Table of contents

Appendix A

Rewriting of the spatial derivative

The particle approximation of the spatial derivative $\nabla \cdot u(\boldsymbol{x})$ may be rewritten using two identifiers derived with the divergence theorem. In this report the rewritten equations are used when taking the derivative of a scalar field with constant value over space and when deriving the Navier-Stokes equations. This is necessary to ensure that the derived field value is zero or to introduce a level of symmetry into the approximation. The alternate spatial derivatives given here are presented in [Liu; 2003] and [Monaghan; 2005] but not derived. The notation in this appendix is similar to the one used with the SPH method in the main report.

$$\left\langle u\left(\boldsymbol{x}_i\right)\right\rangle = \sum_{j=1}^{J} \frac{m_j}{\rho_j} u\left(\boldsymbol{x}_j\right) W_{ij} \tag{A.1}$$

$$\left\langle \nabla \cdot u\left(\boldsymbol{x}_i\right)\right\rangle = \sum_{j=1}^{J} \frac{m_j}{\rho_j} u\left(\boldsymbol{x}_j\right) \cdot \nabla W_{ij} \tag{A.2}$$

When rewriting the spatial derivative, the following two approximations of density are essential.

$$\rho_i = \sum_{j=1}^{J} m_j W_{ij} \tag{A.3}$$

$$\nabla \rho_i = \sum_{j=1}^{J} m_j \nabla_i W_{ij} \tag{A.4}$$

The identifiers (A.6) and (A.8) are derived using the "product rule" of the divergence theorem. The first identifier (A.6) is easily found using this rule:

$$\nabla \cdot \left(\rho u\left(x\right)\right) = \left(\nabla \rho\right) \cdot u\left(x\right) + \rho\left(\nabla \cdot u\left(x\right)\right) \tag{A.5}$$

$$\nabla \cdot u(x) = \frac{1}{\rho}\left[\nabla \cdot \left(\rho u(x)\right) - \left(\nabla \rho\right) \cdot u(x)\right] \tag{A.6}$$

The second identifier (A.8) is found using the same rule with a different input:

$$\nabla \cdot \left(\frac{u(x)}{\rho}\right) = \frac{1}{\rho}\left[\nabla \cdot u(x)\right] + \left[(-1)\frac{1}{\rho^2}\nabla \rho\right] \cdot u(x) \tag{A.7}$$

$$\nabla \cdot u(x) = \rho\left[\nabla \cdot \left(\frac{u(x)}{\rho}\right) + \left(\frac{u(x)}{\rho^2}\right) \cdot \nabla \rho\right] \tag{A.8}$$

The identifiers are now used to rewrite (A.2). Starting with the identifier (A.6) Equations (A.4) and (A.1) are inserted and the result is rewritten into (A.11).

$$\nabla \cdot u(x_i) = \frac{1}{\rho_i}\left[\nabla \cdot \left(\rho_j \sum_{j=1}^{J} \frac{m_j}{\rho_j} u(x_j) W_{ij}\right) - u(x_i) \cdot \sum_{j=1}^{J} m_j \nabla_i W_{ij}\right] \tag{A.9}$$

$$\nabla \cdot u(x_i) = \frac{1}{\rho_i}\left[\sum_{j=1}^{J} m_j u(x_j) \cdot \nabla_i W_{ij} - \sum_{j=1}^{J} m_j u(x_i) \cdot \nabla_i W_{ij}\right] \tag{A.10}$$

$$\boxed{\nabla \cdot u(x_i) = \frac{1}{\rho_i}\left[\sum_{j=1}^{J} m_j \left[u(x_j) - u(x_i)\right] \cdot \nabla_i W_{ij}\right]} \tag{A.11}$$

The identifier (A.8) is used to derive the second alternative to (A.2) following the same procedure resulting in (A.12).

$$\boxed{\nabla \cdot u(x_i) = \rho_i\left[\sum_{j=1}^{J} m_j \left[\frac{u(x_j)}{\rho_j^2} - \frac{u(x_i)}{\rho_i^2}\right] \cdot \nabla_i W_{ij}\right]} \tag{A.12}$$

Appendix B

SPHysics F77 to F95

SPHysics was developed to simulate a wave flue and other similar problems. The original code is written in F77 and in the following chapter it will be explained how the code was updated to F95. The function and cause of the implemented changes are explained in detail together with a full index of the subroutines and modules. In general the original subroutine names have been kept and no changes have been made to the generated output files. The main objective with the rewriting was to get a full understanding of the workings of an advanced SPH code and to take full advantage of some of the advanced Fortran features available in F95. It is important to note that the following is not an attempt to rewrite the user guide of SPHysics 1.0 [Gesteira et al, 2007] and any questions to the original code must be sought there using this paper as a supplement.

