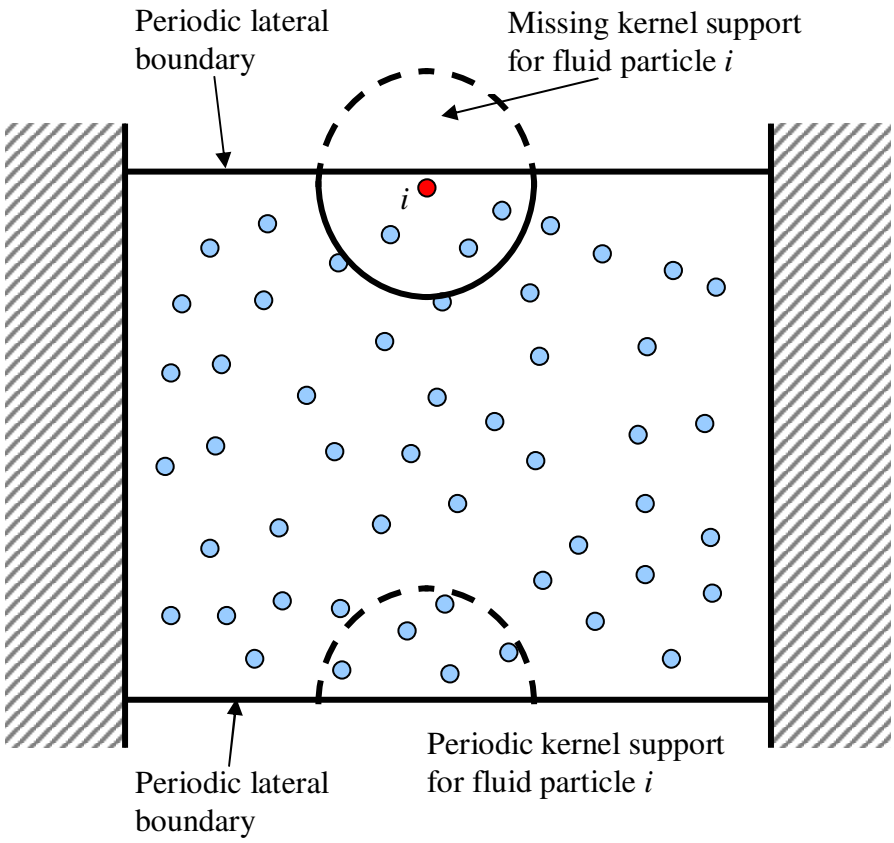


Figure 2.7: Periodic lateral boundaries: area of influence (kernel support) for particle *i* extend beyond top lateral boundary and is continued through periodic bottom boundary.

2.4.4. Floating Objects.

In SPHysics and SPH in general, the boundary is described by a set of discrete boundary particles which exert a repulsive force on water particles when they approach. There are several methods used to try and achieve the repulsion expected including ghost particles, stationary water particles and repulsive forces. In the SPHysics code, we have two types of boundary implemented: stationary water particles (referred to as dynamic boundary particles, Crespo *et al.* 2008) and repulsive wall particles which exert a force on approaching water particles with a singularity in the force field as the interparticle distance approaches zero (Monaghan and Kos, 1999; Rogers and Dalrymple, 2008). By summing the contributions exerted on the boundary particles for an entire body, the motion of a floating object can be evaluated and the object moved accordingly. Techniques such as the ghost particles method become very difficult and particularly unwieldy when there are corners or curved surfaces since when generating the ghost particles it is difficult to generate the correct force.

We assume that the objects are treated as rigid bodies. The force on each boundary particle is computed by summing up the contribution from all the surrounding water particles within the surrounding kernel. Hence, boundary particle k experiences a force per unit mass given by

$$\mathbf{f}_k = \sum_{a \in WPs} \mathbf{f}_{ka} \tag{2.19}$$

where *WPs* denotes water particles and $\mathbf{f}_{ka}$ is the force per unit mass exerted by water particle a on boundary particle k. By the principle of equal and opposite action and reaction, the force exerted by a water particle on each boundary particle is given by

$$m_k \mathbf{f}_{ka} = -m_a \mathbf{f}_{ak} \tag{2.20}$$

This is useful since during the simulation we only actually compute repulsive force, $\mathbf{f}_{ak}$, exerted by the boundary particle k on water particle a. However, using relation (2.20), we can estimate the force exerted on the moving body.

For the motion of the moving body, we use the equations of basic rigid body dynamics. The equations of motion of the body in the translational and rotational degrees of freedom are given by

$$M \frac{\mathrm{d}\mathbf{V}}{\mathrm{d}t} = \sum_{k \in BPs} m_k \mathbf{f}_k \quad \text{and} \quad I \frac{\mathrm{d}\mathbf{\Omega}}{\mathrm{d}t} = \sum_{k \in BPs} m_k (\mathbf{r}_k - \mathbf{R}_0) \times \mathbf{f}_k \tag{2.21}$$

where M is the mass of the object, I is the moment of inertia, $\mathbf{V}$ is the velocity of the object, $\mathbf{\Omega}$ is the rotational velocity of the object, $\mathbf{R}_0$ is the position of the centre of mass and *BPs* denotes boundary particles. Equations (2.21) are integrated in time to predict the values of $\mathbf{V}$ and $\mathbf{\Omega}$ for the beginning of the next time-step. Each boundary particle that describes the moving body has a velocity given by

$$\mathbf{u}_k = \mathbf{V} + \mathbf{\Omega} \times (\mathbf{r}_k - \mathbf{R}_0) \tag{2.22}$$

The boundary particles within the rigid body are then moved by integrating equation (2.22) in time. It can be shown that this technique conserves both linear and angular momenta (Monaghan *et al.*, 2003; Monaghan, 2005).

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 Λ_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 $2h \times 2h \times 2h$ (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 (see Fig. 2.8). 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. Furthermore, the number of boxes decreases when the particles fall down (last frame of Fig. 2.8). This generates important savings in execution time, since no redundant cell computations are performed.

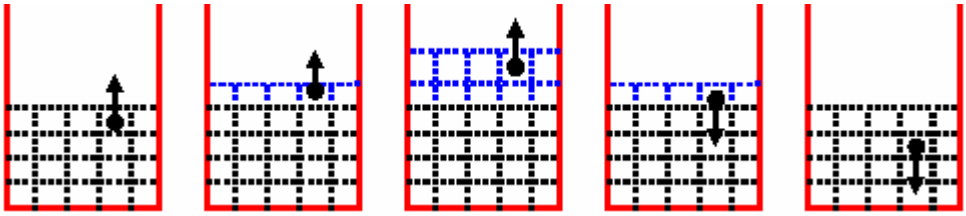


Figure 2.8: Evolution of new cells in Z direction depending on the fluid particles movement.

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.

2.6. Restart runs & checkpointing (repetitive restarts)

Restarting previous (unfinished) runs is controlled using the RESTART parameter. If the code is being run on computer clusters, there are sometimes limits as to how long a particular job can run, e.g. 24 hours. If the code is to run for more than 24 hours, then the code needs to be restarted repetitively, a process known as **checkpointing**. This can be specified when first launching the SPHysics code by setting the `i_restartRun` parameter in the Case files.

`i_restartRun` > 1 is used for Checkpointing = repetitive restarting of code (for clusters)

so that:
`i_restartRun` = 0 : Start new run, once only
`i_restartRun` = 1 : reStart old run, once only
`i_restartRun` = 2 : Start new run, with repetitive restarts (Checkpointing)
`i_restartRun` = 3 : reStart old run, with repetitive restarts (Checkpointing)

(Note: this parameter has been changed from v1.4 to v2.0.)

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 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.

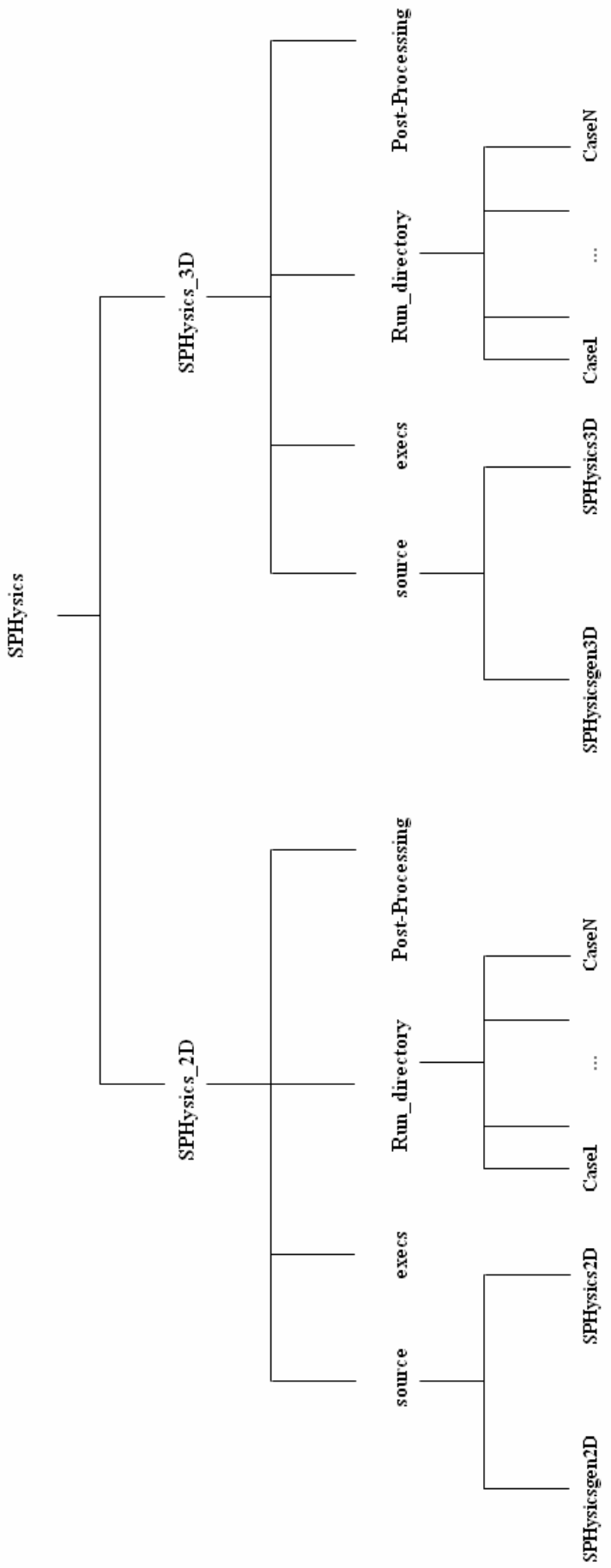


Figure 3.1. Directory tree.

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) Time stepping: (1=Predictor-Corrector;2=Verlet; 3=Symplectic; 4=Beeman).
iii) Density filter: (0=no filter; 1=Shepard; 2=MLS).
iv) Kernel correction: (0=none; 1=Kernel correction; 2=Gradient kernel Correction).
v) Viscosity treatment: (1=Artificial viscosity; 2=Laminar viscosity; 3=Laminar viscosity +SPS).
vi) Equation of state: (1=Weakly Compressible Fluid (Tait equation); 2=Ideal Gas Equation; 3=Incompressible Fluid (Poisson equation)).

vii) Boundary conditions: (1=Repulsive BC; 2=Dynamic BC).

viii) Choice of compilers: (1=gfortran; 2=ifort; 3=win_ifort; 4=Silverfrost FTN95). Please refer to section 4.1.1 for details on running the code on Windows using INTEL Fortran, 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 (referred to herein as Case files, e.g. Case1.txt) 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, Silverfrost FTN95, 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
i_algorithm
i_densityFilter
i_viscos
iBC
i_periodicOBs(1)
i_periodicOBs(2)
i_periodicOBs(3)
lattice
i_EoS
h_SWL
B
gamma
coef
eps
rho0
viscos_val
visc_wall
vlx
vly
vlz
dx
dy
dz
h
np
nb
nbf
ivar_dt
dt
tmax
out
trec_ini
dtrec_det
t_sta_det
t_end_det
i_restartRun
CFL_number
TE0

i_kernelcorrection	i_vort
iRiemannSolver	ndt_VerletPerform
iTVD	ndt_FilterPerform
beta_lim	ndt_DBCPerform

Description:

i_kernel: Kind of kernel (1=Gaussian; 2= Quadratic; 3= Cubic Spline; 5= Quintic).
i_algorithm: Kind of algorithm (1= Predictor1 Corrector algorithm; 2= Verlet algorithm; 3=Symplectic; 4=Beeman algorithm).
i_densityFilter: Use of a density filter: (0= no filter; 1=Shepard; 2=MLS).
i_viscos: Viscosity definition 1= Artificial; 2= Laminar; 3= Laminar + SPS
IBC: Boundary conditions. 1=Monaghan repulsive forces; 2= Dynamic boundaries.
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
i_EoS: Equation of State: (1) Tait equation; (2) Ideal Gas; (3) Poisson equation
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 kg/m^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, vly, vlz: medium extent in X, Y, Z direction.
dx, dy, dz: Initial interparticle spacing in x, y, 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.

i_restartRun: (0) Start a new RUN; (1) Restart an old RUN; (2) New with CheckPointg; (3) Restart with CheckPointing

CFL_number: Constant to calculate the time step following CFL condition (0.1, 0.5).

TE0: Initial value for the thermal energy simulating an Ideal Gas

i_kernelcorrection: Kernel correction: (0=none; 1=Kernel correction; 2=Gradient kernel Correction).

iRiemannSolver: Use of Riemann Solver: (0=None, 1=Conservative, 2=NonConservative

iTVD: Use TVD, slope limiter (beta_lim)? (1=yes)

beta_lim: slope limiter using Riemann Solver

i_vort: vorticity printing ? (1=yes)

ndt_VerletPerform: Number of time steps to apply the Eulerian equations with the Verlet algorithm

ndt_FilterPerform: Number of time steps to apply the density filter

ndt_DBCPerform: Number of time steps to apply the Hughes and Graham (2010) correction for dynamic boundary conditions.

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

In 3D version

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

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
nbf

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 plus the neighbours of 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).

Tsunami_Landslide.txt

Created by *SPHysicsgen*

To be used by *SPHysics* code. Parameters fix the wedge dimensions and movement.

UNIT=88

The file contains the following variables:

iopt_RaichlenWedge
bslope

Floating_Bodies.txt

Created by *SPHysicsgen*

To be used by *SPHysics* code. Parameters fix the floating bodies dimensions and movement.

UNIT=99

The file contains the following variables:

iopt_FloatingBodies
nbfm
num_FB
bigMass(num_FB)
bigInertiaXX(num_FB), bigInertiaYY(num_FB),bigInertiaZZ(num_FB)
XcylinderDimension(num_FB),YcylinderDimension(num_FB), ZcylinderDimension(num_FB)
cylinderDensity(num_FB)
FB_SpecificWeight(num_FB)
friction_coeff(num_FB)
Box_XC(num_FB),Box_YC(num_FB),Box_ZC(num_FB)
bigU(num_FB),bigV(num_FB),bigW(num_FB),
bigOmegaX(num_FB),bigOmegaY(num_FB),bigOmegaZ(num_FB)
nb_FB(num_FB)

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 a gate that can move in any direction.

TRAPEZOID Subroutine to generate a trapezoid inside the container.

RAICHLENWEDGE_PARTICLES Subroutine to generate a sliding wedge inside the beach.

FLOATINGBODY_PARTICLES Subroutine to generate floating bodies inside the container.

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.

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.

POSITION_CHECK Subroutine to ensure that particles are not too close to each other.

PRECISIONWRITE Subroutine to choose precision of SPHYSICS variables of position

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

TOCOMPILE_FTN95 Subroutine to create the MAKEFILE necessary to compile SPHysics using a Silverfrost FTN95 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.

VORT_klmn

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

UNIT=24

The following variables are recorded:

vortx_temp vorty_temp vortz_temp

(Note this is new for v2.0)

Description:

vortx_temp, vorty_temp, vortz_temp: correspond to vorticity in x,y and z constant planes

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.

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.

RESTART

Created by *SPHysics* at SPHYSICS_3D.f or SPHYSICS_2D.f

UNIT=44

The following variables are recorded:

itime time ngrab dt

Description:

itime: Number of iterations since the beginning of the run.
time: Time instant (in seconds).
ngrab: Recording instant.
dt: Time step

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, POUTE_3D.f, POUTE_CONSERVATIVE_2D.f and POUTE_CONSERVATIVE_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 DT, PART and VORT files previously described.

STEP (Source files: STEP_PREDICTOR_CORRECTOR_2D.f, STEP_PREDICTOR_CORRECTOR_3D.f, STEP_BEEMAN_2D.f, STEP_BEEMAN_3D.f, STEP_SYMPLECTIC_2D.f, STEP_SYMPLECTIC_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, Verlet, Symplectic or Beeman).

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.

DENSITYFILTER (SHEPARD) (Source file: DENSITYFILTER_SHEPARD_2D.f and DENSITYFILTER_SHEPARD_3D.f). Subroutine called from SPHysics every 30 time steps. It uses a Shepard filter when selected in initial conditions.

DENSITYFILTER (MLS) (Source file: DENSITYFILTER_MLS _2D.f and DENSITYFILTER_MLS _3D.f). Subroutine called from SPHysics every 30 time steps. It uses a MLS filter when selected in initial conditions.

AC_SHEPARD (Source files: AC_SHEPARD_2D.f and AC_ SHEPARD_3D.f). This subroutine is called from DENSITYFILTER_MLS. It calls the subroutines PRE_SELF_SHEPARD and PRE_CELIJ_SHEPARD.

AC_MLS (Source files: AC_MLS_2D.f and AC_MLS_3D.f). This subroutine is called from DENSITYFILTER_MLS. It calls the subroutines PRE_SELF_MLS and PRE_CELIJ_MLS.

AC_KC (Source files: AC_KC_2D.f and AC_KC_3D.f). This subroutine is called from STEP every time step. It calls the subroutines AC.

AC_KGC (Source files: AC_KGC_2D.f and AC_KGC_3D.f). This subroutine is called from STEP every time step. It calls the subroutines AC, PRE_SELF_KGC and PRE_CELIJ_KGC.

AC (Source files: AC_2D.f, AC_CONSEVATIVE_2D.f, AC_3D.f and AC_CONSEVATIVE_3D.f). This subroutine is called from AC_NONE or AC_KC or AC_KGC every time step. It controls the boundary particles movement (gates and wavemakers) and calls the subroutines SELF and CELIJ.

