

## ACSE-4 Project 2: A Smoothed Particle Hydrodynamics Solver

### Introduction to SPH

Smoothed Particle Hydrodynamics (SPH) is a meshless method for solving the Navier-Stokes equation. As it is meshless and Lagrangian (reference frame follows the fluid motion), it is ideal for solving problems involving fluid flow with interfaces and free surfaces. In this task you are going to write an SPH-based simulator and use it to solve a wave generation problem. The equations listed below include some which are purely there for background context and some which are required within the SPH simulation code. Those which you will need to implement are surrounded by a box.

The basic idea of SPH is that you model individual points which follow the fluid and are each associated with a given mass of the fluid,  $m_i$ . In order to carry out differentiation on them, the discrete values of any field (e.g., pressure) associated with the points are converted into a continuous function using a smoothing kernel (hence the smoothed in the name):

$$\langle A \rangle(\mathbf{x}) = \sum_{j=1}^N m_j \frac{A_j}{\rho_j} W(\mathbf{x} - \mathbf{x}_j, h)$$

Where  $\langle A \rangle(\mathbf{x})$  is the smoothed value of  $A$  at location  $\mathbf{x}$ , which is a vector,  $A_j$  is the value of  $A$  associated with particle  $j$ ,  $\rho_j$  is the density of particle  $j$  and  $W(\mathbf{x} - \mathbf{x}_j, h)$  is a smoothing kernel which depends upon the vector  $\mathbf{x} - \mathbf{x}_j$  and a characteristic smoothing length  $h$ .

In SPH we are typically only interested in the values at the location of the points. We will thus use the following notation:

$$\langle A \rangle(\mathbf{x}_i) = \langle A \rangle_i = \sum_{j=1}^N m_j \frac{A_j}{\rho_j} W(\mathbf{r}_{ij}, h)$$

Where  $\mathbf{r}_{ij} = \mathbf{x}_i - \mathbf{x}_j$ . Also note that as this is a smoothing rather than an interpolation, in general  $\langle A \rangle_i \neq A_i$ . At any point there is therefore a choice that can be made as to whether to use the smoothed or nominal value at that point.

The smoothing kernel must possess the property that it integrates to 1 when integrated over its area (or volume in 3D). The form of this smoothing kernel is important. It should have compact support (i.e. it should go to zero at a finite distance) so that only the neighbouring particles are involved in the calculations. It should also be smooth and, ideally, close to normal in shape. A number of different kernels have been proposed, but in this project I wish you to use the cubic spline:

$$W(\mathbf{r}, h) = \frac{10}{7\pi h^2} \begin{cases} 1 - \frac{3}{2}q^2 + \frac{3}{4}q^3 & \text{if } 0 \leq q \leq 1 \\ \frac{1}{4}(2 - q)^3 & \text{if } 1 \leq q \leq 2 \\ 0 & \text{if } q > 2 \end{cases}$$

Where  $q = \frac{|\mathbf{r}|}{h}$

Note that this kernel is symmetric and thus only depends on the distance between the points being considered. The prefactor in the equation will depend upon the number of dimensions and this equation is thus for 2D problems only.

If we want to use SPH to approximate the Navier-Stokes equation we need to have approximations for the partial derivatives of the variables of interest:

$$\nabla A_i \approx \sum_{j=1}^N m_j \frac{A_j}{\rho_j} \nabla W(\mathbf{r}_{ij}, h)$$

This is useful as  $\nabla W(\mathbf{r}_{ij}, h)$  is known analytically, while all the other values are also known for each point. This means that calculating the gradient of a quantity is no more computationally expensive than calculating the average value at a point. For symmetric kernels such as we will be using in this project, we can further simplify this equation by using the following relationship:

$$\nabla W(\mathbf{r}_{ij}, h) = \frac{dW}{dr}(|\mathbf{r}_{ij}|, h) \mathbf{e}_{ij}$$

Where  $\mathbf{e}_{ij} = \frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|}$  and  $\frac{dW}{dr}(r, h)$  is a known function of  $r$  and  $h$ , which you will need to derive by differentiating the kernel (note that you must do this analytically, not numerically). Substituting this back into the previous equation gives:

$$\nabla A_i \approx \sum_{j=1}^N m_j \frac{A_j}{\rho_j} \frac{dW}{dr}(|\mathbf{r}_{ij}|, h) \mathbf{e}_{ij}$$

The problem with this equation is that it is neither symmetric nor anti-symmetric (the contribution of particle  $i$  on particle  $j$  is not the same as the contribution of particle  $j$  on particle  $i$ ). (Anti)symmetry is desirable if we wish the system to explicitly conserve momentum and mass.