SPHysics is written in Fortran 77 and there are three newer major releases of Fortran in 1990, 1995 and 2003. After initially reading the code it was decided to rewrite into Fortran 95 to take advantage of some of the new features. From now on the original SPHysics 1.0 is referred to as SPHysics F77 while the rewritten program is SPHysics F95 and any Fortran names are marked as modules or subroutines. During the rewriting it would also be time to get a better understanding of Fortran, the advanced SPH options available and how they were implemented. The main goal is to make a rewritten version of the 2-D part of the code as it is the one needed in the project. With the new features from F95 it is possible to make a leaner code that is more easily read for the beginner. The program Compaq Visual Fortran is used to compile the code as a free license is available for students at Aalborg University.

SPHysics consists of six programs and the code is able to handle problems in 2D and 3D. There are three programs/file groups available for each dimension organized as follows:

- **SPHysicsgen** is run initially and generates the geometry and input for the main program in the form of seven output files. SPHysicsgen is written in F77.

- **SPHysics** is the main program that computes the CFD problem using the SPH method and the files generated by SPHysicsgen. SPHysics is written in F77.

- The post processing is written in **MatLab** and uses the files generated by SPHysics to make a plot of the solution. Some of the MatLab routines prepare data for visualization in ParaView, an open source viewer. This program is not part of the report but more information is available at [ParaView; 2008]. All post processing of SPHysics F95 is done using MatLab the files are available on the CD-ROM.

To get the most of this chapter a basic understanding of Fortran is needed or a look-up source like [Chapman; 2004] must be available.

B.1 Modules

Modules became available with F95 and they are used in the rewritten code for two purposes. One is to organize the subroutines in fewer files (CONTAIN) and the other is to share variables between the subroutines (SAVE). How to use the two commands is depicted in Figure B.1. More information about Fortran modules and how to use them is available in [Chapman; 2004].

```
MODULE modulename1
IMPLICIT NONE

SAVE
real    :: a,b,c
a = 3.0
b = 4.0
END MODULE modulename1
```

```
MODULE modulename2
IMPLICIT NONE
USE modulename1
CONTAINS
c = sqrt(a*a+b*b)


END MODULE modulename2
```

Figure B.1.Two different properties of the modules are used in the rewritten code. The SAVE command is used to share variables between subroutines (to the left) and the CONTAIN command is used to organize subroutines and perform the computations.

The module PARAMETERS is made to share variables between the subroutines and modules thus replacing the redundant COMMON command used in the F77 version. The pros of this new method is that there is no risk of mixing two different variables as the variable name, type and precision is defined once and saved in the memory. It is also possible to use arrays without a fixed size with the ALLOCATE command thus saving space in the memory.

Modules are used extensively when rewriting between F77 and F95 as it makes it possible to keep the original structure of the program intact.

B.2 SPHysicsgen F77 to F95

The purpose of SPHysicsgen is to generate the geometry and initial conditions solved with SPHysics. The program has a variety of options and it is not the purpose of this appendix to run through them all but on a general level the following is possible:

- **Geometry** - Two different geometries of the problem domain is available; a beach and a box. Externally defined geometries are not supported.
- **Moving objects** - There are four types of moving objects. Two wave makers (Piston and Piston-Paddle), a gate and an initial wave defined somewhere in the problem domain.
- **Time stepping** – There are two different time stepping algorithms (predictor corrector and verlet) and it is possible to change the duration of the run and the size of the time steps.
- **SPH** - The SPH options are closely attached to the theory and are generally about the choice of h, viscosity conditions, boundary options and type of kernel functions.
- **Water** – Two ways are used to fill the water areas with particles. Both are depicted in Figure B.2.

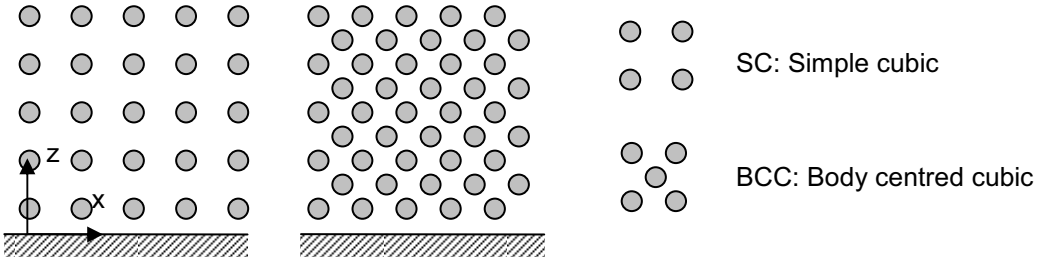


Figure B.2. The two possible ways to arrange the particles initial position in 2D is depicted above. In 3D the fifth particle of BCC is placed in the centre of the eight corner particles.