PRE_SELF_SHEPARD (Source files: PRE_SELF_SHEPARD_2D.f, PRE_SELF_SHEPARD_3D.f). This subroutine is called from AC_ SHEPARD.

PRE_CELIJ_SHEPARD (Source files: PRE_CELIJ_SHEPARD_2D.f, PRE_CELIJ_SHEPARD_3D.f). This subroutine is called from AC_ SHEPARD.

PRE_SELF_MLS (Source files: PRE_SELF_MLS_2D.f, PRE_SELF_MLS_3D.f). This subroutine is called from AC_MLS.

PRE_CELIJ_MLS (Source files: PRE_CELIJ_MLS_2D.f, PRE_CELIJ_MLS_3D.f). This subroutine is called from AC_MLS.

PRE_SELF_KGC (Source files: PRE_SELF_KGC_2D.f, PRE_SELF_KGC_3D.f). This subroutine is called from AC_KGC.

PRE_CELIJ_KGC (Source files: PRE_CELIJ_KGC_2D.f, PRE_CELIJ_KGC_3D.f). This subroutine is called from AC_KGC.

SELF (Source files: all SELF_*.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: all CELIJ_*.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_3D.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: all MONAGHANBC_*.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.

MOVINGGATE (Source file: MOVINGGATE_2D.f and MOVINGGATE _3D.f). This subroutine is called from MOVINGOBJECTS.

MOVINGPADDLE (Source file: MOVINGPADDLE_2D.f and MOVINGPADDLE _3D.f). This subroutine is called from MOVINGOBJECTS.

MOVINGWEDGE (Source file: MOVINGWEDGE_2D.f and MOVINGWEDGE _3D.f). This subroutine is called from MOVINGOBJECTS.

RIGID_BODY_MOTION (Source file: RIGID_BODY_MOTION_2D.f, RIGID_BODY_MOTION_3D.f, RIGID_BODY_MOTION_CONSERVATIVE_2D.f, RIGID_BODY_MOTION_CONSERVATIVE_3D.f). It describes the movement of the floating bodies. This subroutine is called from MOVINGOBJECTS.

LU_DECOMPOSITION (Source files: LU_DECOMPOSITION_2D.f, LU_DECOMPOSITION_3D.f,). This subroutine is called from DENSITYFILTER_MLS. It constructs the LU-decomposition matrix.

EOS_IDEALGAS (Source files: EOS_IDEALGAS_2D.f, EOS_IDEALGAS_3D.f,). It uses the equation of Ideal Gases to solve the pressure.

EOS_MORRIS (Source files: EOS_ MORRIS_2D.f, EOS_MORRIS_3D.f,). It uses the equation of Morris to solve the pressure.

EOS_TAIT (Source files: EOS_TAIT_2D.f, EOS_TAIT_3D.f,). It uses the equation of Tait to solve the pressure.

VORTICITY (Source files: VORTICITY_2D.f, VORTICITY_3D.f,). It calculates the vorticity terms. This subroutine is called from CELIJ and SELF.

UPDATENORMALS (Source file: UPDATENORMALS_2D.f and UPDATENORMALS_3D.f). It calculates the new normals of the moving boundary particles when Monaghan Boundary conditions are used. This subroutine is called from MOVINGPADDLE and RIGID_BODY_MOTION.

PERIODICITYCORRECTION (Source file: PERIODICITYCORRECTION_2D.f and PERIODICITYCORRECTION_3D.f). This subroutine corrects problems when periodicity domain is higher than dimension extent. It is called from all the PRE_CELIJ and CELIJ subroutines.

GRADIENTS_CALC (Source files: GRADIENTS_CALC_BASIC_2D.f, GRADIENTS_CALC_CONSERVATIVE_2D.f, GRADIENTS_CALC_BASIC_3D.f and GRADIENTS_CALC_CONSERVATIVE_3D.f). This subroutine is called from all the SELF and CELIJ subroutines.

APPROX_RIEMANNSOLVER (Source files: APPROX_RIEMANNSOLVER_CONSERVATIVE_2D.f, APPROX_RIEMANNSOLVER_NONCONSERVATIVE_2D.f, APPROX_RIEMANNSOLVER_CONSERVATIVE_3D.f, APPROX_RIEMANNSOLVER_NONCONSERVATIVE_3D.f,). This subroutine is called from all the SELF and CELIJ subroutines when Riemann Solver solution is selected.

LMITER (Source files: LIMITER_BETAMINMOD_2D.f, LIMITER_BETAMINMOD_3D.f). This subroutine is called from all the APPROX_RIEMANNSOLVER subroutines when Riemann Solver solution is selected.

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.

NOTE: the **default Compiler chosen is INTEL IFORT for WINDOWS**, which is option 3 near the end of each Case file.

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 can be downloaded from http://www.intel.com website.

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_unix_gfortran.bat* which is located in SPHysics_2D/run_directory/Case1. This batch file can be executed, while in the Case1 directory, by typing *Case1_unix_gfortran.bat* at the command prompt.

COMMAND	**COMMENTS**
cd ../../source/SPHYSICSgen2D/	Change to source directory in order to compile SPHysicsgen using SPHysicsgen.mak
make -f SPHYSICSgen_gfortran.mak clean	Remove any preexisting object files
make -f SPHYSICSgen_gfortran.mak	Compile and generate SPHysicsgen_2D using SPHysicsgen.make. This Makefile compiles and places the SPHysicsgen_2D executable in the execs directory and moves the older executable to the execs.bak directory.
cd ../../run_directory/Case1	Change to the Case1 example directory.
../../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 preexisting 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	Execute SPHysics_2D and direct the output from the run to sph.out

4.1.2. Compiling and executing on Windows.

In the SPHysics_2D/source/SPHysicsgen2D directory there are two Makefiles named SPHysicsgen_win_ifort.mak and SPHysicsgen_ftn95.mak. They are used to compile SPHysicsgen_2D using the INTEL IFORT compiler and Silverfrost FTN95 compiler (previously Salford Fortran).

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 and tocompile_ftn95, in SPHysicsgen_2D, write out SPHysics.mak for ifort and silverfrost ftn95 compilers respectively.

There are windows batch files located in the example directories, The batch file *Case1_windows_ifort.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_windows_ifort.bat* on a command window. The content of this file is briefly describe in next table.

COMMAND	**COMMENTS**
del *.exe	Remove previous executable files.
cd ..\..\source\SPHYSICSgen2D	Change to the directory containing the SPHysicsgen_2D source files

NMAKE/f"SPHYSICSgen_win_ifort.mak"	NMAKE /f "SPHysicsgen.mak" is used to compile SPHysicsgen_2D.exe.
cd ..\..\ run_directory\Case1	Change directory
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. 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. 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
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. 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_windows_ifort.bat, Case1_windows_ftn95.bat, Case1_unix_gfortran.bat or Case1_unix_ifort.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:

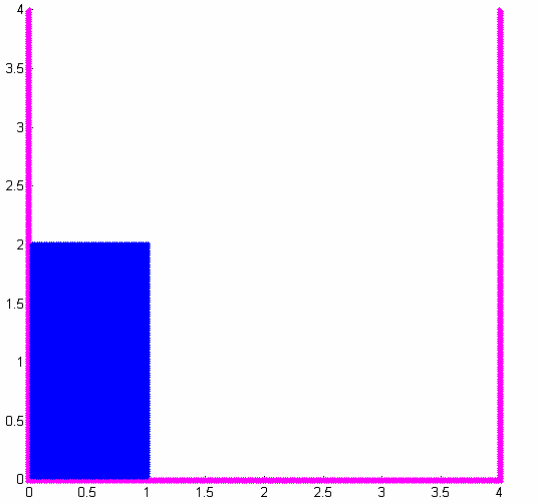


Figure 4.1: Initial configuration of Case1.

Input data	Variable description
0	Choose Starting options: 0=new, 1=restart, 2=new with CheckPointg, 3=restart with CheckPointing
5	Kernel: 1=gaussian, 2=quadratic; 3=cubic; 5=Wendland
1	Time-stepping algorithm: 1=Predictor-corrector, 2=Verlet, 3=Symplectic, 4=Beeman
2	Density Filter: 0=none, 1=Shepard filter, 2=MLS
30	ndt_FilterPerform (if density filter is used) ?
0	Kernel correction 0=None, 1=Kernel correction, 2=Gradient kernel Correction
1	Viscosity treatment: 1=artificial; 2=laminar; 3=laminar + SPS
0.3	Viscosity value(if visc.treatment=1 it's alpha, if not kinem. visc approx 1.e-6)
0	vorticity printing ? (1=yes)
1	Equation of State: 1=Tait's equation, 2=Ideal Gas, 3= Morris
2	Maximum Depth (h_SWL) to calculate B
10	Coefficient of speed of sound (recommended 10 - 40) ??
2	Boundary Conditions: 1=Repulsive Force; 2=Dalrymple
15	ndt_DBCPerform ? (1 means no correction)
1	Geometry of the zone: 1=BOX, 2=BEACH, 3=COMPLEX GEOMETRY
2	Initial Fluid Particle Structure: 1= SC, 2= BCC
4.,4.	Box dimension LX,LZ?
0.03,0.03	Spacing dx,dz?

0	Inclination of floor in X (beta) ??
0,0,0	Periodic Lateral boundaries in X, Y, & Z-Directions ? (1=yes)
0	Add wall
0	Add obstacle (1=y)
0	Add wavemaker (1=y)
0	Add gate (1=y)
0	Add Floating Body (1=yes)
2	Initial conditions: 2) particles on a staggered grid
0	Correct pressure at boundaries ?? (1=y)
0.03,1.	Cube containing particles : XMin, Xmax ??
0.03,2.	Cube containing particles : ZMin, Zmax ??
0	Fill a new region
3,0.02	Input the tmax and out
0.	initial time of outputting general data
0.0005,1.0,-1.0	For detailed recording during RUN: out_detail, start, end
0.0001,1	Input dt?? , i_var_dt ??
0.2	CFL number (0.1-0.5)
0.92	h=coefficient*sqrt(dx*dx+dz*dz): coefficient ???
0	Use of Riemann Solver: 0=None, 1=Conservative (Vila), 2=NonConservative (Parshikov)
3	Which compiler is desired: 1=gfortran, 2=ifort, 3=win_ifort, 4=Silverfrost FTN95
1	Precision of XYZ Variables: 1=Single, 2=Double

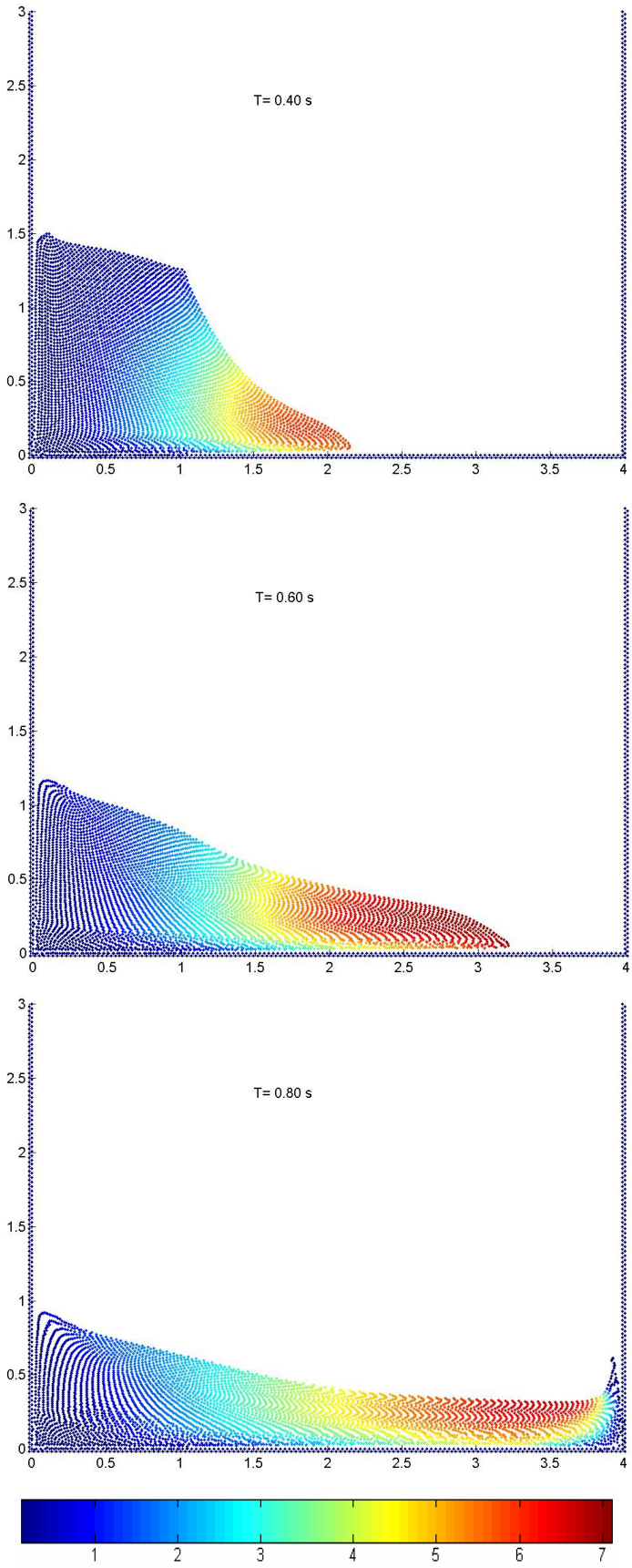


Figure 4.2: X-Velocity plot in Case1.

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:

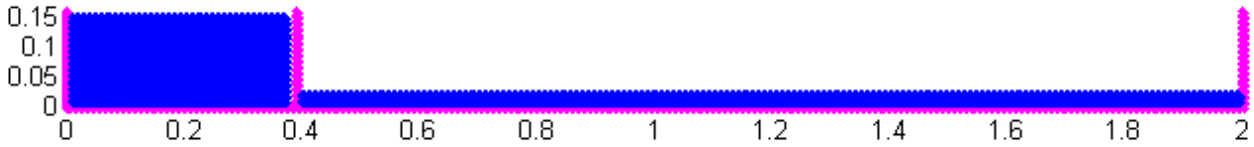


Figure 4.3: Initial configuration of Case2

Input data	Variable description
0	Choose Starting options: 0=new, 1=restart, 2=new with CheckPointg, 3=restart with CheckPointing
3	Kernel: 1=gaussian, 2=quadratic; 3=cubic; 5=Wendland
1	Time-stepping algorithm: 1=Predictor-corrector, 2=Verlet, 3=Symplectic, 4=Beeman
0	Density Filter: 0=none, 1=Shepard filter, 2=MLS
0	Kernel correction 0=None, 1=Kernel correction, 2=Gradient kernel Correction
1	Viscosity treatment: 1=artificial; 2=laminar; 3=laminar + SPS
0.08	Viscosity value(if visc.treatment=1 it's alpha, if not kinem. visc approx 1.e-6)
0	Vorticity printing ? (1=yes)
1	Equation of State: 1=Tait's equation, 2=Ideal Gas, 3= Morris
0.15	Maximum Depth (h_SWL) to calculate B
13	Coefficient of speed of sound (recommended 10 - 40) ??
2	Boundary Conditions: 1=Repulsive Force; 2=Dalrymple
1	ndt_DBCPerform ? (1 means no correction)
1	Geometry of the zone: 1=BOX, 2=BEACH, 3=COMPLEX GEOMETRY
2	Initial Fluid Particle Structure: 1= SC, 2= BCC
2,0.16	Box dimension LX,LZ?
0.005,0.005	Spacing dx,dz?
0	Inclination of floor in X (beta) ??
0,0,0	Periodic Lateral boundaries in X, Y, & Z-Directions ? (1=yes)
0	Add wall
0	Add an obstacle (1=y)
0	Add wavemaker (1=y)
1	Add gate (1=y)
0.388	Gate position in X coordinates ??'
0.,0.16	Gate height ??