To achieve symmetric or anti-symmetric differentials we can use one of the following two relationships:

$$\rho \nabla A = \nabla(\rho A) - A \nabla \rho$$

$$\frac{\nabla A}{\rho} = \nabla \left( \frac{A}{\rho} \right) + \frac{A}{\rho^2} \nabla \rho$$

Substituting the previous relationship into these equations results in the following approximations:

$$\rho \nabla A \approx \sum_{j=1}^N m_j (A_j - A_i) \frac{dW}{dr}(|\mathbf{r}_{ij}|, h) \mathbf{e}_{ij} \quad - \text{Symmetric (note the } \mathbf{e}_{ij} = -\mathbf{e}_{ji})$$

$$\frac{\nabla A}{\rho} \approx \sum_{j=1}^N m_j \left( \frac{A_i}{\rho_i^2} + \frac{A_j}{\rho_j^2} \right) \frac{dW}{dr}(|\mathbf{r}_{ij}|, h) \mathbf{e}_{ij} \quad - \text{Anti-symmetric}$$

### Approximating the Navier-Stokes Equation using SPH

We now have all the tools required to approximate the Navier-Stokes equation and the associated continuity equations:

$$\frac{D\mathbf{v}}{Dt} = -\frac{\nabla P}{\rho} + \frac{\mu}{\rho} \nabla^2 \mathbf{v} + \mathbf{g}$$

$$\frac{D\rho}{Dt} = -\rho \nabla \cdot \mathbf{v}$$

It should be noted that in a Lagrangian reference frame  $\frac{D}{Dt} \equiv \frac{\partial}{\partial t}$  when the time derivatives are carried out within the reference frame of differential volumes of the moving fluid (which in the SPH context is equivalent to the reference frame of the particles).

We have thus far not dealt with approximations for second derivatives, which are required to approximate the shear stress term. We could do this by taking the 2<sup>nd</sup> derivative of the kernel, but this is not a good approximation unless there are a large number of particles in the support region of the kernel, which is computationally expensive. An alternative is to use the following relationship:

$$\nabla^2 \mathbf{v}_i = \nabla \cdot \nabla \mathbf{v}_i \approx \sum_{j=1}^N m_j \left( \frac{1}{\rho_i^2} + \frac{1}{\rho_j^2} \right) \frac{dW}{dr}(|\mathbf{r}_{ij}|, h) \nabla \mathbf{v}_{ij} \cdot \mathbf{e}_{ij}$$

We can make use of the fact that  $\nabla \mathbf{v}_{ij} \cdot \mathbf{e}_{ij}$  is the velocity gradient in the  $ij$  direction. This can be approximated as the difference in the velocity in the  $ij$  direction divided by the distance over which this occurs. Combining this approximation with the previous gradient approximations results in the following approximations for the Navier-Stokes and continuity equations:

$$\begin{aligned} \frac{\partial \mathbf{v}_i}{\partial t} \approx \mathbf{a}_i &= - \sum_{j=1}^N m_j \left( \frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} \right) \frac{dW}{dr}(|\mathbf{r}_{ij}|, h) \mathbf{e}_{ij} + \mu \sum_{j=1}^N m_j \left( \frac{1}{\rho_i^2} + \frac{1}{\rho_j^2} \right) \frac{dW}{dr}(|\mathbf{r}_{ij}|, h) \frac{\mathbf{v}_{ij}}{|\mathbf{r}_{ij}|} \\ &\quad + \mathbf{g} \\ \frac{\partial \rho_i}{\partial t} \approx D_i &= \sum_{j=1}^N m_j \frac{dW}{dr}(|\mathbf{r}_{ij}|, h) \mathbf{v}_{ij} \cdot \mathbf{e}_{ij} \end{aligned}$$

Note that the sums in the above equations MUST NOT include the combinations where  $i = j$  as this will result in a division by zero! They also need only include neighbouring particles (those within  $2h$  of the particle being considered. Note the dot product in the continuity equation.

These are the main equations that you will need to solve. You will notice that you currently have 3 equations (momentum equations in the  $x$  and  $y$  directions and the continuity equation) and 4 unknowns (velocity in the  $x$  and  $y$  directions, pressure and density) at each point. There is therefore still a missing equation required to close the system.