To getter a better picture of the possibilities study the flowcharts of Section B.6 and be aware that not all the options function as intended and much depends on how they are combined. A new user of SPHysicsgen F77 would be well advised to start with one of the five test cases given by [Gesteira et al; 2007].

B.2.1 Notes on SPHysicsgen F77 and F95

The main parts of SPHysicsgen are written in F77 and collected into one file with all the cons and pros of that approach. To help with future work on the F77 or F95 code a number of items are listed that turned up during the rewriting. Notes on how this is different in the rewritten F95 version are written as *italics*.

- Units used are (m, kg, s, etc.)
- No variables types are defined in SPHysics. They all depend on the first letter e.g. all variables that start with I, J, K, L, M, N are integers. *In the F95 version IMPLICIT NONE is used and all variables are defined as integer, real, array etc.*
- Variables are shared among the different subroutines with the COMMON statement available in the file common.2D. *This has been changed in the F95 version as the variables and their type are now defined in a module using the SAVE statement and implemented with the USE statement. Demonstrated in Section B.1.*
- Variables are normally defined with single precision throughout the code. *Nothing has been done to change this in the F95 version as it is possible to make a FORTRAN compiler define all variables with double precision if necessary.*

- All the subroutines are collected in one file. There is nothing wrong with this approach but it makes the file rather long and difficult to examine. *In the F95 version the files have been separated into different modules by type.*

- The input is not done at once but spread out between the different subroutines as READ(*,*) followed by WRITE(*,*) *variable* statements. This makes the input procedure cumbersome and easy to mess up. *In order to get a picture of all the possibilities the variables in the used subroutines have been collected in an input file. Yet, as not all subroutines are supported some may still be used the old way.*

The SPHysicsgen F77 is build to display the test cases available in the user's manual. It was necessary to make quite a few adoptions to the code in order to build a model capable of generating the wanted geometry. The individual subroutines and modules described on the next few pages are all working independently but had to be put together in new ways. The necessary custom made subroutines are collected in the module Waveflume_AAU. If further work is done with the code it might be advisable to spend time rebuilding the geometry generation into a more versatile tool. The following problems were experienced with the generated geometries.

- Particles were overlapping or large gaps appeared when the Fill_part subroutine was used several times to fill a problem area.

- The two subroutines Obstacle and Normals_Cals_2D do not work together. The limits of the generated arrays of boundary particles are broken.

- When choosing the dynamic boundary (*IBC* = 2) it is necessary to choose the BCC particle distribution depicted in Figure B.2. The code might run smoothly but post processing will show that the particles fall through the bottom.

- The generated boundaries must face the right way. This is the real difference between the three boundary subroutines (left, right and bottom). If for instance Boundaries_bottom is used as the lid of a box it must be rotated 180º for the normals to point in the right direction.

B.2.2 Subroutines in SPHysicsgen F77 and F95

The following is a list of the subroutines in the program and their functions.

Table B.1. This is a table of the different files and the subroutines they contain in the 2-D version of SPHysicsgen. In the 3-D version 2D is exchanged with 3D in the filename. [Gesteira et al; 2007]

Subroutine CALL name	**Subroutine properties**
Box	Creates box filled with particles (only two pieces of bottom)
Beach	Creates beach filled with particles
Boundaries_left	Left boundaries of beach/box
Boundaries_right	Right boundaries of beach/box
Boundaries_bottom	Bottom boundaries of beach/box
Wall	Creates wall (only in box)
Obstacle	Creates obstacle (does not work with dynamic boundaries, IBC=1)
Wavemaker	Creates one/several paddles
Gate	Creates a gate
External_geometry	Imports geometry (not working)
Fluid_particles	Initial distribution of fluid
Drop	Creates area of particles (each particle position is inserted manually)
Set	Generates a limited area of particles
Fill_part	Fills cubic area with particles
Wave	Generates an initial wave moving in the x direction
Pos_veloc	Calculates initial particle position and speed
Pressure	Calculates initial particle pressure
P_boundaries	Assigns density to boundary particles
Correct_P_boundaries	Corrects pressure at boundaries
Normals_calc_2D	Calculates normals to boundary particles
Normals_filewrite	Writes normals to file
Tocompile_ifort	Prepares SPHysics for compilation using IFORT (Linux)
Tocompile_gfortran	Prepares SPHysics for compilation using gFortran (HP)
Tocompile_cvf	Prepares SPHysics for compilation using Compaq Visual Fortran (MS Windows)

B.2.3 Modules in SPHysicsgen F95

Table B.2. List of modules and the subroutines they contain. The function of the different subroutines has not been changed.