0.,1.5	Gate Velocity ??
0	t gate??
0	Add new gate (1=y)
0	Add Floating Bodies (1=yes) ?
2	Initial conditions: 2) particles on a staggered grid
0	Correct pressure at boundaries ?? (1=y)
0.005,0.376	Cube containing particles : XMin, Xmax ??
0.005,0.15	Cube containing particles : ZMin, Zmax ??
1	Fill a new region
0.40,1.995	Cube containing particles : XMin, Xmax ??
0.005,0.018	Cube containing particles : ZMin, Zmax ??
0	Fill a new region
1.2,0.01	Input the tmax and out
0.0	Initial time of outputting general data
0.0005,1.0,-1.0	For detailed recording during RUN: out_detail, start, end
0.0001,1	Input dt, i_var_dt ??
0.2	CFL number (0.1-0.5)
0.92	h=coefficient*sqrt(dx*dx+dz*dz): coefficient ???
0	Use of Riemann Solver: 0=None, 1=Conservative (Vila), 2=NonConservative (Parshikov)
3	Which compiler is desired: 1=gfortran, 2=ifort, 3=win_ifort, 4=Silverfrost FTN95
1	Precision of XYZ Variables: 1=Single, 2=Double

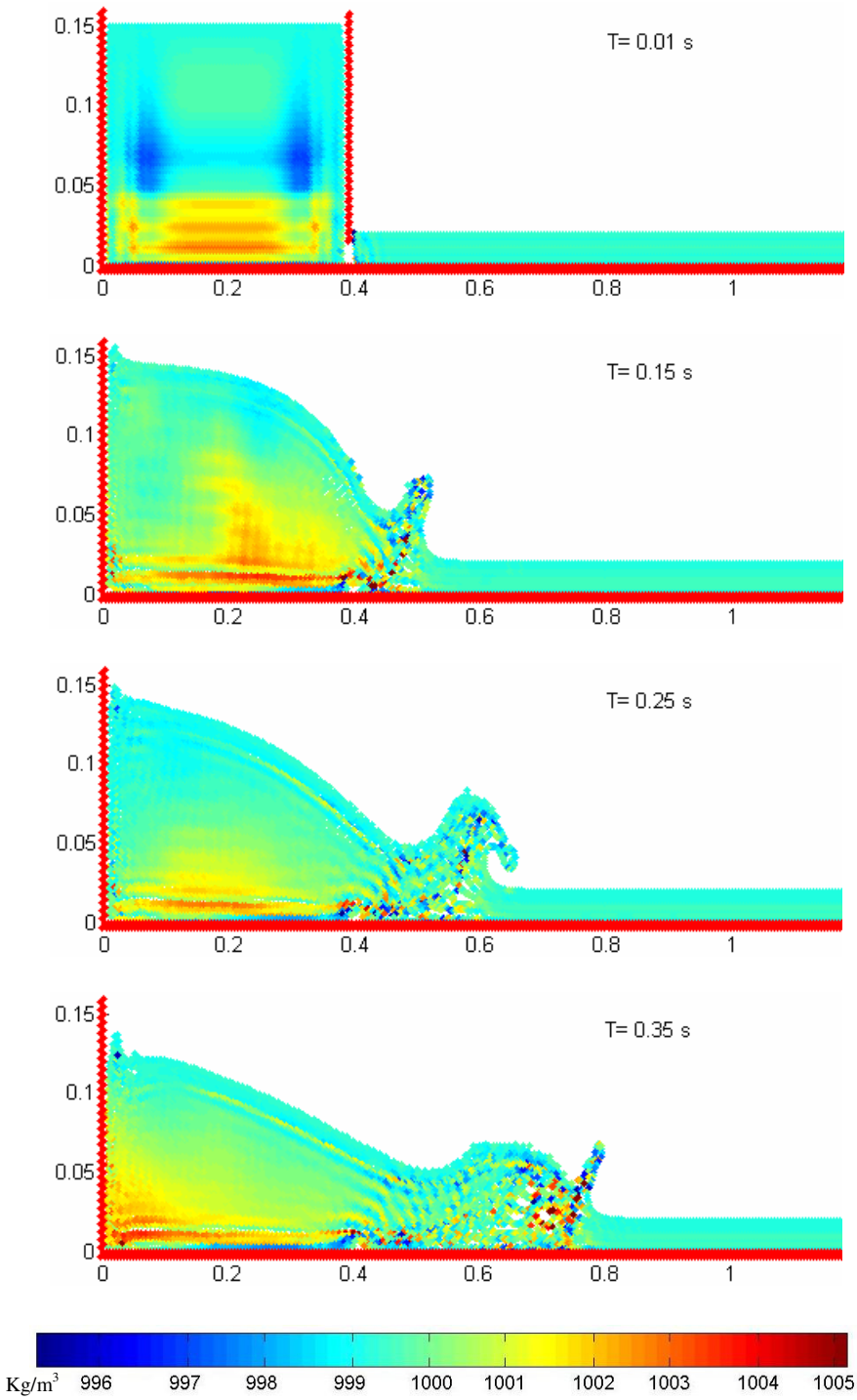


Figure 4.4: Density plot in Case2

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

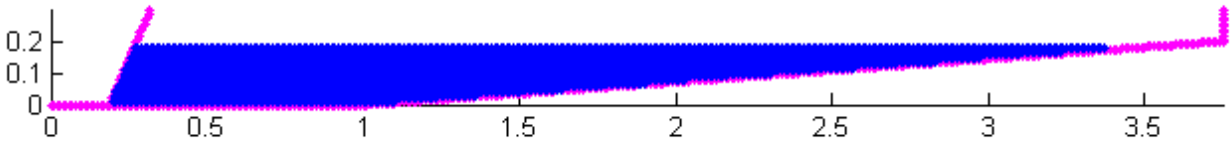


Figure 4.5: Initial configuration of Case3 in 2D.

Input data	Variable description
0	Choose Starting options: 0=new, 1=restart, 2=new with CheckPointg, 3=restart with CheckPointing
3	Kernel: 1=gaussian, 2=quadratic; 3=cubic; 5=Wendland
3	Time-stepping algorithm: 1=Predictor-corrector, 2=Verlet, 3=Symplectic, 4=Beeman
1	Density Filter: 0=none, 1=Shepard filter, 2=MLS
30	ndt_FilterPerform (if density filter is used) ?
0	Kernel correction 0=None, 1=Kernel correction, 2=Gradient kernel Correction
2	Viscosity treatment: 1=artificial; 2=laminar; 3=laminar + SPS
1.0e-6	Viscosity value(if visc.treatment=1 it's alpha, if not kinem. visc approx 1.e-6)
0	Vorticity printing ? (1=yes)
1	Equation of State: 1=Tait's equation, 2=Ideal Gas, 3= Morris
0.2	Maximum Depth (h_SWL) to calculate B
16	Coefficient of speed of sound (recommended 10 - 40) ??
1	Boundary Conditions: 1=Repulsive Force; 2=Dalrymple
1.0e-5	Wall viscosity value for Repulsive Force BC
2	Geometry of the zone: 1=BOX, 2=BEACH, 3=COMPLEX GEOMETRY
1	Initial Fluid Particle Structure: 1= SC, 2= BCC
3.75,0.3	Box dimension LX, LZ?
0.01,0.01	Spacing dx, dz?
1.0	Length of Flat Domain
4.2364	Slope (deg) of the inclined plane (beta) ??
0,0,0	Periodic Lateral boundaries in X, Y, & Z-Directions ? (1=yes)
1	If wavemaker will be added, left pannel is not needed (1=yes)
0	Add obstacle (1=yes)

2	Enter Paddle-Type: 1=Piston, 2=Piston-flap, 3=Piston with prescribed motion
0.13	X_PaddleCentre
0.15	paddle_SWL
0.1344	flap_length = distance of pivot point under bed
0.0,0.3	ZZMin, ZZmax of the wavemaker ??
0.0	Initial time of wavemaker = twavemaker ??
1	Number of frequencies ??
0.2422	Wavemaker Stroke = 2*Amplitude ??
1.4	Period ??
0	Phase ??
0	twinitial ??
0	Add another wavemaker inside the beach (1=yes)
0	Add gate (1=yes)
0	Add Sliding Raichlen Wedge (1=yes)
0	Add Floating Body (1=yes)
1	Add water in the flat region ?? (1=yes)
0, 1.0	Cube containing particles : XMin, Xmax ??
0.025, 0.18	Cube containing particles : ZMin, Zmax ??
1	Add water in the inclined region ?? (1=yes)
1.01, 3.75	Cube containing particles : XMin, Xmax ??
0.025, 0.18	Cube containing particles : ZMin, Zmax ??
0	Add a solitary wave ?? (1=yes)
5.0,0.05	Input the tmax and out
0.0	initial time of recording
0.0,1.0,-1.0	detailed recording: out_dtrecording, Start time, End Time
0.000045,1	input dt ??, variable dt ??
0.2	CFL number (0.1-0.5)
0.92	h=coefficient*sqrt(dx*dx+dy*dy+dz*dz): coefficient ???
0	Use of Riemann Solver: 0=None, 1=Conservative (Vila), 2=NonConservative (Parshikov)
3	Which compiler is desired: 1=gfortran, 2=ifort, 3=win_ifort, 4=Silverfrost FTN95
2	Precision of XYZ Variables: 1=Single, 2=Double

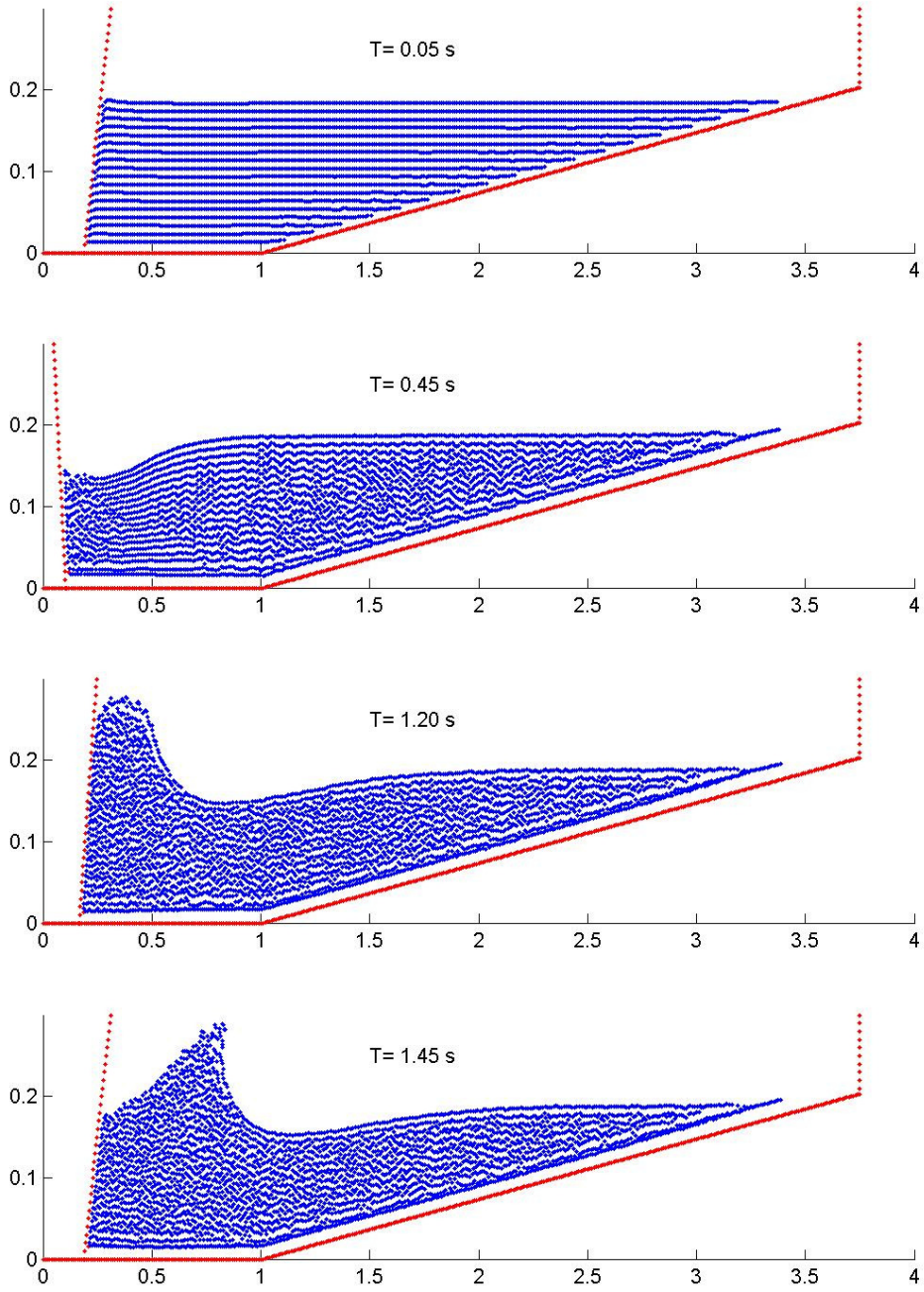


Figure 4.6: Wave formation in Case3 (for 2D).

4.4.2 Case 3D

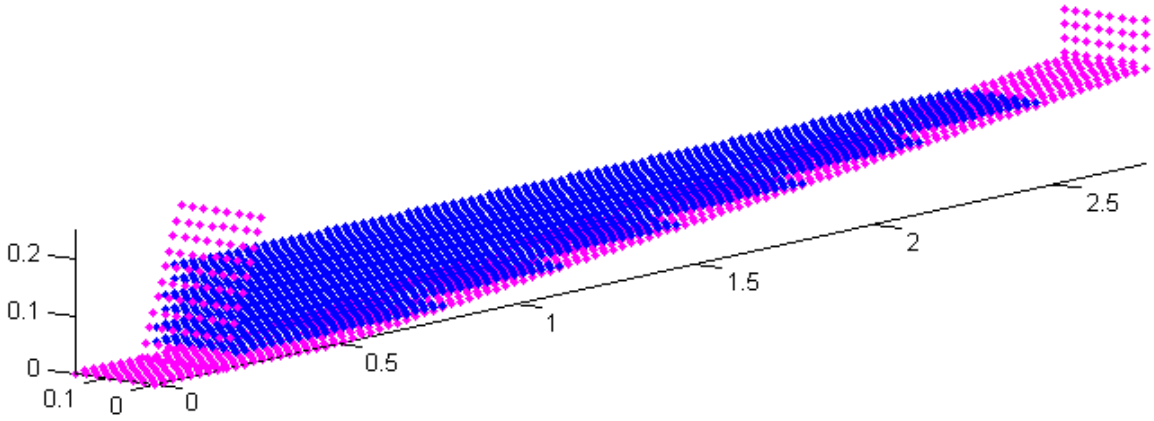


Figure 4.7: Initial configuration of Case3 in 3D.

Input data	Variable description
0	Choose Starting options: 0=new, 1=restart, 2=new with CheckPointg, 3=restart with CheckPointing
2	Kernel: 1=gaussian, 2=quadratic; 3=cubic; 5=Wendland
1	Time-stepping algorithm: 1=predictor-corrector, 2=verlet, 3=symplectic, 4=Beeman
1	Density Filter: 0=none, 1=Shepard filter, 2=MLS
30	ndt_FilterPerform (if density filter is used) ?
0	Kernel correction 0=None, 1=Kernel correction, 2=Gradient kernel Correction
3	Viscosity treatment 1=artificial; 2=laminar; 3=laminar + SPS
1.0e-6	Viscosity value(if visc.treatment=1 it's alpha, if not kinem. visc approx 1.e-6)
0	Vorticity printing ? (1=yes)
1	Equation of State: 1=Tait's equation, 2=Ideal Gas, 3= Morris
0.15	Maximum Depth (Hmax) to calculate B
16	coefficient (10 , 40) ??
1	Boundary Conditions: Monaghan = 1 or Dalrymple = 2
8.0e-1	Wall viscosity value for Repulsive Force BC
2	Geometry of the zone: 1=BOX, 2=BEACH, 3=COMPLEX GEOMETRY
2	Initial Fluid Particle Structure: 1= SC, 2= BCC
2.75,0.20,0.25	Box dimension LX, LY, LZ?
0.02,0.02,0.02	Spacing dx, dy, dz?
0.5	Length of Flat Domain
4.2364	Slope (deg) of the inclined plane (beta) ??
0,1,0	Periodic Lateral boundaries in X, Y, & Z-Directions ? (1=yes)
1	If wavemaker will be added, left pannel is not needed (1=yes)
0	Add obstacle (1=yes)
2	Enter Paddle-Type: 1=Piston, 2=Piston-flap, 3=Piston with prescribed motion

0.20	X_PaddleCentre
0.15	paddle_SWL
0.1344	flap_length = distance of pivot point under bed
0.0,0.2	YYMin, YYmax of the wavemaker ??
0.0,0.25	ZZMin, ZZmax of the wavemaker ??
0.0	Initial time of wavemaker = twavemaker ??
1	Number of frequencies ??
0.2442	Wavemaker Stroke = 2*Amplitude ??
1.4	Period ??
0	Phase ??
0	twinitial ??
0	Add another wavemaker inside the beach (1=yes)
0	Add gate (1=yes)
0	Add Sliding Raichlen Wedge (1=yes)
0	Add Floating Body (1=yes,0=no)
1	Add water in the flat region ?? (1=yes)
0, 0.49	Cube containing particles : XMin, Xmax ??
0.00, 0.20	Cube containing particles : YMin, Ymax ??
0.02, 0.15	Cube containing particles : ZMin, Zmax ??
1	Add water in the inclined region ?? (1=yes)
0.50, 2.5	Cube containing particles : XMin, Xmax ??
0.00, 0.20	Cube containing particles : YMin, Ymax ??
0.02, 0.15	Cube containing particles : ZMin, Zmax ??
0	Add a solitary wave ?? (1=yes)
5.0,0.025	Input the tmax and out
0.0	initial time of recording
0,1,-1	detailed recording: out_dtrecording, Start time, End Time
0.00005,0	input dt ??, variable dt ??
0.2	CFL number (0.1 - 0.5)
0.866025	h=coefficient*sqrt(dx*dx+dy*dy+dz*dz): coefficient ???
0	Use of Riemann Solver: 0=None, 1=Conservative (Vila), 2=NonConservative (Parshikov)
3	Which compiler is desired: 1=gfortran, 2=ifort, 3=win_ifort, 4=Silverfrost FTN95
2	Precision of XYZ Variables: 1=Single, 2=Double

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

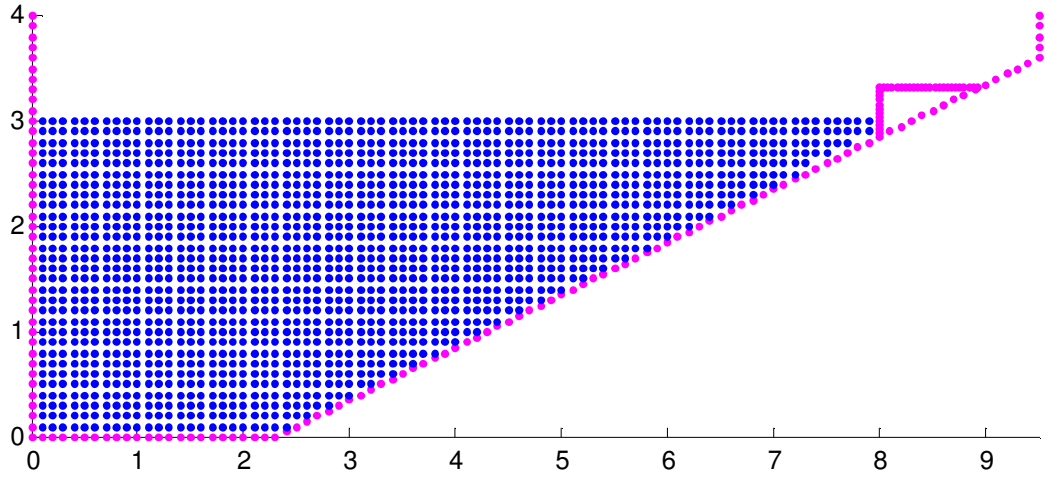


Figure 4.8: Initial configuration of Case4 in 2D.