The way we will be closing the system is to assume that it is slightly compressible and to use an equation of state, which relates the pressure to the density. In this work we will use the Tait equation, which is a very stiff equation of state:

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

Where  $B = \frac{\rho_0 c_0^2}{\gamma}$ .  $\rho_0$  is the initial or reference density of the fluid and  $c_0$  is a numerical speed of sound, which should NOT be the same as the actual speed of sound (the simulation would be prohibitively slow). The value of  $c_0$  should be chosen to be about an order of magnitude larger than the highest sustained speed in the system (for your work you can use  $c_0 = 20 \text{ m/s}$ , though you should test its impact on the results). To make sure that the equation is stiff enough use  $\gamma = 7$ .

### Time stepping in SPH

The above equations allow a particle's acceleration and rate of change of density to be calculated as a function of its neighbour's positions, velocities and densities/pressures. We now need to be able to update the particle's position, velocity and density, with the updated density being used to calculate a new pressure. To save complexity we will be doing this explicitly. We could use a simple explicit Euler scheme:

$$\begin{aligned} \mathbf{x}_i^{t+1} &= \mathbf{x}_i^t + \Delta t \mathbf{v}_i^t \\ \mathbf{v}_i^{t+1} &= \mathbf{v}_i^t + \Delta t \mathbf{a}_i^t \\ \rho_i^{t+1} &= \rho_i^t + \Delta t D_i^t \end{aligned}$$

This, though, is only first order accurate and is likely to be not very stable unless very small time steps are used. A scheme which is second order accurate in time is a predictor-corrector scheme:

#### **Half-step**

$$\mathbf{x}_i^{t+1/2} = \mathbf{x}_i^t + 0.5\Delta t \mathbf{v}_i^t$$

$$\mathbf{v}_i^{t+\frac{1}{2}} = \mathbf{v}_i^t + 0.5\Delta t \mathbf{a}_i^t$$

$$\rho_i^{t+1/2} = \rho_i^t + 0.5\Delta t D_i^t$$

#### **Full-step**

$$\bar{\mathbf{x}}_i^{t+1/2} = \mathbf{x}_i^t + 0.5\Delta t \mathbf{v}_i^{t+1/2}$$

$$\bar{\mathbf{v}}_i^{t+\frac{1}{2}} = \mathbf{v}_i^t + 0.5\Delta t \mathbf{a}_i^{t+\frac{1}{2}}$$

$$\bar{\rho}_i^{t+1/2} = \rho_i^t + 0.5\Delta t D_i^{t+1/2}$$

$$\mathbf{x}_i^{t+1} = 2 \bar{\mathbf{x}}_i^{t+1/2} - \mathbf{x}_i^t$$

$$\mathbf{v}_i^{t+1} = 2 \bar{\mathbf{v}}_i^{t+1/2} - \mathbf{v}_i^t$$

$$\rho_i^{t+1} = 2 \bar{\rho}_i^{t+1/2} - \rho_i^t$$

The added complexity with the predictor-corrector scheme is that 2 values for each of the variables need to be stored (the initial value and the half-step value) and the neighbour searching needs to be based on the appropriate  $\mathbf{x}$  values.

We also need to choose a time step that will ensure stability. In this system the following time step constraints need to be satisfied:

$$\Delta t_{CFL} = \min \left\{ \frac{h}{|\mathbf{v}_{ij}|} \right\}$$

$$\Delta t_F = \min \left\{ \sqrt{\frac{h}{|\mathbf{a}_i|}} \right\}$$

$$\Delta t_A = \min \left\{ \frac{h}{c_0 \sqrt{(\rho/\rho_0)^{\gamma-1}}} \right\}$$

$$\Delta t = C_{CFL} \min\{\Delta t_{CFL}, \Delta t_F, \Delta t_A\}$$

With an appropriate value of  $C_{CFL}$  being in the range 0.1 - 0.3 (top end of the range is quicker, but less stable/accurate). Ideally a dynamic time step should be used, but as long as the speed of sound is high enough, it should dominate in the example you are required to solve. You can therefore use the following fixed time step for your initial testing:

$$\Delta t = 0.1 \frac{h}{c_0}$$

#### **Other required relationships**

Another variable that you need to choose in the system is the relationship between the initial particle spacing,  $\Delta x$  and the characteristic smoothing length,  $h$ . You can use:

$$h = 1.3 \Delta x$$

Once you have the simulator working you can test the impact of changing this proportionality.