Input data	Variable description
0	Choose Starting options: 0=new, 1=restart, 2=new with CheckPointg, 3=restart with CheckPointing
2	Kernel: 1=gaussian, 2=quadratic; 3=cubic; 5=Wendland
1	Time-stepping algorithm: 1=predictor-corrector, 2=Verlet, 3=Symplectic, 4=Beeman
1	Density Filter: 0=none, 1=Shepard filter, 2=MLS
30	ndt_FilterPerform (if density filter is used) ?
0	Kernel correction 0=None, 1=Kernel correction, 2=Gradient kernel Correction
3	Viscosity treatment: 1=artificial; 2=laminar; 3=laminar + SPS
1.0e-6	Viscosity value(if visc.treatment=1 it's alpha, if not kinem. visc approx 1.e-6)
0	Vorticity printing ? (1=yes)
1	Equation of State: 1=Tait's equation, 2=Ideal Gas, 3= Morris
3.0	Maximum Depth (h_SWL) to calculate B
16	Coefficient of speed of sound (recommended 10 - 40) ??
1	Boundary Conditions: 1=Repulsive Force; 2=Dalrymple
2.0e-4	Wall viscosity value for Repulsive Force BC
2	Geometry of the zone: 1=BOX, 2=BEACH, 3=COMPLEX GEOMETRY

1	Initial Fluid Particle Structure: 1= SC, 2= BCC
9.5,4.0	Box dimension LX, LZ?
0.05,0.05	Spacing dx, dz?
2.25	Length of Flat Domain
26.565051	Slope (deg) of the inclined plane (beta) ??
0,0,0	Periodic Lateral boundaries in X, Y, & Z-Directions ? (1=yes)
0	If wavemaker will be added, left pannel is not needed (1=yes)
0	Add obstacle (1=yes)
0	Add gate (1=yes)
1	Add Raichlen Wedge (1=yes)
0.3	Enter block-top elevation above SWL
2.14	Enter specific Weight
0.91, 0.455, 0.61	Enter block_length, block_height, block_width
0.04, 0.04	Enter block dxW, dzW
0	Add Floating Body (1=yes)
1	Add water in the flat region ?? (1=yes)
0.05, 2.25	Cube containing particles : XMin, Xmax ??
0.05, 3.5	Cube containing particles : ZMin, Zmax ??
1	Add water in the inclined region ?? (1=yes)
2.30, 9.5	Cube containing particles : XMin, Xmax ??
0.05, 3.5	Cube containing particles : ZMin, Zmax ??
0	Add a solitary wave ?? (1=yes)
3.0,0.015	Input the tmax and out
0.0	initial time of outputting general data
0,1.0,-1.0	For detailed recording: out_dtrecording, Start time, End Time
0.00011,1	input dt ??, variable dt ??
0.2	CFL number (0.1 - 0.5)
0.92	h=coefficient*sqrt(dx*dx+dy*dy+dz*dz): coefficient ???
0	Use of Riemann Solver: 0=None, 1=Conservative (Vila), 2=NonConservative (Parshikov)
3	Which compiler is desired: 1=gfortran, 2=ifort, 3=win_ifort, 4=Silverfrost FTN95
2	Precision of XYZ Variables: 1=Single, 2=Double

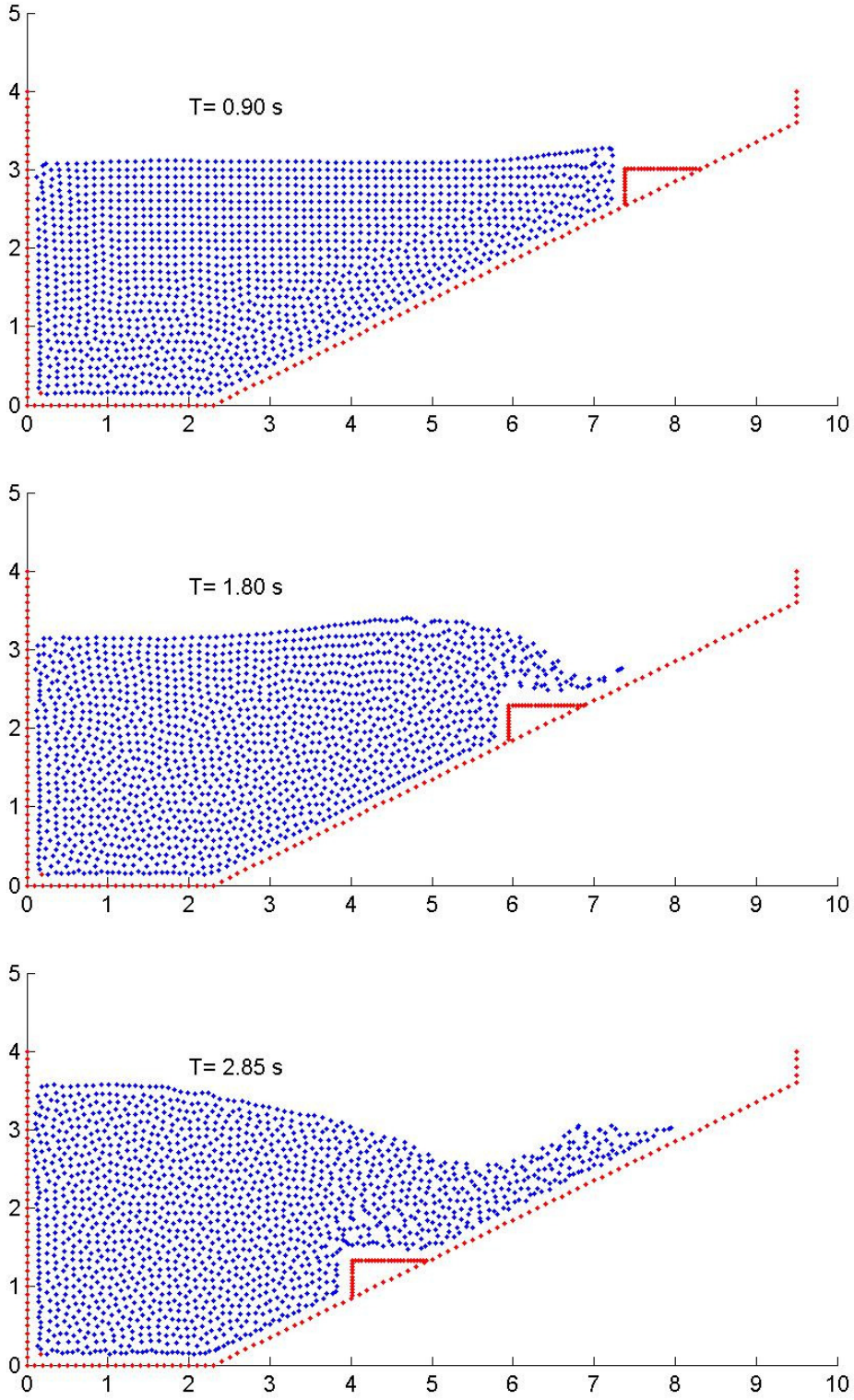


Figure 4.9: Tsunami generation using sliding Wedge (for 2D).

4.5.2. Case 3D

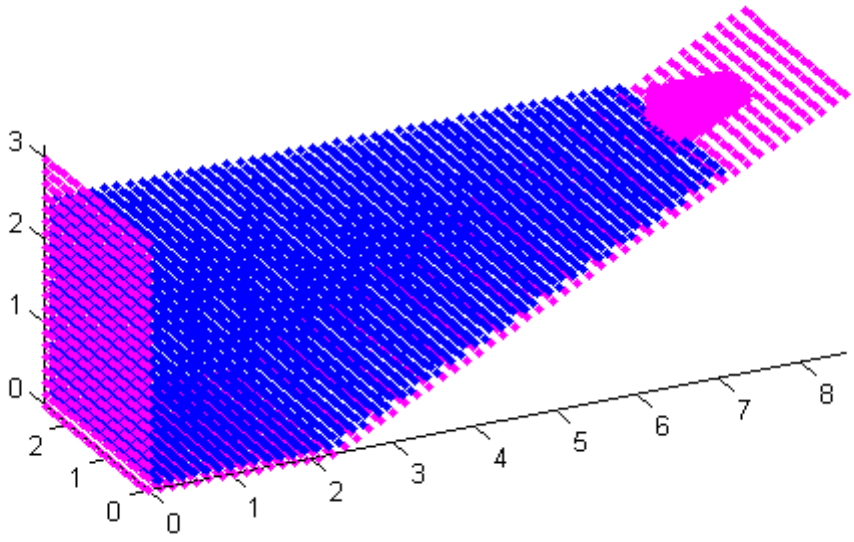


Figure 4.10: Initial configuration of Case4 in 3D.

Input data	Variable description
0	Choose Starting options: 0=new, 1=restart, 2=new with CheckPointg, 3=restart with CheckPointing
2	Kernel: 1=gaussian, 2=quadratic; 3=cubic; 5=Wendland
1	Time-stepping algorithm: 1=Predictor-corrector, 2=Verlet, 3=Symplectic, 4=Beeman
1	Density Filter: 0=none, 1=Shepard filter, 2=MLS
30	ndt_FilterPerform (if density filter is used) ?
0	Kernel correction 0=None, 1=Kernel correction, 2=Gradient kernel Correction
3	Viscosity treatment: 1=artificial; 2=laminar; 3=laminar + SPS
1.0e-6	Viscosity value(if visc.treatment=1 it's alpha, if not kinem. visc approx 1.e-6)
0	vorticity printing ? (1=yes)
1	Equation of State: 1=Tait's equation, 2=Ideal Gas, 3= Morris
2.5	Maximum Depth (h_SWL) to calculate B
16	Coefficient of speed of sound (recommended 10 - 40) ??
1	Boundary Conditions: 1=Repulsive Force; 2=Dalrymple
8.0e-6	Wall viscosity value for Repulsive Force BC
2	Geometry of the zone: 1=BOX, 2=BEACH, 3=COMPLEX GEOMETRY
2	Initial Fluid Particle Structure: 1= SC, 2= BCC
8.5,2.7,3.0	Box dimension LX, LY, LZ?
0.15,0.15,0.15	Spacing dx, dy, dz?
2.25	Length of Flat Domain
26.565051	Slope (deg) of the inclined plane (beta) ??

0,1,0	Periodic Lateral boundaries in X, Y, & Z-Directions ? (1=yes)
0	If wavemaker will be added, left pannel is not needed (1=yes)
0	Add obstacle (1=yes)
0	Add gate (1=yes)
1	Add Raichlen Wedge (1=yes)
0.3	Enter block-top elevation above SWL
2.14	Enter specific Weight
0.91, 0.455, 0.61	Enter block_length, block_height, block_width
0.04, 0.04, 0.04	Enter block dxW, dyW, dzW
0	Add Floating Body (1=yes)
1	Add water in the flat region ?? (1=yes)
0.075, 2.25	Cube containing particles : XMin, Xmax ??
0.0751, 2.651	Cube containing particles : YMin, Ymax ??
0.075, 2.5	Cube containing particles : ZMin, Zmax ??
1	Add water in the inclined region ?? (1=yes)
2.3249999, 8.5	Cube containing particles : XMin, Xmax ??
0.0751, 2.651	Cube containing particles : YMin, Ymax ??
0.075, 2.5	Cube containing particles : ZMin, Zmax ??
0	Add a solitary wave ?? (1=yes)
3.0,0.015	Input the tmax and out
0.0	initial time of recording
0,1,-1	detailed recording: out_dtrecording, Start time, End Time
0.0001729,1	input dt ??, variable dt ??
0.2	CFL number (0.1 - 0.5)
0.866025	h=coefficient*sqrt(dx*dx+dy*dy+dz*dz): coefficient ???
0	Use of Riemann Solver: 0=None, 1=Conservative (Vila), 2=NonConservative (Parshikov)
3	Which compiler is desired: 1=gfortran, 2=ifort, 3=win_ifort, 4=Silverfrost FTN95
2	Precision of XYZ Variables: 1=Single, 2=Double

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:

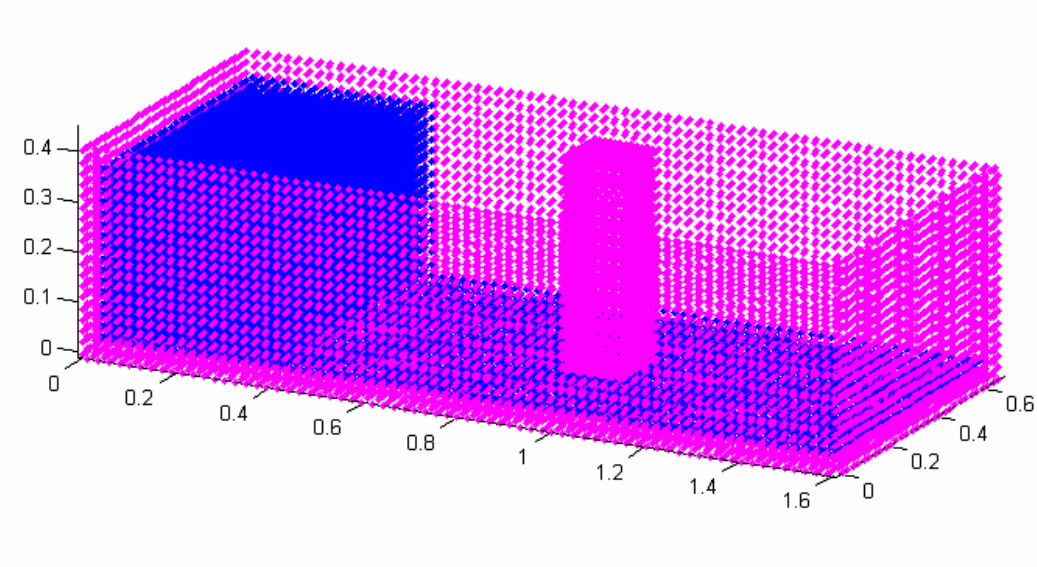


Figure 4.11: Initial configuration of Case5.

Input data	Variable description
0	Choose Starting options: 0=new, 1=restart, 2=new with CheckPointg, 3=restart with CheckPointing
3	Kernel: 1=gaussian, 2=quadratic; 3=cubic; 5=Wendland
1	Time-stepping algorithm: 1=predictor-corrector, 2=verlet, 3=symplectic, 4=Beeman
2	Density Filter: 0=none, 1=Shepard filter, 2=MLS
30	ndt_FilterPerform
0	Kernel correction 0=None, 1=Kernel correction, 2=Gradient kernel Correction
1	Viscosity treatment 1=artificial; 2=laminar; 3=laminar + SPS
0.1	Viscosity value(if visc.treatment=1 it's alpha, if not kinem. visc approx 1.e-6)
0	Vorticity printing ? (1=yes)
1	Equation of State: 1=Tait's equation, 2=Ideal Gas, 3= Morris
0.30	Maximum Depth (h_SWL) to calculate B
10	coefficient (10 , 40) ??
2	Boundary Conditions: 1=Repulsive Force; 2=Dalrymple
1	ndt_DBCPerform ? (1 means no correction)
1	Geometry of the zone: 1=BOX, 2=BEACH, 3=COMPLEX GEOMETRY
2	Initial Fluid Particle Structure: 1= SC, 2= BCC
1.6,0.67,0.4	Box dimension LX,LY,LZ?
0.0225,0.0225,0.0225	Spacing dx,dy,dz?

0	inclination of floor in X (beta) ??
0	inclination of floor in Y (tita) ??
0,0,0	Periodic Lateral boundaries in X, Y, & Z-Directions ? (1=yes)
0	If wavemaker will be added, left wall is not needed
0	Add wall (1=y)
0	Add wall with slot (1=y)
0	Add wall with round hole (1=y)
1	Add obstacle (1=y)
1	Choose obstacle: 1=rectangular 2=trapezoid
2	Kind of obstacle: 1=Solid, 2=Solid Walls
2	Density of points(ndens): dxi_new=dxi_old/ndens (ndens >1 increases density)
0.9,1.02	XMin, Xmax ??
0.24,0.36	YMin, Ymax ??
0.,0.45	ZMin, Zmax ??
90	slope
0	Add new obstacle (1=y)
0	Add wavemaker
0	Add gate
0	Add Floating Bodies (1=yes) ?
2	Initial conditions: 2) particles on a staggered grid without filling the box
0	Correct pressure at boundaries ?? (1=y)
0.0225,0.4	XMin, Xmax ??
0.0225,0.6475	YMin, Ymax ??
0.0225,0.36	ZMin, Zmax ??
1	Fill a new region
0.4225,0.8775	XMin, Xmax ??
0.0225,0.6475	YMin, Ymax ??
0.0225,0.03	ZMin, Zmax ??
1	Fill a new region
0.9,1.025	XMin, Xmax ??
0.0225,0.2175	YMin, Ymax ??
0.0225,0.03	ZMin, Zmax ??
1	Fill a new region
0.9,1.025	XMin, Xmax ??
0.3825,0.6475	YMin, Ymax ??
0.0225,0.03	ZMin, Zmax ??
1	Fill a new region
1.0425,1.5775	XMin, Xmax ??
0.0225,0.6475	YMin, Ymax ??
0.0225,0.03	ZMin, Zmax ??
0	Fill a new region

2,0.01	Input the tmax and out
0.	initial time of outputting general data
0.0005,1.0,-1.0	For detailed recording during RUN: out_detail, start, end
0.00005,1	input dt ??, variable dt ??
0.2	CFL number (0.1-0.5)
0.866025	h=coefficient*sqrt(dx*dx+dy*dy+dz*dz): coefficient ???
0	Use of Riemann Solver: 0=None, 1=Conservative (Vila), 2=NonConservative (Parshikov)
3	Which compiler is desired: 1=gfortran, 2=ifort, 3=win_ifort, 4=Silverfrost FTN95
2	Precision of XYZ Variables: 1=Single, 2=Double