The initial mass of your particles also depends upon the initial particle spacing:

$$m_i = \Delta x^2 \rho_0$$

### Smoothing the Density/Pressure field

In SPH the integration of the continuity equation will result in not only variations in the density (and thus the pressure) based on the macroscopic velocity gradients, but also on local variations in particle spacing and velocity. This will eventually result in a very “rough” density and pressure distribution.

To get around this problem the density should be smoothed every 10 to 20 time steps. The following smoothing should be used:

$$\rho_i = \frac{\sum_{j=1}^n W(\mathbf{r}_{ij}, h)}{\sum_{j=1}^n \frac{W(\mathbf{r}_{ij}, h)}{\rho_j}}$$

Note that the sums MUST include the  $i = j$  combinations.

### Finding Neighbours

Virtually all of the above equations require summing over neighbouring particles. If we were to naively search over all possible neighbours for each of the points required in the calculation, then the overall complexity of the algorithm would be  $O(N^2)$  in the number of particles, which would be prohibitively expensive for all but the smallest problems.

We can get around this problem by exploiting the fact that the kernel has compact support and that we therefore only need to search for particles that are within  $2h$  of the particle that we are considering. Considering that the particles will also be quite evenly spaced, the best way to do this search is using a linked cell method.

The way this works is that the entire domain is divided into a grid of  $2h$  wide cells. A time step (or half time step in the predictor-corrector case) starts with each particle being assigned to their appropriate cell. This involves looping over all the particles and thus has a complexity of  $O(N)$ . The calculation for each particle involves looping over all the particles in its cell, together with all the particles in the neighbouring cells. Since the number of particles in these neighbouring cells does not depend upon the number of particles in the system, the calculations required to sum over all the neighbouring particles of a single particle is  $O(1)$ , with the calculations for all the particles thus being  $O(N)$ . The overall algorithm is thus  $O(N)$ .

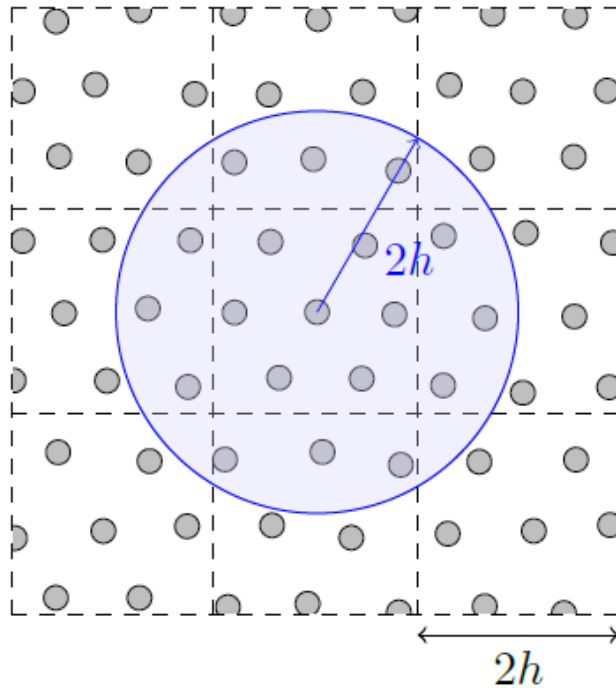


Figure 1: Searching using linked cell method

In order to aid the development of your program you have been given an implementation of the linked cell search method. This implementation is comparatively basic and does not include the ability to use different half-step locations. It will, though, show you the basics of how to implement this method and form a basis on which you can build and improve.

### Boundaries and initial conditions in SPH

Being a Lagrangian method, domain boundaries can be problematic to implement in SPH. An easy way to do this is to put a layer of fixed particles  $2h$  wide around the edge of the whole domain. These particles should interact with other particles in the same way as the fluid particles and should have a density (and thus also pressure) that is allowed to evolve in the same way as that of other particles. Their position should remain fixed, though, and their velocity should remain zero.

Despite the boundary particles possessing a pressure, fluid particles may occasionally leak out of the domain. The easiest thing to do is to delete them, but this results in a loss of fluid mass. You can push them back, but you need to watch out for what the appropriate density to give them is (if you don't they will clump as the local density will be higher than the average density assigned to the particle). The final alternative is to give the walls a repulsive force that gets high enough at short range that the particles never leak. This can be effective, but needs tuning and can result in instabilities if done incorrectly. Note that in order to ascertain if a particle has leaked you do need to know the location of the boundaries. Think about storing boundaries in their own data structures, as this will be especially useful for implementing arbitrary shaped boundaries.