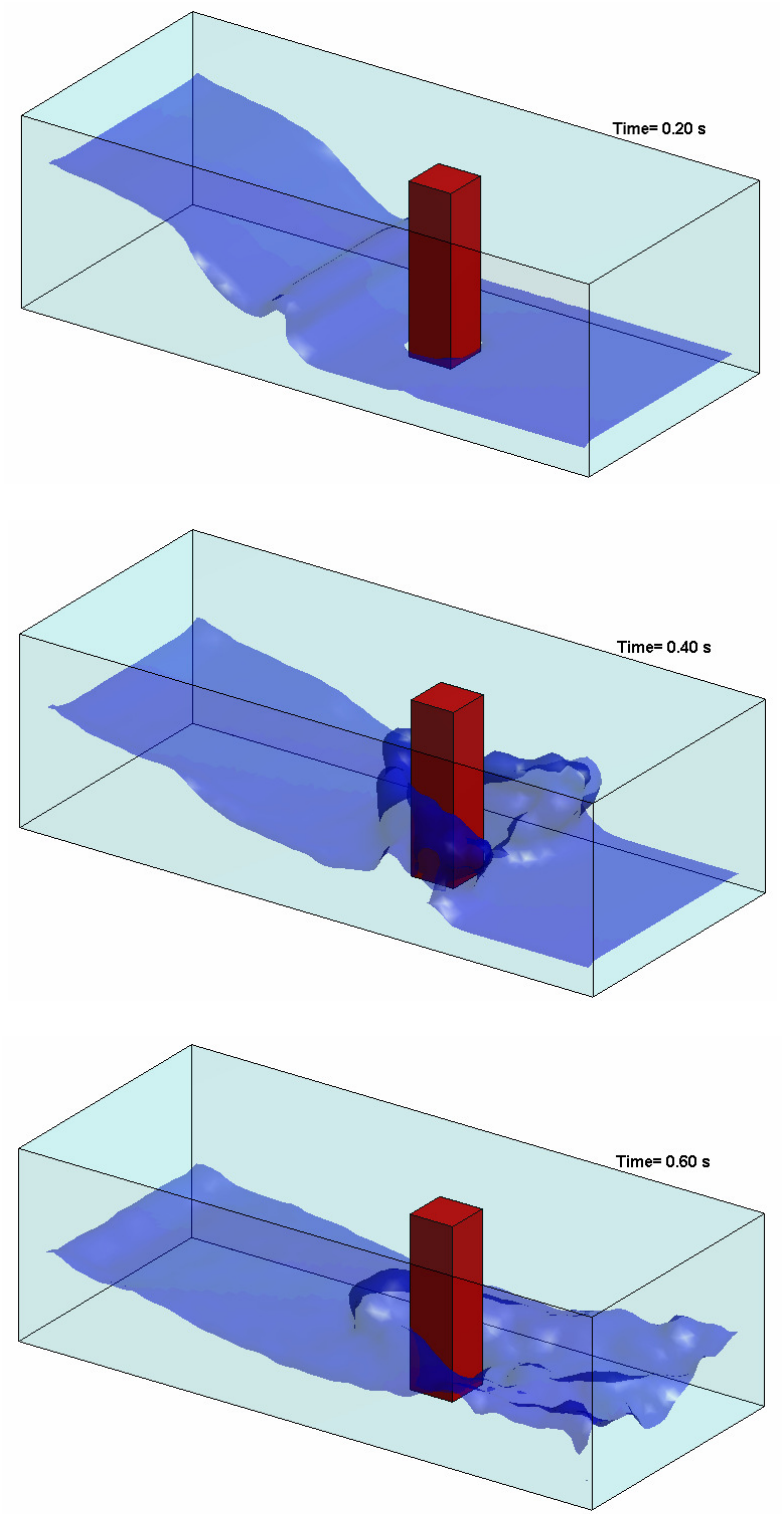


Figure 4.12: Interaction wave-structure in Case5

4.7. Test case 6: Floating bodies in waves

The case can be run using *Case6.bat* whose output directory is *Case6*. The input file *Case6.txt* is located in the output directory. The information contained in that file can be summarized as follows:

4.7.1 Case 2D

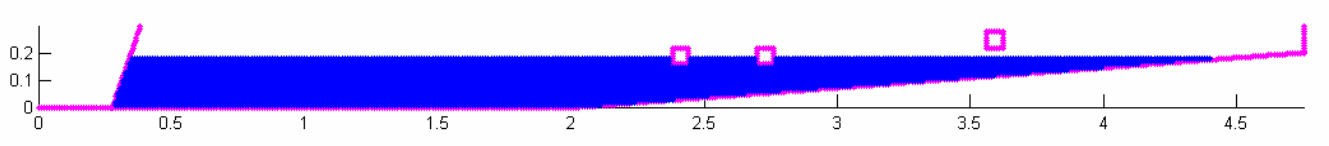


Figure 4.13: Initial configuration of Case6 in 2D.

Input data	Variable description
0	Choose Starting options: 0=new, 1=restart, 2=new with CheckPointg, 3=restart with CheckPointing
2	Kernel: 1=gaussian, 2=quadratic; 3=cubic; 5=Wendland
3	Time-stepping algorithm (1=Predictor-corrector, 2=Verlet, 3=Symplectic, 4=Beeman)
0	Density Filter: 0=none, 1=Shepard filter, 2=MLS
0	Kernel correction 0=None, 1=Kernel correction, 2=Gradient kernel Correction
2	Viscosity treatment: 1=artificial; 2=laminar; 3=laminar + SPS
1.0e-6	Viscosity value(if visc.treatment=1 it's alpha, if not kinem. visc approx 1.e-6)
0	Vorticity printing ? (1=yes)
1	Equation of State: 1=Tait's equation, 2=Ideal Gas, 3= Morris
0.18	Maximum Depth (h_SWL) to calculate B
16	Coefficient of speed of sound (recommended 10 - 40) ??
1	Boundary Conditions: 1=Repulsive Force; 2=Dalrymple
0.0e-3	Wall viscosity value for Repulsive Force BC
2	Geometry of the zone: 1=BOX, 2=BEACH, 3=COMPLEX GEOMETRY
1	Initial Fluid Particle Structure: 1= SC, 2= BCC
4.75,0.3	Box dimension LX, LZ?
0.01,0.01	Spacing dx, dz?
2.0	Length of Flat Domain
4.2364	Slope (deg) of the inclined plane (beta) ??
0,0,0	Periodic Lateral boundaries in X, Y, & Z-Directions ? (1=yes)
1	If wavemaker will be added, left pannel is not needed (1=yes)
0	Add obstacle (1=yes)
2	Enter Paddle-Type: 1=Piston, 2=Piston-flap, 3=Piston with prescribed motion

0.23	X_PaddleCentre
0.15	paddle_SWL
0.1344	flap_length = distance of pivot point under bed
0.0,0.3	ZZMin, ZZmax of the wavemaker ??
0.0	Initial time of wavemaker = twavemaker ??
1	Number of frequencies ??
0.2	Wavemaker Stroke = 2*Amplitude ??
1.4	Period ??
0	Phase ??
0	twinitial ??
0	Add another wavemaker inside the beach (1=yes)
0	Add gate (1=yes)
0	Add Sliding Raichlen Wedge (1=yes)
1	Add Floating Body (1=yes)
0.06, 0.06	Enter square cylinderDimension
1.0	Enter specific Weight
2.38, 0.16	Enter x,z of Bottom Left Corner (x_BottomLeft, z_BottomLeft)
0.0, 0.0	Enter (x,z) shift of Centre of Gravity for use in Parallel Axis Theorem
0.0, 0.0	Enter initial U,W velocity of Object
0.0, 0.0	Enter initial Body Angle and Rotation Rate (Omega) - Positive anticlockwise
0.20	Enter coefficient of Friction
1	Add another Floating Body (1=yes)
0.06, 0.06	Enter square cylinderDimension
1.0	Enter specific Weight
2.70, 0.16	Enter x,z of Bottom Left Corner (x_BottomLeft, z_BottomLeft)
0.0, 0.0	Enter (x,z) shift of Centre of Gravity for use in Parallel Axis Theorem
0.0, 0.0	Enter initial U,W velocity of Object
0.0, 0.0	Enter initial Body Angle and Rotation Rate (Omega) - Positive anticlockwise
0.20	Enter coefficient of Friction
1	Add another Floating Body (1=yes)
0.06, 0.06	Enter square cylinderDimension
1.0	Enter specific Weight
3.56, 0.22	Enter x,z of Bottom Left Corner (x_BottomLeft, z_BottomLeft)
0.0, 0.0	Enter (x,z) shift of Centre of Gravity for use in Parallel Axis Theorem
0.0, 0.0	Enter initial U,W velocity of Object
0.0, 10.0	Enter initial Body Angle and Rotation Rate (Omega) - Positive anticlockwise
0.20	Enter coefficient of Friction
0	Add another Floating Body (1=yes)
1	Add water in the flat region ?? (1=yes)
0, 2.0	Cube containing particles : XMin, Xmax ??
0.025, 0.18	Cube containing particles : ZMin, Zmax ??
1	Add water in the inclined region ?? (1=yes)

2.01, 4.75	Cube containing particles : XMin, Xmax ??
0.025, 0.18	Cube containing particles : ZMin, Zmax ??
0	Add a solitary wave ?? (1=yes)
7.0,0.0500	Input the tmax and out
0.0	initial time of recording
0.0,1.0,-1.0	detailed recording: out_dtrecording, Start time, End Time
0.00020,1	input dt ??, variable dt ??
0.2	CFL number (0.1-0.5)
0.92	h=coefficient*sqrt(dx*dx+dy*dy+dz*dz): coefficient ???
1	Use of Riemann Solver: 0=None, 1=Conservative (Vila), 2=NonConservative (Parshikov)
1,1.3	Use TVD, slope limiter (beta_lim)?'
3	Which compiler is desired: 1=gfortran, 2=ifort, 3=win_ifort, 4=Silverfrost FTN95
2	Precision of XYZ Variables: 1=Single, 2=Double

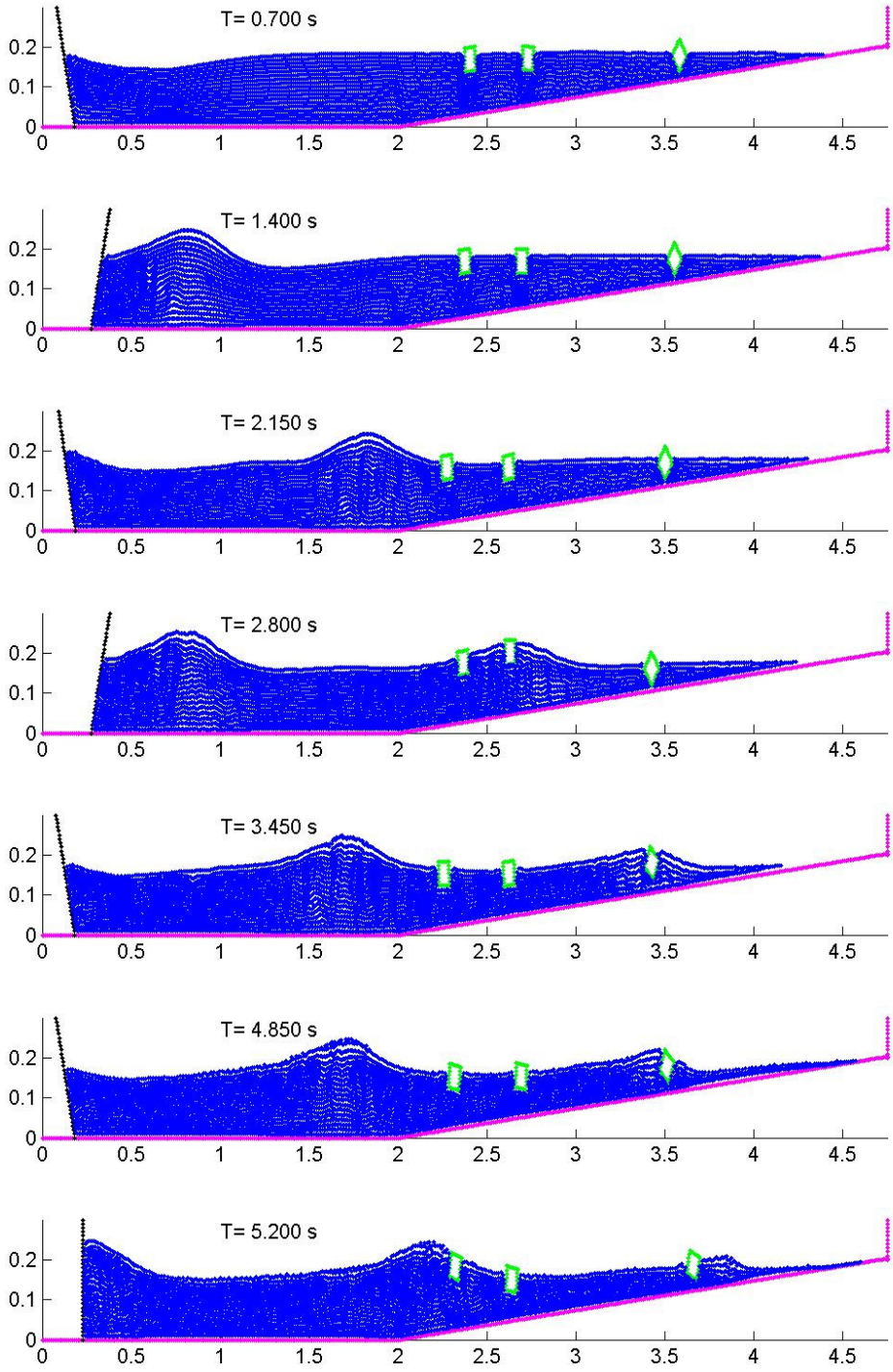


Figure 4.14: Floating bodies in waves 2D.

4.7.2. Case 3D

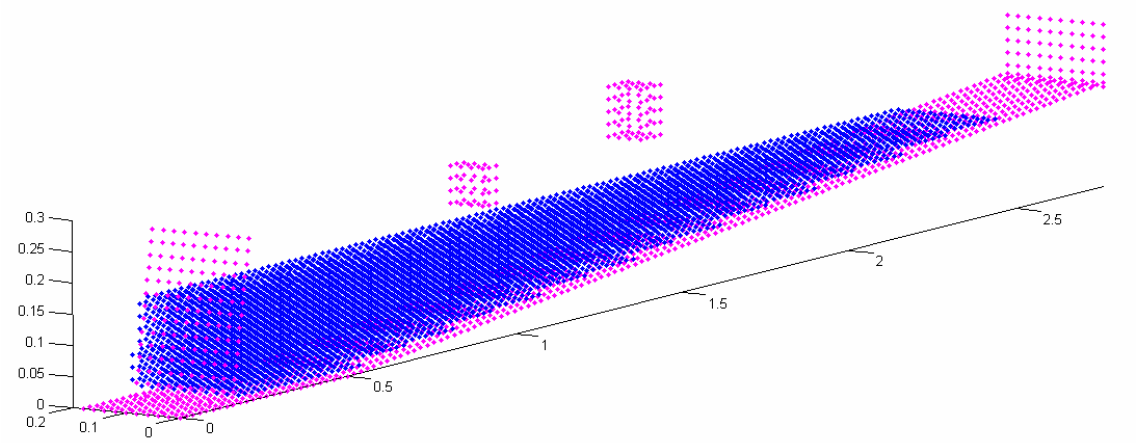


Figure 4.15: Initial configuration of Case6 in 3D.

Input data	Variable description
0	Choose Starting options: 0=new, 1=restart, 2=new with CheckPointg, 3=restart with CheckPointing
2	Kernel: 1=gaussian, 2=quadratic; 3=cubic; 5=Wendland
3	Time-stepping algorithm: 1=predictor-corrector, 2=verlet, 3=symplectic, 4=Beeman
0	Density Filter: 0=none, 1=Shepard filter, 2=MLS
0	Kernel correction 0=None, 1=Kernel correction, 2=Gradient kernel Correction
2	Viscosity treatment: 1=artificial; 2=laminar; 3=laminar + SPS
1.0e-6	Viscosity value(if visc.treatment=1 it's alpha, if not kinem. visc approx 1.e-6)
0	vorticity printing ? (1=yes)
1	Equation of State: 1=Tait's equation, 2=Ideal Gas, 3= Morris
0.15	Maximum Depth (Hmax) to calculate B
16	coefficient (10 , 40) ??
1	Boundary Conditions: 1=Repulsive Force; 2=Dalrymple
0.0e-1	Wall viscosity value for Repulsive Force BC
2	Geometry of the zone: 1=BOX, 2=BEACH, 3=COMPLEX GEOMETRY
2	Initial Fluid Particle Structure: 1= SC, 2= BCC
2.75,0.20,0.25	Box dimension LX, LY, LZ?
0.02,0.02,0.02	Spacing dx, dy, dz?
0.5	Length of Flat Domain
4.2364	Slope (deg) of the inclined plane (beta) ??
0,1,0	Periodic Lateral boundaries in X, Y, & Z-Directions ? (1=yes)
1	If wavemaker will be added, left pannel is not needed (1=yes)
0	Add obstacle (1=yes)
2	Enter Paddle-Type: 1=Piston, 2=Piston-flap, 3=Piston with prescribed motion

0.13	X_PaddleCentre
0.15	paddle_SWL
0.1344	flap_length = distance of pivot point under bed
0.0,0.2	YYMin, YYmax of the wavemaker ??
0.0,0.25	ZZMin, ZZmax of the wavemaker ??
0.0	Initial time of wavemaker = twavemaker ??
1	Number of frequencies ??
0.111	Wavemaker Stroke = 2*Amplitude ??
1.5	Period ??
0	Phase ??
0	twinitial ??
0	Add another wavemaker inside the beach (1=yes)
0	Add gate (1=yes)
0	Add Sliding Raichlen Wedge (1=yes)
1	Add Floating Body (1=yes)
0.06, 0.06, 0.06	Enter X, Y & Z cylinderDimensions
1.0	Enter specific Weight
1.0, 0.08,0.20	Enter x,y,z of Bottom Left Corner (x_BottomLeft, z_BottomLeft)
0.0, 0.0, 0.0	Enter (x,y,z) shift of Centre of Gravity for use in Parallel Axis Theorem
0.0, 0.0, 0.0	Enter initial U,V,W velocity of Object
0.0, 0.0, 0.0	Enter initial Body Angle in X, Y Z Directions
0.0, 0.0, 0.0	Enter initial Rotation Rate (Omega) in X, Y Z Directions
0.20	Enter coefficient of Friction
1	Add another Floating Body (1=yes)
0.06, 0.06, 0.06	Enter X, Y & Z cylinderDimensions
1.0	Enter specific Weight
1.6, 0.16,0.24	Enter x,y,z of Bottom Left Corner (x_BottomLeft, z_BottomLeft)
0.0, 0.0, 0.0	Enter (x,y,z) shift of Centre of Gravity for use in Parallel Axis Theorem
0.0, 0.0, 0.0	Enter initial U,V,W velocity of Object
0.0, 0.0, 0.0	Enter initial Body Angle in X, Y Z Directions
10.0, 0.0, 0.0	Enter initial Rotation Rate (Omega) in X, Y Z Directions
0.20	Enter coefficient of Friction
0	Add another Floating Body (1=yes,0=no)
1	Add water in the flat region ?? (1=yes)
0, 0.49	Cube containing particles : XMin, Xmax ??
0.00, 0.20	Cube containing particles : YMin, Ymax ??
0.02, 0.15	Cube containing particles : ZMin, Zmax ??
1	Add water in the inclined region ?? (1=yes)
0.50, 2.5	Cube containing particles : XMin, Xmax ??
0.00, 0.20	Cube containing particles : YMin, Ymax ??

0.02, 0.15	Cube containing particles : ZMin, Zmax ??
0	Add a solitary wave ?? (1=yes)
4.0,0.01	Input the tmax and out
0.0	initial time of recording
0,1,-1	detailed recording: out_dtrecording, Start time, End Time
0.00020,1	input dt ??, variable dt ??
0.4	CFL number (0.1-0.5)
0.866025	h=coefficient*sqrt(dx*dx+dy*dy+dz*dz): coefficient ???
1	Use of Riemann Solver: 0=None, 1=Conservative (Vila), 2=NonConservative (Parshikov)
1,1.3	Use TVD, slope limiter (beta_lim)?'
3	Which compiler is desired: 1=gfortran, 2=ifort, 3=win_ifort, 4=Silverfrost FTN95
2	Precision of XYZ Variables: 1=Single, 2=Double

4.8. Test case 7: Focused wave group approaching trapezoid

The case can be run using *Case7.bat* whose output directory is *Case7*. The input file *Case7.txt* is located in the output directory. The information contained in that file can be summarized as follows:

4.8.1. Case 2D

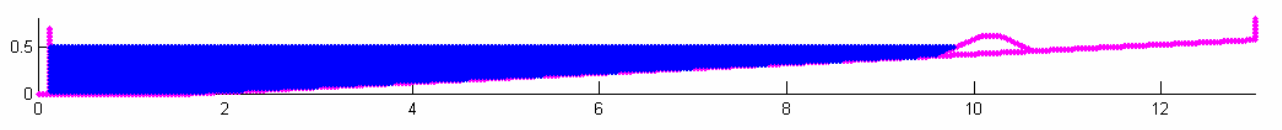


Figure 4.16: Initial configuration of Case7 in 2D.

Input data	Variable description
0	Choose Starting options: 0=new, 1=restart, 2=new with CheckPointg, 3=restart with CheckPointing
3	Kernel: 1=gaussian, 2=quadratic; 3=cubic; 5=Wendland
1	Time-stepping algorithm: 1=Predictor-corrector, 2=Verlet, 3=Symplectic, 4=Beeman
0	Density Filter: 0=none, 1=Shepard filter, 2=MLS
2	Kernel correction 0=None, 1=Kernel correction, 2=Gradient kernel Correction
2	Viscosity treatment: 1=artificial; 2=laminar; 3=laminar + SPS
1.0e-6	Viscosity value(if visc.treatment=1 it's alpha, if not kinem. visc approx 1.e-6)
0	Vorticity printing ? (1=yes)
1	Equation of State: 1=Tait's equation, 2=Ideal Gas, 3= Morris
0.50	Maximum Depth (h_SWL) to calculate B
30	Coefficient of speed of sound (recommended 10 - 40) ??
1	Boundary Conditions: 1=Repulsive Force; 2=Dalrymple
8.0e-4	Wall viscosity value for Repulsive Force BC
2	Geometry of the zone: 1=BOX, 2=BEACH, 3=COMPLEX GEOMETRY
1	Initial Fluid Particle Structure: 1= SC, 2= BCC
13.0,0.8	Box dimension LX, LZ?
0.02,0.02	Spacing dx, dz?
1.48	Length of Flat Domain
2.8624	Slope (deg) of the inclined plane (beta) ??
0,0,0	Periodic Lateral boundaries in X, Y, & Z-Directions ? (1=yes)
1	If wavemaker will be added, left pannel is not needed (1=yes)
1	Add obstacle (1=yes)
2	Choose obstacle: 1=rectangular, 2=trapezoid

9.605,0.40625	Enter (x,z)-start of trapezoid
10.065,0.6085	Enter (x,z)-start of trapezoid top
10.28,0.6085	Enter (x,z)-finish of trapezoid top
10.62,0.457	Enter (x,z)-finish of trapezoid
0	Add another obstacle (1=yes)
3	Enter Paddle-Type: 1=Piston, 2=Piston-flap, 3=Piston with prescribed motion
0.0	X_PaddleCentre
FocusWavePaddle.dat	Enter filename of prescribed motion
0.5	paddle_SWL
0.0,0.7	ZZMin, ZZmax of the wavemaker ??
0.0	Initial time of wavemaker = twavemaker ??
1	Number of frequencies ??
0.0	Wavemaker Stroke = 2*Amplitude ??
0	Period ??
0	Phase ??
0	twinitial ??
0	Add another wavemaker inside the beach (1=yes)
0	Add gate (1=yes)
0	Add Sliding Raichlen Wedge (1=yes)
0	Add Floating Body (1=yes)
1	Add water in the flat region ?? (1=yes)
0.0, 1.48	Cube containing particles : XMin, Xmax ??
0.02, 0.50	Cube containing particles : ZMin, Zmax ??
1	Add water in the inclined region ?? (1=yes)
1.50, 9.915	Cube containing particles : XMin, Xmax ??
0.02, 0.50	Cube containing particles : ZMin, Zmax ??
0	Add a solitary wave ?? (1=yes)
12.0,0.040	Input the tmax and out
0.0	initial time of recording
0.0,1.0,-1.0	detailed recording: out_dtrecording, Start time, End Time
0.00005,1	input dt ??, variable dt ??
0.2	CFL number (0.1 - 0.5)
0.92	h=coefficient*sqrt(dx*dx+dy*dy+dz*dz): coefficient ???
2	Use of Riemann Solver: 0=None, 1=Conservative (Vila), 2=NonConservative (Parshikov)
1, 1.0	Use TVD Riemann Solver, slope limiter (beta_lim)
3	Which compiler is desired: 1=gfortran, 2=ifort, 3=win_ifort, 4=Silverfrost FTN95
1	Precision of XYZ Variables: 1=Single, 2=Double

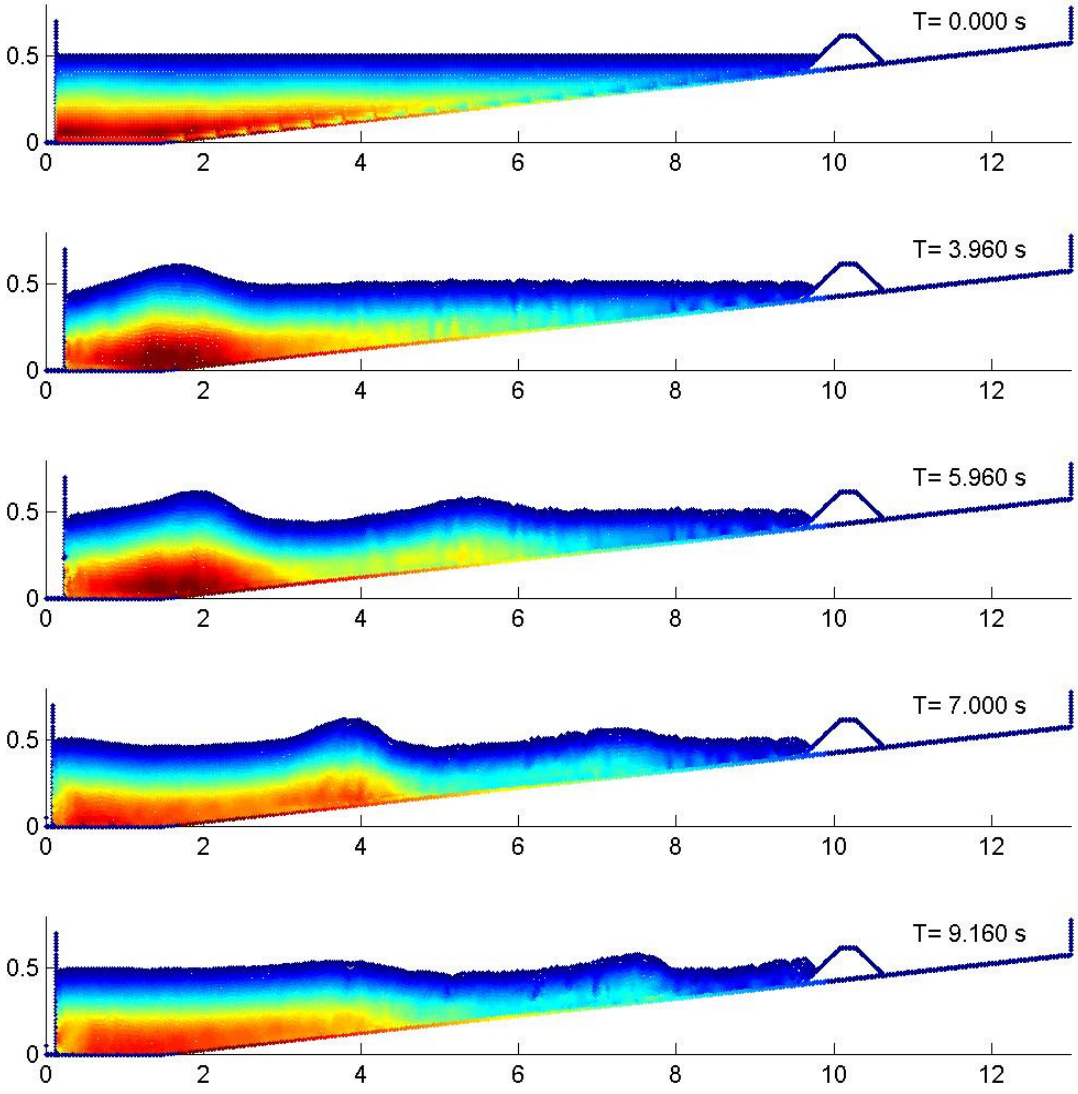


Figure 4.17: Pressure plot in Case7.

4.8.2. Case 3D

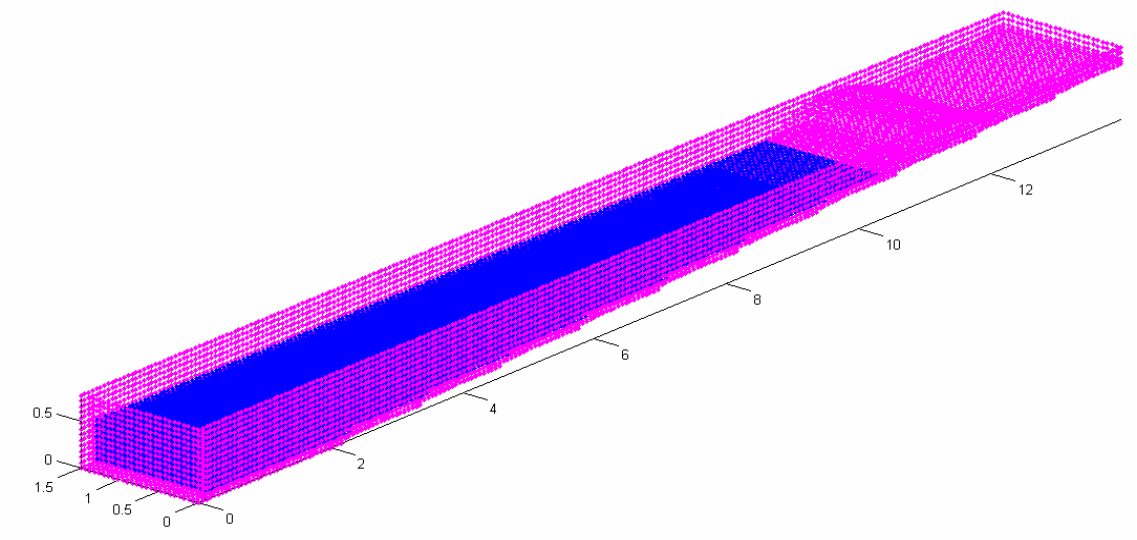


Figure 4.18: Initial configuration of Case7 in 3D.

Input data	Variable description
0	Choose Starting options: 0=new, 1=restart, 2=new with CheckPointg, 3=restart with CheckPointing
3	Kernel: 1=gaussian, 2=quadratic; 3=cubic; 5=Wendland
1	Time-stepping algorithm: 1=Predictor-corrector, 2=Verlet, 3=Symplectic, 4=Beeman
0	Density Filter: 0=none, 1=Shepard filter, 2=MLS
2	Kernel correction 0=None, 1=Kernel correction, 2=Gradient kernel Correction
3	Viscosity treatment: 1=artificial; 2=laminar; 3=laminar + SPS
1.0e-6	Viscosity value(if visc.treatment=1 it's alpha, if not kinem. visc approx 1.e-6)
0	vorticity printing ? (1=yes)
1	Equation of State: 1=Tait's equation, 2=Ideal Gas, 3= Morris
0.5	Maximum Depth (h_SWL) to calculate B
16	Coefficient of speed of sound (recommended 10 - 40) ??
1	Boundary Conditions: 1=Repulsive Force; 2=Dalrymple
8.0e-5	Wall viscosity value for Repulsive Force BC
2	Geometry of the zone: 1=BOX, 2=BEACH, 3=COMPLEX GEOMETRY
1	Initial Fluid Particle Structure: 1= SC, 2= BCC
14.0,1.5,0.8	Box dimension LX, LY, LZ?
0.06,0.06,0.06	Spacing dx, dy, dz?
1.48	Length of Flat Domain
2.8624	Slope (deg) of the inclined plane (beta) ??
0,0,0	Periodic Lateral boundaries in X, Y, & Z-Directions ? (1=yes)
1	If wavemaker will be added, left pannel is not needed (1=yes)
1	Add obstacle (1=yes)
2	Choose obstacle: 1=rectangular, 2=trapezoid
10.357,0.443	Enter (x,z)-start of trapezoid

11.0,0.7	Enter (x,z)-start of trapezoid top
11.5,0.7	Enter (x,z)-finish of trapezoid top
12.,0.55	Enter (x,z)-finish of trapezoid
0.0,1.5	Enter YYMin, YYmax of trapezoid ??
0	Add another obstacle (1=yes)
3	Enter Paddle-Type: 1=Piston, 2=Piston-flap, 3=Piston with prescribed motion
0.0	X_PaddleCentre
FocusWavePaddle.dat	Enter filename of prescribed motion
0.5	paddle_SWL
0.0,1.5	YYMin, YYmax of the wavemaker ??
0.0,0.7	ZZMin, ZZmax of the wavemaker ??
0.0	Initial time of wavemaker = twavemaker ??
1	Number of frequencies ??
0.0	Wavemaker Stroke = 2*Amplitude ??
0	Period ??
0	Phase ??
0	twinitial ??
0	Add another wavemaker inside the beach (1=yes)
0	Add gate (1=yes)
0	Add Sliding Raichlen Wedge (1=yes)
0	Add Floating Body (1=yes,0=no)
1	Add water in the flat region ?? (1=yes)
0.0, 1.48	Cube containing particles : XMin, Xmax ??
0.06, 1.44	Cube containing particles : YMin, Ymax ??
0.06, 0.5	Cube containing particles : ZMin, Zmax ??
1	Add water in the inclined region ?? (1=yes)
1.56, 11.0	Cube containing particles : XMin, Xmax ??
0.06, 1.44	Cube containing particles : YMin, Ymax ??
0.06, 0.5	Cube containing particles : ZMin, Zmax ??
0	Add a solitary wave ?? (1=yes)
10.0,0.10	Input the tmax and out
0.0	initial time of recording
0.0,1.0,-1.0	detailed recording: out_dtrecording, Start time, End Time
0.00005,1	input dt ??, variable dt ??
0.5	CFL number (0.1 - 0.5)
0.866025	h=coefficient*sqrt(dx*dx+dy*dy+dz*dz): coefficient ???
2	Use of Riemann Solver: 0=None, 1=Conservative (Vila), 2=NonConservative (Parshikov)
1,1.4	Use TVD, slope limiter (beta_lim)?'
3	Which compiler is desired: 1=gfortran, 2=ifort, 3=win_ifort, 4=Silverfrost FTN95
2	Precision of XYZ Variables: 1=Single, 2=Double

4.9. Test case 8: Floating bodies with 2D Periodicity

The case can be run using *Case8.bat* whose output directory is *Case8*. The input file *Case8.txt* is located in the output directory.

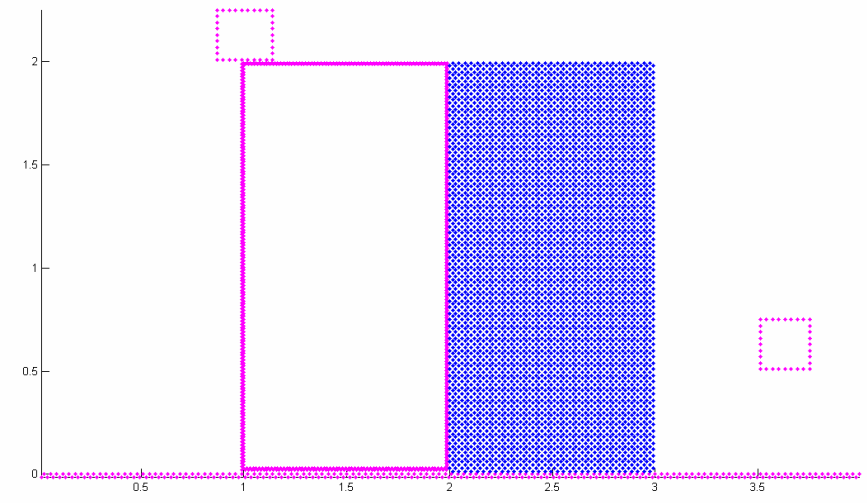


Figure 4.19: Initial configuration of Case8.