The figure below gives the initial configuration that should be used for these simulations. In these simulations the following initial particle spacing is used:

$$\Delta x = 0.2 \text{ m}$$

Remember to make  $\Delta x$  a variable that can be changed in the simulation configuration so that you can easily carry out a convergence test. A warning when placing the points is that the code is likely to crash if there are 2 points that are either on top of one another or close to being on top of one another. I would therefore recommend writing a function which checks for concurrent or near concurrent points and run this function after the points have initially been placed, but before the simulation runs. Make sure that you only remove one of any two near concurrent points! Also remember that if a fluid particle is near concurrent to a wall particle it is the fluid rather than the wall particle that must be removed.

The simulations will also be of water and so the appropriate physical properties should be used ( $\rho_0 = 1000 \frac{kg}{m^3}$  and  $\mu = 0.001 Pa s$ ).

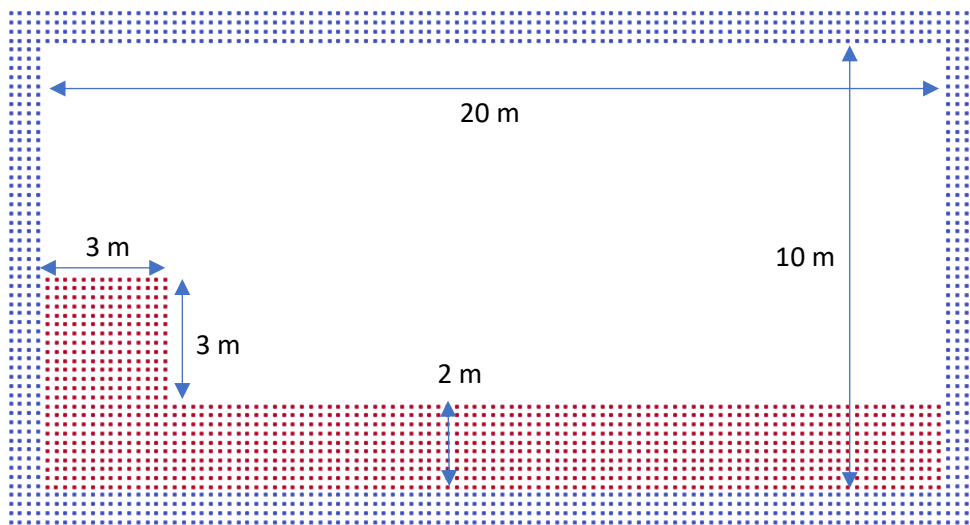


Figure 2: Initial configuration for the simulation. Red are mobile fluid particles and blue are fixed boundary particles.

## Code Structure

With your main simulation code you should allow the geometry, initial fluid layout and physical parameters to be entered in a relatively user friendly manner (this can be either via an input file or hard-coded). The simulation code should output the current state of the simulation at specified output time intervals (note that the output time interval will typically be much larger than the simulation time step). The simulation code must be written in C++.

There should be a separate program which loads the output files and performs the required visualisation and post-processing on them. This can be written in Python or C++. A useful way of visualising data is using Paraview. We have given you a short piece of code that is able to generate Paraview compatible files, though you are free to choose how you wish to visualise the outputs.

## What is required of each team?

### Tasks

- 1) Create an SPH simulator that can run based on the initial problem geometry and conditions described above. These simulations should run until  $t = 30$  s. Start by implementing a simple forward Euler timestep method, for which you may need to use a small timestep.
- 2) Output particle data to file in such a form that it can be used for subsequent post-processing. Remember that the time steps will be quite small and so only produce outputs at a user set simulation time output interval.
- 3) Create a post-processing code to plot the particle data stored in the output files produced by the SPH simulator. The output from this code should be an animation of the simulation results, saved in a sensible format.
- 4) Improve your time-stepping method by implementing the predictor-corrector scheme described above. Note that you should save your functioning Forward Euler version as a separate file; you are not expected to have both time-stepping methods implemented in the same code.
- 5) Measure the wave height and location of the crest as a function of time. What is the sloshing interval across the domain and how does this compare to the expected shallow water wave speed of  $\sqrt{gh_0}$  where  $g$  is gravity and  $h_0$  is the average initial water depth? Why might there be a discrepancy?
- 6) Simulate different shaped domains, especially ones with non-vertical and horizontal walls. Try and simulate a shoaling wave (e.g. running up onto a beach).
- 7) Carry out a convergence study in which the resolution ( $\Delta x$ ) of the simulation is varied. You will need to decide upon a suitable metric for the error (for example, timescale for several wave transits). What is the order of convergence?