Input data	Variable description
0	Choose Starting options: 0=new, 1=restart, 2=new with CheckPointg, 3=restart with CheckPointing
5	Kernel: 1=gaussian, 2=quadratic; 3=cubic; 5=Wendland
1	Time-stepping algorithm: 1=Predictor-corrector, 2=Verlet, 3=Symplectic, 4=Beeman
2	Density Filter: 0=none, 1=Shepard filter, 2=MLS
30	ndt_FilterPerform (if density filter is used) ?
0	Kernel correction 0=None, 1=Kernel correction, 2=Gradient kernel Correction
1	Viscosity treatment: 1=artificial; 2=laminar; 3=laminar + SPS
0.3	Viscosity value(if visc.treatment=1 it's alpha, if not kinem. visc approx 1.e-6)
0	vorticity printing ? (1=yes)
1	Equation of State: 1=Tait's equation, 2=Ideal Gas, 3= Morris
2	Maximum Depth (h_SWL) to calculate B
10	Coefficient of speed of sound (recommended 10 - 40) ??
2	Boundary Conditions: 1=Repulsive Force; 2=Dalrymple
15	ndt_DBCPerform ? (1 means no correction)
1	Geometry of the zone: 1=BOX, 2=BEACH, 3=COMPLEX GEOMETRY
2	Initial Fluid Particle Structure: 1= SC, 2= BCC
4.,4.	Box dimension LX,LZ?
0.03,0.03	Spacing dx,dz?
0	Inclination of floor in X (beta) ??
1,0,0	Periodic Lateral boundaries in X, Y, & Z-Directions ? (1=yes)
0	Add wall
1	Add obstacle (1=y)

1	Choose rectangular (1) or trapezoid (2)
2	Which kind of obstacle: (1) Solid, (2) With Solid Walls
2	Density of points, ndens
1.00,2.0	Cube containing particles : XMin, Xmax ??
0.03,2.0	Cube containing particles : ZMin, Zmax ??
90.00	Inclination (beta) ??
0	Add another obstacle (1=y)
0	Add wavemaker (1=y)
0	Add gate (1=y)
1	Add Floating Body (1=yes)
0.25, 0.25	Enter X & Z cylinderDimensions
1.0	Enter specific Weight
3.50, 0.50	Enter x,z of Bottom Left Corner (x_BottomLeft, z_BottomLeft)
0.0, 0.0	Enter (x,z) shift of Centre of Gravity for use in Parallel Axis Theorem
0.0, 0.0	Enter initial U,W velocity of Object
0.0, 0.0	Enter initial Body Angle and Rotation Rate (Omega) - Positive anticlockwise
0.20	Enter coefficient of Friction
1	Add another Floating Body (1=yes)
0.25, 0.25	Enter X & Z cylinderDimensions
1.0	Enter specific Weight
0.875, 2.01	Enter x,z of Bottom Left Corner (x_BottomLeft, z_BottomLeft)
0.0, 0.0	Enter (x,z) shift of Centre of Gravity for use in Parallel Axis Theorem
0.0, 0.0	Enter initial U,W velocity of Object
0.0, 0.0	Enter initial Body Angle and Rotation Rate (Omega) - Positive anticlockwise
0.0	Enter coefficient of Friction
0	Add another Floating Body (1=yes)
2	Initial conditions: 2) particles on a staggered grid
0	Correct pressure at boundaries ?? (1=y)
2.03,3.0	Cube containing particles : XMin, Xmax ??
0.03,2.	Cube containing particles : ZMin, Zmax ??
0	Fill a new region
3,0.02	Input the tmax and out
0.	initial time of outputting general data
0.0005,1.0,-1.0	For detailed recording during RUN: out_detail, start, end
0.0001,1	Input dt?? , i_var_dt ??
0.2	CFL number (0.1-0.5)
0.92	h=coefficient*sqrt(dx*dx+dz*dz): coefficient ???
0	Use of Riemann Solver: 0=None, 1=Conservative (Vila), 2=NonConservative (Parshikov)
3	Which compiler is desired: 1=gfortran, 2=ifort, 3=win_ifort, 4=Silverfrost FTN95
1	Precision of XYZ Variables: 1=Single, 2=Double

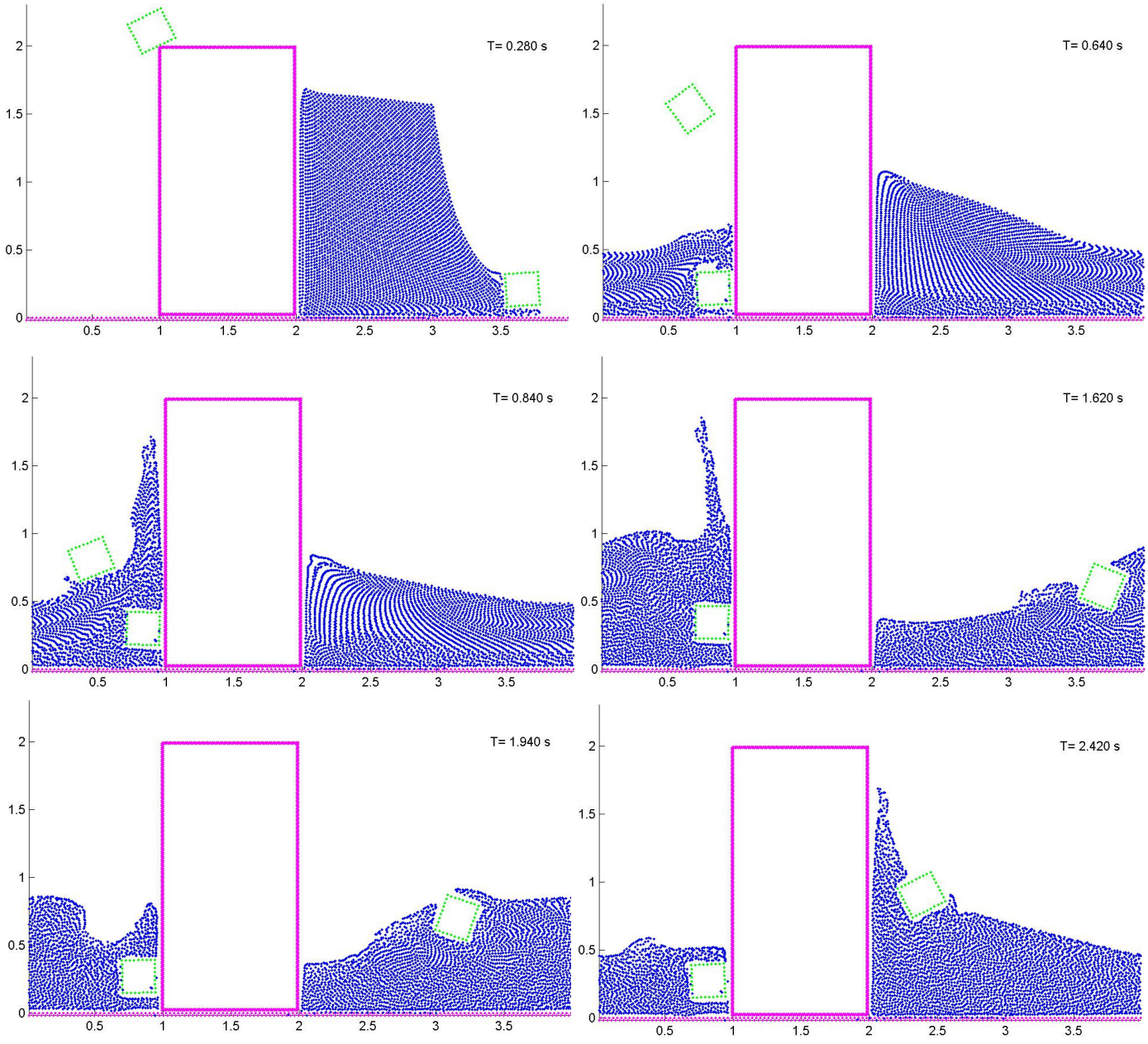
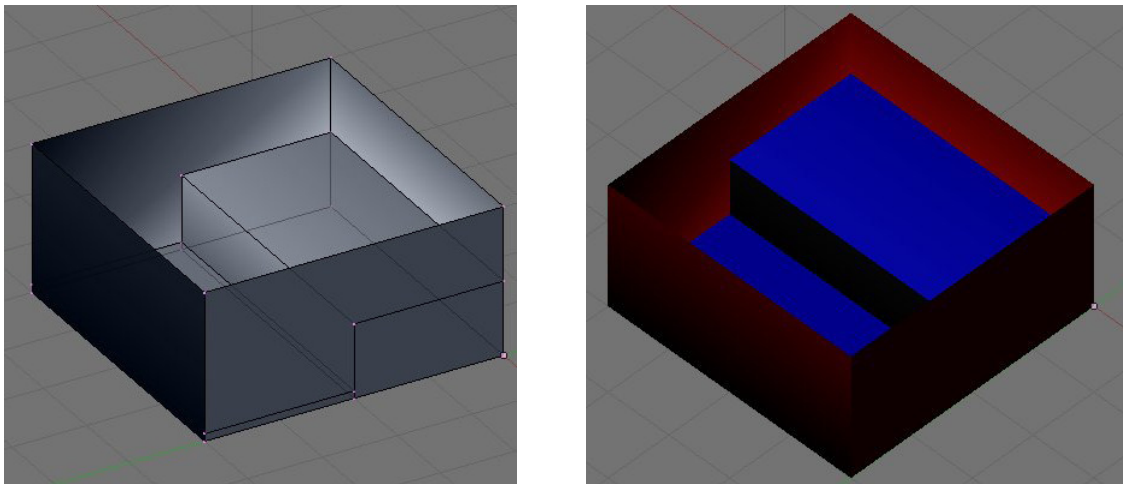


Figure 4.20: Floating bodies with 2D Periodicity.

4.10. Test case 9: Blender

The case shows the capability of the open-source program named Blender (www.blender.org). The user can create a case with more complex geometries using the Blender tools. A blender file and a python script are provided. The script allows creating the input data needed in SPHYSICSGEN.

A complete description can be found at: SPHysics-Blender guide: Mayrhofer, A., Gomez-Gesteira, M., Crespo, A.J.C. and Rogers, B.D. Combining Blender with SPHysics, an Introduction, July 2010.



(a) Creating gate and water surface **(b) Model ready for exporting**

Figure 4.21 Using Blender to generate geometries (Mayrhofer et al. 2010)

5. HOW TO CHANGE SPHysics FOR YOUR APPLICATION

Introduction

When people start using the SPHysics code, we often get asked if the code can do a particular function that is not included in the demonstration cases. The answer we give is normally yes, but the particular functionality required may require some re-coding. We do not normally propose to do this re-coding ourselves unless the application area coincides closely with our own area and current projects, or there is a bug. The reason behind this is that SPHysics is primarily a research code and we have released what we have found useful for our own research. As the code is research oriented, it is up to the user to adapt the code and the subroutines to their satisfaction.

This short section is aimed at helping those people who want to change the code for their own purposes. Figure 5.1 displays the main structure of the code. Here, we list which subroutines in the code you should examine for possible modification. **Important Note**: if you create any new subroutines for the main source code, you must include the names of these new files in the "make files" used for compiling the code which are written in subroutines `tocompile_win_ifort`, `tocompile_ftn95`, `tocompile_gfortran`, `tocompile_ifort` in `SPHYSICSgen_2D/3D.f`. Read Section 3.2.2.3 to see where each of the subroutines is compiled.

1. **Changing the motion of moving objects (forced motion)**
   `movingObjects_2D/3D.f` controls the calling of `movingGate_2D/3D.f`, `movingPaddle_2D/3D.f`, `movingWedge_2D/3D.f` and `rigid_body_motion_2D/3D.f`. If the motion you desire is not covered by these subroutines, then you must create your own.
2. **Changing the boundary conditions.**
   Boundary conditions are treated in each `celij` & `self` subroutines. Any modification to the boundary conditions should be done in these subroutines.
3. **Changing the timestepping algorithm**
   The timestepping is performed in all of the step subroutines: `step_predictor_corrector_2D/3D.f`, `step_verlet_2D/3D.f`, `step_symplectic_ 2D/3D.f`, `step_Beeman_2D/3D.f`. These subroutines then call subroutines ac which control the sweep across the particles (or $2h$ grid) for each (part of the) timestep.
4. **Changing the kernel calculation**
   The smoothing kernel and its derivatives are calculated in the kernel subroutines: `kernel_gaussian_2D/3D.f`, `kernel_quadratic_2D/3D.f`, `kernel_cubic_2D/3D.f` and `kernel_Wendland_2D/3D.f`. In version 2.0 of SPHysics, these can now optionally be corrected for lack of complete support in subroutine `kernel_correction` (see Section 1.9).
5. **Changing the viscous formulation**
   The viscous terms are all calculated in the viscosity subroutines which are called

from `celij` & `self`: `viscosity_artificial_2D/3D.f`, `viscosity_laminar_2D/3D.f`, `viscosity_laminar+SPS_2D/3D.f`. For the SPS turbulence model, the shear stresses are zeroed in subroutine ac and then defined for the next timestep in subroutine `correct_SPS_2D/3D.f`.

6. **Loading in data files and setting useful parameters**

   If you wish to examine and modify what data SPHysics loads initially, all the useful data is imported in subroutine `getdata_2D/3D.f` Furthermore, all the useful parameters that remain the same throughout the simulation are calculated here such as the kernel normalization factors, etc. All global variables are stored in the common blocks contained in `common.2D/3D`.

7. **Zeroing variables**

   Many variables that are evaluated throughout the timestep, such as the accelerations, `ax, ay, az` are zeroed initially in the different ac subroutines: `ac_2D/3D.f`, `ac_Conservative_2D/3D.f`, `ac_Shepard_2D/3D.f`, `ac_MLS_2D/3D.f`, `ac_KGC_2D/3D.f`, `ac_KC_2D/3D.f`.

8. **Changing the input geometry**

   At present, SPHysics is limited to generating a few simple geometric structures both in 2-D and 3-D such as boxes, planar beaches, triangular moving wedges, square floating objects. Generating the geometry is controlled by the code `SPHYSICSgen_2D/3D.f`. As explained in Section 3.2, the input case files can be used to generate a mixture of these basic options. If you wish to modify or add new options, you will need to edit and modify `SPHYSICSgen_2D/3D.f`. Here, we try to give you some indications which subroutines to change:

   (i) main geometric container shape: `box, beach`
   (ii) static obstacles: `trapezoid, wall, obstacles`
   (iii) filling the particles: `fill_part` (& maybe `fluid_particles`)
   (iv) forced motion objects: `gate, wavemaker, RaichlenWedge_Particles, fill_part`
   (v) free-motion objects (floating): `FloatingBody_Particles, fill_part`

As described in Section 4.10, a complex geometry generator is now provided for 3-D applications (see http://wiki.manchester.ac.uk/sphysics/index.php/Contributors) making the creation of new geometries and loading in CAD files more accessible.

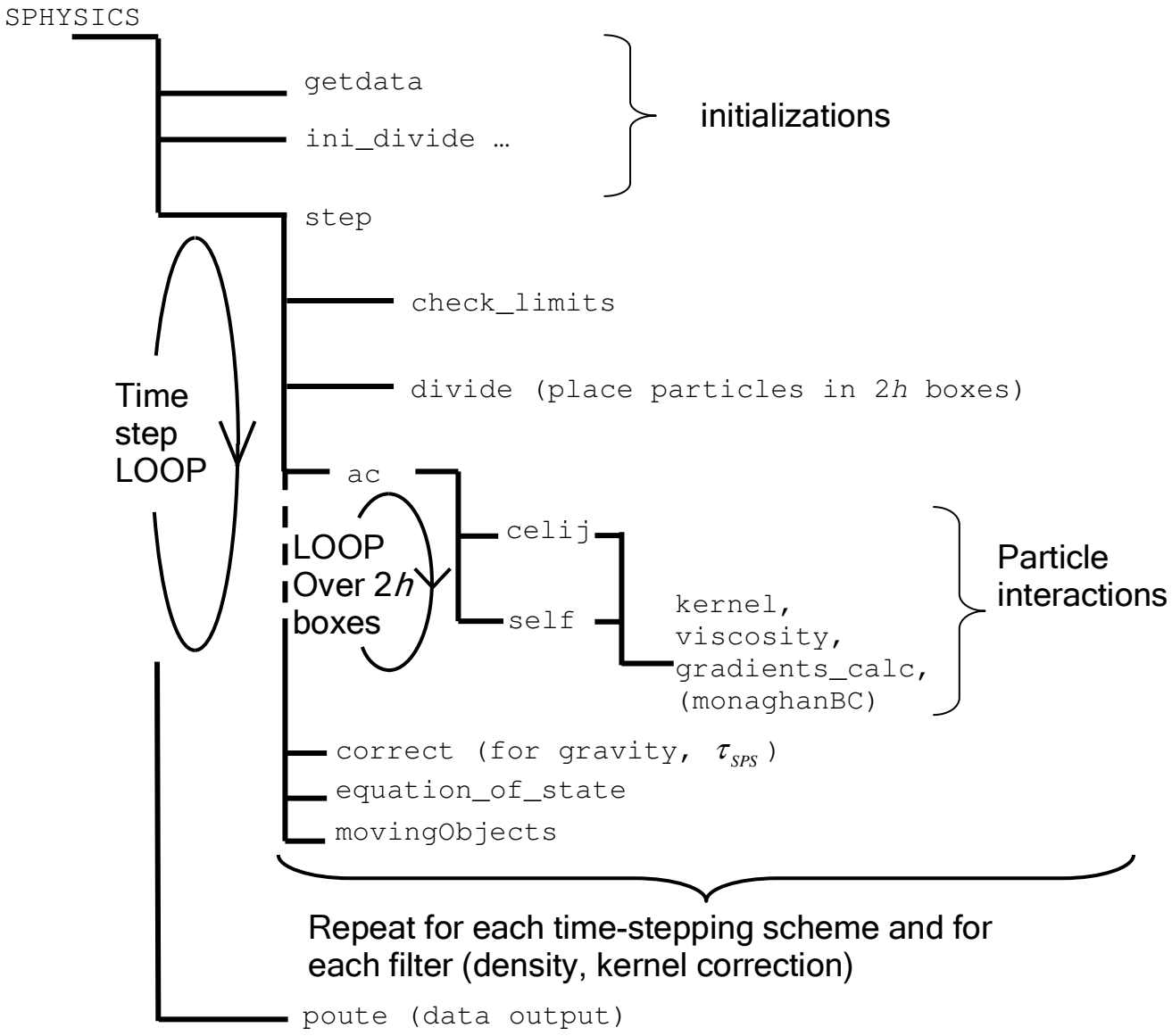


Figure 5.1 Outline of code structure

Here we present a table of the main variables (or those with less than obvious names) and the counterpart in equations:

SPHysics variable	SPH quantity
ax(i), ay(i), az(i)	$\frac{\mathrm{d}\mathbf{v}_a}{\mathrm{d}t} = \frac{\mathrm{d}\vec{v}_a}{\mathrm{d}t}$
ar(i)	$\frac{\mathrm{d}\rho_a}{\mathrm{d}t}$
aTE(i)	$\frac{\mathrm{d}e_a}{\mathrm{d}t}$
cbar	$\overline{c_{ab}} = (c_a + c_b)/2$

cs(i)	c_a
deltaptb(j,1), deltaptb(j,2) deltapsb(j,1), deltapsb(j,2)	$\vec{t}_b \cdot (\vec{r}_{b+1} - \vec{r}_b)/\lvert\vec{r}_{b+1} - \vec{r}_b\rvert$ & $\vec{t}_b \cdot (\vec{r}_b - \vec{r}_{b-1})/\lvert\vec{r}_b - \vec{r}_{b-1}\rvert$ $\vec{s}_b \cdot (\vec{r}_{b+1} - \vec{r}_b)/\lvert\vec{r}_{b+1} - \vec{r}_b\rvert$ & $\vec{s}_b \cdot (\vec{r}_b - \vec{r}_{b-1})/\lvert\vec{r}_b - \vec{r}_{b-1}\rvert$ (for boundary particles only)
dudx_CSPH(i), dudy_CSPH(i), …	$\frac{\partial u_a}{\partial x_a}$, $\frac{\partial u_a}{\partial y_a}$, …
drx, dry, drz	$x_{ab} = (x_a - x_b),\ y_{ab} = (y_a - y_b),\ z_{ab} = (z_a - z_b)$
duxp, duyp, duzp	$u_{ab} = (u_a - u_b),\ v_{ab} = (v_a - v_b),\ w_{ab} = (w_a - w_b)$
frxi, frxj, frzi, frzj	$\frac{\partial W_{ab}}{\partial x_a}$, $\frac{\partial W_{ba}}{\partial x_b}$, $\frac{\partial W_{ab}}{\partial z_a}$, $\frac{\partial W_{ba}}{\partial z_b}$
fxbp, fybp, fzbp	Boundary forces $\mathbf{f} = (f_x, f_y, f_z)\mathbf{n}$
pm(j)	m_b
pr(i)	$\frac{P_a}{\rho_a^2}$
pVol(j)	$V_b = \frac{m_b}{\rho_b}$
p_v	$\left(\frac{P_a}{\rho_a^2} + \frac{P_b}{\rho_b^2} + \Pi_{ab}\right)$ or $\left(\frac{P_a}{\rho_a^2} + \frac{P_b}{\rho_b^2} + \Pi_{ab} + Rf_{ab}\right)$
rhop(i)	ρ_a
rhop_sum	$\sum_b \rho_b W_{ab} \frac{m_b}{\rho_b} = \sum_b m_b W_{ab}$
rr2	r_{ij}^2
sum_wab	$\sum_b W_{ab} \frac{m_b}{\rho_b}$
up(i), vp(i), wp(i)	$\mathbf{v}_a = \vec{v}_a = (u_a, v_a, w_a)$
ux(i), vx(i), wx(i)	$\sum_b \frac{m_b}{\overline{\rho}_{ab}} \vec{v}_{ba} W_{ab}$ (XSPH correction)
xnb(j), ynb(j), znb(j)	$\mathbf{n}_b = \vec{n}_b = (n_x, n_y, n_z)_b$ (Boundary normals)
xsb(j), ysb(j), zsb(j)	$\mathbf{s}_b = \vec{s}_b = (s_x, s_y, s_z)_b$ (Boundary tangent **s**)
xtb(j), ytb(j), ztb(j)	$\mathbf{t}_b = \vec{t}_b = (t_x, t_y, t_z)_b$ (Boundary tangent **t**)
xp(i), yp(i), zp(i)	$\mathbf{r}_a = \vec{r}_a = (x_a, y_a, z_a)$
Wab	$W_{ab} = W(\lvert\vec{r}_a - \vec{r}_b\rvert)$
bigUdot, bigVdot, bigWdot, OmegaX/Y/Zdot, bigMass	$\frac{\mathrm{d}\mathbf{V}}{\mathrm{d}t}$, Ω, M (for rigid body dynamics, eq 2.21)
bigU, BigV, bigW, bigOmegaX	$\mathbf{V} = \vec{V} = (U, V, W)$, $\mathbf{\Omega} = (\Omega_x, \Omega_y, \Omega_z)$

6. 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.

7. REFERENCES

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

Beeman, D. (1976). Some Multistep Methods for Use in Molecular Dynamics Calculations. *Journal of Computational Physics*, **20**, 130–139.

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 T.-S. L. Lok. Variational and momentum preservation aspects of Smoothed Particle Hydrodynamic formulations, *Comput. Methods Appl. Mech.* Engrg, **180**, 97-115, 1999.

Capone, T., Panizzo, A., Cecioni, C. and Dalrymple, R.A. (2007). "Accuracy and Stability of Numerical Schemes in SPH", *SPHERIC Second Intl. Workshop, Madrid.*

Cha, S.-H., and Whitworth, A.P., Implementations and tests of Godunov-particle hydrodynamics, *Mon. Not. R. Astro. Soc*., **340**, 73-90, 2003.

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

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.

Dilts, G. A. Moving-Least–Squares-Particle Hydrodynamics – I. Consistency and stability, *Int. J. Numer. Meth. Engng,* **44**, 1115-1155, 1999.

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.

Guilcher, P.M., Ducorzet, G., Alessandrini, B. and P. Ferrant, Water wave propagation using SPH models, *Proc. of 2nd Int. SPHERIC Workshop*, Spain, 119-124, 2007.

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

Hirsch C. *Numerical Computation of Internal and External Flows, Vol. 1,* John Wiley and Sons 1998.

Hughes, J. and Graham, D., Comparison of incompressible and weakly-compressible SPH models fro free-surface water flows, *J. Hyd. Res.*, in press, 2010.

Leimkuhler B J, Reich S, Skeel RD. *Integration Methods for Molecular dynamic IMA Volume in Mathematics and its application*. Springer 1997.

Lo, E.Y.M. and Shao, S., Simulation of near-shore solitary wave mechanics by an incompressible SPH method, *Applied Ocean Research*, **24**, 275-286, 2002.

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

Liu, W., Li, S., and Belytscho, T. 1997. "Moving least square Kernel Galerkin method (I) methodology and convergence". *Comput. Methods Appl. Mech. Engineering.*, 143:113.

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.

Monaghan, J.J., A. Kos, and N. Issa., Fluid motion generated by impact. *J. of Waterway, Port, Coastal and Ocean Engineering*, **129**, 250-259, 2003.

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.

Rogers, B.D. and R.A. Dalrymple, SPH Modeling of tsunami waves, *Advances in Coastal and Ocean Engineering, Vol. 10 Advanced Numerical Models for tsunami waves and runup*, World Scientific, 2008.

Rogers, B.D., Dalrymple, R.A., Stansby, P.K., Simulation of caisson breakwater movement using SPH, *Journal of Hydraulic Research*, **48**, Extra Issue, 135-141, 2010.

Toro, E.F., Spruce, M. & Speares, W., Restoration of the contact surface in the HLL-Riemann solver, *Shock Waves*, **4**, 25-34, 1994.

Toro, E.F., *Shock capturing methods for free surface shallow flows*, John Wiley & Sons, 2001.

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

Vila, J.-P., On particle weighted methods and Smooth Particle Hydrodynamics, *Mathematical Models and Methods in Applied Sciences*, **9**(2), 161-209, 1999.

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

8. APPENDIX: SPS Turbulence Model

This section provides a short overview of the LES-type Sub-Particle Scale (SPS) turbulence model implemented in SPHysics. This work was inspired by the pioneering work of Lo & Shao (2002). For a detailed description of turbulence and Large Eddy Simulation (LES), the reader is referred to the comprehensive text by Pope (2000).

Large-eddy simulation (LES) for SPH

Large–eddy simulation (LES) is a numerical approach for modelling the effect of turbulence. In mesh-based schemes, the principal idea of LES is that the largest scales of motion are resolved by the grid while motions on the scales that are smaller than the grid are *modelled* or represented by a Sub-Grid Scale (SGS) turbulence model. Hence, the larger scales are solved explicitly while the SGS motions are modelled. The physical reasoning behind this is that the main energy-containing scales are dependent on the flow, while the smaller dissipative scales are more straightforward to characterise in terms of energy dissipation. This avoids having to perform computationally expensive Direct Numerical Simulation (DNS) calculations. However, there are still some minimum requirements that an LES calculation must resolve 80% of the flow, etc., (Pope 2000).

LES uses spatial filters in the surrounding grid to include the effect of the sub-grid motions. This is based on the basic convolution integral for a variable u:

$$\overline{u}(\mathbf{x},t)=\int_{\Omega} G(\mathbf{x}-\xi)u(\xi,t)\mathrm{d}\xi, \tag{A1}$$

where $\overline{u}$ is the spatially-filtered variable, $\mathbf{x}$ is a position in space, G is the filter function (with characteristic length Δ) which can be a variety of functions including a box function or a Fourier filter performed in the frequency domain, etc. The analogies between SPH and LES are clear and have been noted by several researchers (Pope 2000, Issa 2005, etc.). We can then decompose the velocity u into the sum of the mean spatial average and a spatial fluctuation

$$u=\overline{u}+u', \tag{A2}$$

such that $\overline{u'}\neq 0$.

In LES, the equations being solved are commonly in the incompressible Navier-Stokes equations. However, in SPH we are using the continuity equation for a slightly compressible fluid to represent water, etc. Hence we are generally solving the compressible Navier-Stokes equations. Here, for the presentation of LES-type equations we use the standard tensor subscript notation of i and j to denote coordinate directions:

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_j}\left(\rho u_j\right) = 0 \tag{A3b}$$

$$\frac{\partial}{\partial t}\left(\rho u_i\right) + \frac{\partial}{\partial x_j}\left(\rho u_i u_j + p\delta_{ij} - \sigma_{ij}\right) = f_i \tag{A3c}$$

$$\frac{\partial}{\partial t}\left(\rho E_i\right) + \frac{\partial}{\partial x_j}\left(\rho u_j E - \sigma_{ij} u_i + q_j\right) = f_j u_j \tag{A3d}$$

where ρ is density, u_j is the velocity in the x_i direction, p is pressure, δ_{ij} is the delta function ($\delta_{ij} = \{1 \quad i = j, 0 \quad i \neq j \}$), E is the total energy, q is a heat flux and the stress σ is given by

$$\sigma_{ij} = 2\mu S_{ij} - \tfrac{2}{3}\mu\delta_{ij} S_{kk} \tag{A4}$$

And the strain rate tensor S_{ij} is given by by

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right). \tag{A5}$$

We are then ready to apply the LES filter in Equation (A1) above to each equation of (A3) in turn. Since we are solving the compressible Navier-Stokes equations, we use Favre-averaging, $\tilde{f} = \overline{\rho f}/\overline{\rho}$, which avoids the generation of SGS terms in the filtered continuity equation. For the continuity and momentum equations, this gives us

$$\frac{\partial \overline{\rho}}{\partial t} + \frac{\partial}{\partial x_j}\left(\overline{\rho}\tilde{u}_j\right) = 0 \tag{A6a}$$

$$\frac{\partial}{\partial t}\left(\overline{\rho}\tilde{u}_i\right) + \frac{\partial}{\partial x_j}\left(\overline{\rho}\tilde{u}_i\tilde{u}_j + \overline{p}\delta_{ij} - \tilde{\sigma}_{ij}\right) = -\frac{\partial \tau_{ij}}{\partial x_j} \tag{A6b}$$

where τ_{ij} are the Sub-Grid Scale (SGS) shear stresses that arise from the filtering process and physically the represent the motion that occurs on a scale smaller than the grid spacing Δx:

$$\tau_{ij} = -\overline{\rho}\left(\tilde{u}_i\tilde{u}_j - \widetilde{u_i u_j}\right). \tag{A7}$$

This SGS model then requires closure which is normally expressed as

$$\tau_{ij} = -\overline{\rho}\left(\tilde{u}_i\tilde{u}_j - \widetilde{u_i u_j}\right) \approx \overline{\rho}\nu_t\left(2\tilde{S}_{ij} - \tfrac{2}{3}\tilde{S}_{kk}\delta_{ij}\right) - \tfrac{2}{3}\overline{\rho}C_I\Delta^2\delta_{ij}\left|\tilde{S}_{ij}\right|^2 \tag{A8}$$

Note that $\tilde{\sigma}$ does not represent turbulent stresses, but the filtered laminar stress components. The key part of any LES model is then providing a value for the turbulent eddy viscosity ν_t which in SPHysics is achieved using a non-dynamic Smagorinsky model:

$$\nu_t = \left(C_s\Delta\right)^2\left|\tilde{S}_{ij}\right|. \tag{A9}$$

In order to solve the LES equations in SPH, we express in Lagrangian form:

$$\frac{\mathrm{d}\overline{\rho}}{\mathrm{d}t} = -\overline{\rho}\frac{\partial \tilde{u}_j}{\partial x_j} \tag{A10a}$$

$$\frac{\mathrm{d}\tilde{u}_i}{\mathrm{d}t} = -\frac{1}{\overline{\rho}}\frac{\partial \overline{p}}{\partial x_j} - \frac{1}{\overline{\rho}}\frac{\partial \tilde{\sigma}_{ij}}{\partial x_j} - \frac{1}{\overline{\rho}}\frac{\partial \tau_{ij}}{\partial x_j} \tag{A10b}$$

References

Issa, R., *Numerical assessment of the Smoothed Particle Hydrodynamics gridless method for incompressible flows and its extension to turbulent flows*, Ph.D. Thesis, University of Manchester Institute of Science and Technology (UMIST), 2004.

Lo, E.Y.M. and Shao, S., Simulation of near-shore solitary wave mechanics by an incompressible SPH method, *Applied Ocean Research*, **24**, 275-286, 2002.

Pope, S.B., *Turbulent Flows*, Cambridge University Press, 2000.

9. PUBLICATIONS USING THE SPHysics CODE

Journal Papers

Dalrymple, R.A. and B.D. Rogers, "Numerical Modeling of Water Waves with the SPH Method," Coastal Engineering, 53/2-3, 141-147, 2006.

Crespo, A.J., M. Gómez-Gesteira, and R.A. Dalrymple, "Boundary Conditions Generated by Dynamic Particles in SPH Methods", CMC: Computers, Materials, & Continua, **5**, 3, 173-184, 2007.

Gómez-Gesteira, M., D. Cerquiero, A.J.C. Crespo and R.A. Dalrymple, "Green Water Overtopping Analyzed with an SPH Model," Ocean Engineering, 32, 2, 223-238, 2005.

Gómez-Gesteira, M., R.A. Dalrymple, A.J.C. Crespo, and D. Cerquiero, "Uso de la Tecnica SPH para el Estudio de la Interaccion entre Olas y Estructuras," Ingenieria del Agua, 11, 2, 2004.

Gómez-Gesteira, M. and R.A. Dalrymple, "Using a 3D SPH Method for Wave Impact on a Tall Structure, *J. Waterways, Port, Coastal, Ocean Engineering,* 130, 2, 63-69, 2004.

Crespo, A.J.C., M. Gómez-Gesteira, and R.A. Dalrymple, "3D SPH Simulation of large waves mitigation with a dike", Journal of Hydraulic Research, **45**, 5, 631-642, 2007.

Crespo, A.J.C., M. Gómez-Gesteira, and R.A. Dalrymple, "Modeling Dam Break Behavior over a Wet Bed by a SPH Technique", *Journal of Waterway, Port, Coastal and Ocean Engineering*, 134(6), 3131320, 2008.

M. Gómez-Gesteira, B. D. Rogers, R. A. Dalrymple and A.J.C. Crespo, " State of the art of classical SPH for free-surface flows", *Journal of Hydraulic Research*, In Press, 2010.

M. Gómez-Gesteira, B. D. Rogers, D. Violeau, J. M. Grassa and A.J.C. Crespo, " SPH for free-surface flows", *Journal of Hydraulic Research*, In Press, 20010.

M. S. Narayanaswamy, A.J.C. Crespo, M. Gómez-Gesteira and R. A. Dalrymple, " SPHysics-Funwave hybrid model for coastal wave propagation", *Journal of Hydraulic Research*, In Press, 2010.

Rogers, B.D., Dalrymple, R.A., Stansby, P.K., Simulation of caisson breakwater movement using SPH, *Journal of Hydraulic Research*, In Press, 2010.

Conference Proceedings

Crespo, A.J., M. Gómez-Gesteira, and R.A. Dalrymple, "Vorticity Generated By A Dam Break Over A Wet Bed Modeled By Smoothed Particle Hydrodynamics", Proceedings of 32nd Congress of IAHR, the International Association of Hydraulic Engineering & Research, CORILA, 2007.

Dalrymple, R., B.D. Rogers, M. Narayanaswamy, S. Zou, M. Gesteira, A.J.C. Crespo and A. Panizzo, "Smoothed Particle Hydrodynamics for Water Waves", Proceedings

of the 26th International Conference on Offshore Mechanics and Artic Engineering, ASME, 2007.

Rogers, B.D. and R.A. Dalrymple, "Three-Dimensional SPH-SPS Modeling of Wave Breaking", *Symposium on Ocean Wave Measurements and Analysis, ASCE, Madrid, 2005.*

Dalrymple, R.A., ``New Technology: SPH for Coastal Processes, *Keynote Address, Symposium on Ocean Wave Measurements and Analysis, ASCE, Madrid, 2005.*

Narayanaswamy, M. and R.A. Dalrymple, "A Hybrid Boussinesq and SPH Model for Forced Oscillations", *Symposium on Ocean Wave Measurements and Analysis, ASCE, Madrid, 2005.*

Rogers, B.D. and R.A. Dalrymple, "SPH Modeling of Breaking Waves", *Proc. 29th Intl. Conference on Coastal Engineering, Lisbon, World Scientific Press, 2004.*

Book Chapters

Rogers, B.D. and R.A. Dalrymple, "SPH Modeling of Tsunamis", *Proc. Intl. Workshop on Longwave Run-up, Advances in Coastal Engineering Series, P.L.-F. Liu, ed., World Scientific* In Press*,* 2006.

Dalrymple, R.A., Gómez-Gesteira, M., Rogers, B.D., Panizzo, A., Zou, S., Crespo, A.J.C., Cuomo, G., And Narayanaswamy, M., "Smoothed Particle Hydrodynamics for Water Waves", *in Advances in numerical simulation of nonlinear water waves,* Ma, Q. ed., World Scientific Publishing, 2009.

PhD Thesis

Shan Zou, Coastal Sediment Transport Simulation by Smoothed Particle Hydrodynamics, Johns Hopkins University, 2007.

Crespo, A.J.C., Application of the Smoothed Particle Hydrodynamics model SPHysics to free-surface hydrodynamics, University of Vigo, 2008

Narayanaswamy, M., A Hybrid Boussinesq SPH Wave Propagation Model with Applications to Forced Waves in Rectangular Tanks, Johns Hopkins University, 2008.