Note that you are not expected to complete all of these tasks, though you should complete tasks 1-3 as a minimum. The SPH algorithm outlined in this description is a “bare bones” one and there are many ways in which the performance and stability can be improved. Extra credit will be given for enhancements to the code and so I would recommend looking in the literature for some of these improvements starting with the papers of J.J. Monaghan who invented the SPH method.

### ***Recommendation for how to initially proceed***

I would recommend that you quite rapidly decide upon a data structure for the particles. This can be based directly on the neighbour search code that you have been given for the Forward Euler time-stepping. Once the data structure has been agreed, you can split the team into one group to implement the numerical solver and another group to handle data output and rendering. Code to write the data to file and the post-processing code can be developed without a functioning SPH simulator, using just the initial positions of the particles, which can be defined by the code given. It is important to get both aspects developed at the same time as you can't tell if the code is working correctly if you can't see the results.

In Task 4 you will need to modify this data structure to include the need to store two values of position, velocity and density for each particle in order to implement the predictor-corrector time-stepping scheme, as well as modifying the neighbour search to use the appropriate particle positions. You do not need to modify the output routine or post-processing software for the predictor-corrector time-stepping as only the full timestep state needs to be outputted.



## **Technical requirements**

You should use the assigned GitHub repository exclusively for your project

Your main simulation software must be written in C++,

You are not allowed to import any external libraries for the main simulation code beyond the standard ones associated with C++ or the parallel libraries for MPI or openMP.

Your code should demonstrate its health and sustainability through a method such as automated testing. You may use the testing framework supplied in your starter repository or any alternative, but should use Travis as your CI engine.

## **Marking Scheme**

### **Software (70 marks)**

#### *Functionality (30 marks)*

This will be assessed based on the code's ability to perform the tasks 1-6 outlined above. The required functionality therefore includes:

- 1) The ability to run successfully for 30 seconds of simulation time.
- 2) The ability to output the results to file.
- 3) A post-processor that can load the outputs and save them in a suitable format for rendering as an video/animation
- 4) The implementation of a 2<sup>nd</sup> order accurate time-stepping scheme.
- 5) The extension of the post-processing to include the tracking of wave velocity.
- 6) The ability to simulate other domain geometries.

#### *Performance (20 marks)*

This mark will be assessed based on a combination of the correctness of the solution and the speed of the code. Included in the assessment of performance will be the convergence analysis carried out in task 7.

#### *Sustainability (20 marks)*

As with all software projects, you should employ all the elements of best practice in software development that you have learned so far. A GitHub repository will be created for your project to host your software. The quality and sustainability of your software and its documentation will be assessed based on your final repository and how it evolves during the week. Specific attention will be given to the following elements:

- 1) Installation and usage instructions
- 2) Documentation (via a suitable format).
- 3) Coding style
- 4) Quality and coverage of automatic testing framework
- 5) General repository usage
- 6) Licensing

#### **Presentation (20 marks)**

On Friday afternoon, you must demonstrate your software to staff during a 90 minute interactive session. Each assessor will ask you to demonstrate one of the following five elements of your software.

- 1) Describe the algorithm used to carry out the task, including any enhancements that you have made to the code. Particular focus should be given to how the particle containment was addressed, as well as stability/smoothness of the solution was ensured.
- 2) Demonstrate how the user would interact with the code, including setting the input conditions and post-processing the output.
- 3) Describe and discuss the results obtained, including how they compare with simplified analytic prediction for wave speed. Why might discrepancies exist?

- 4) Describe the quantification of the performance of the code by means of the convergence analysis. How does this convergence compare to the expected convergence? How else was the code performance quantified and assessed?
- 5) How did you address the problem of flow in arbitrary shaped domains? How were the boundaries specified? How was both the placement of the boundary particles and the containment of the fluid particles addressed?

***Teamwork (peer assessment; 10 marks)***

After the presentations, you will complete a self-evaluation of your group's performance. This will inform the teamwork component of your mark. Please refer to the ACSE-4 guidelines for more information about the assessment of teamwork.