Module name	Subroutine file name *.f	Description
BoundaryParticles	Boundaries_left Boundaries_right Boundaries_bottom	Creates boundary particles
Calculations	Pos_veloc Pressure P_boundaries Correct_p_boundaries	Initial computations for generated particles
Checking	Position_check	Checks particle position
Compile	Tocompile_gfortran Tocompile_ifort Tocompile_cvf	Compiler option for MS Windows, Linux and HP
FluidParticles	Fluid_particles Fill_part Set Drop Wave	Creates area with fluid particles
Geometry	Box Beach External_geometry	Initiates geometry
Input	Inputdata	Module created to control the input more easily
MovingObj	Gate Wavemaker Raichlenwedge_particles	Moving objects
Normals	Normals_Calc_2D Normals_FileWrite_2D	Calculates and exports normals to output file
Obstacles	Wall Obstacle	Creates wall/obstacle in flume
Parameters	-	Parameters used in SPHysicsgen
Waveflume_AAU	Boundaries_bottom_flume Boundaries_right_flume Fill_part_flume Cutaway	Customized subroutines to build the flume from the AAU wave laboratory

B.3 SPHysics F77 to F95

The Fortran language has undergone a great deal of development between the two versions F77 and F95. Newer versions of Fortran are able to use most of the old or redundant commands although there are parts of the code that has to be changed. In the previous sections a number of shortcomings in F77 were listed together with the changes implemented to remedy these problems. The same work has been done to the code of SPHysics.

B.3.1 Notes on SPHysics F77

The main parts of SPHysics are written in F77 with all the cons and pros of that approach. To help with future work on the F77 or F95 code a number of items are listed that turned up during the rewriting. Notes on how this is different in the re-written F95 version are written as *italics*.

- Units used are (m, kg, s, etc.)
- Smoothing length *h* has a constant value throughout the program. This is necessary when using the linked list algorithm [Liu; 2003]
- This code is not similar to the one presented by [Liu; 2003]. This code is chosen rather than [Liu; 2003] because it is constructed to be used on wave flumes.
- Be aware that a number of methods have calculations spread across several different subroutines. Constants are calculated in getdata_2D and used again in for instance kernels and viscosity.
- The variable *coef* in the INDAT file is not used to calculate the smoothing length it is instead a variable for the equation of state recommended to have the value [16; 40]
- The 2-D version is a downgraded version of the 3-D code. For some subroutines a better explanation is available in the 3-D version and some variables have not been removed although they are now obsolete. *An effort has been made to track all these redundant variables but some may still be left in the F95 version.*
- No variables types are defined in SPHysics. They are all depending on what type they have at the first input. *In the F95 version IMPLICIT NONE is used and all variables are defined as integer, real, array etc.*

- Variables are shared among the different subroutines with the COMMON statement available in the file common.2D. *This has been changed in the F95 version as the variables and their type are now defined in a module using the SAVE statement and implemented with the USE statement.*

- Variables are normally defined with single precision throughout the code. *Nothing has been done to change this in the F95 version as it is possible to make a FORTRAN compiler define all variables with double precision if necessary.*

- In the original code the same subroutine name is used multiple times shown on the flowchart Section B.6 as (n) where n are the number of options. There are for instance four different kernel subroutines that are all called kernel. Remember to remove the superfluous files before commencing a build. *In F95 the subroutines has been separated using the IF statement.*

B.3.2 Notes on SPHysics F95

The main parts of SPHysics are rewritten in F95 with all the cons and pros of that approach. All the notes given in Section 0 are still valid with the noted changes. To help with future work on the new F95 code a number of items are listed that were implemented in the rewritten version. The changes in the used modules and subroutines are explained in this section. Notes on the changes from the original version are written as *italics*.

- The file extension of F95 files written in the Fortran free format is *.f90 as the F77 files are written in fixed format *.f their content has been copied to new files.

- It is easy to restart the computing of a problem by replacing the IPART file with the PART file generated as output. Descriptions of different files are available in [Gesteira et al, 2007].

- All variables used by more than one subroutine are defined in the module Parameters together with a short explanation of their role in the program. They are grouped together depending on when they are read from the input files or which part of the code they belong to.

- Variable type has been defined with IMPLICIT NONE and arrays are defined as allocatable. Most arrays are allocated in getdata_2D and deallocated at program termination in the main SPHysics95 file. *This has been done to allow a better control of the array size i.e. how much memory used. In the*

original code the size was fixed at 3,200,000 particles in COMMON.2D, even if a lower number was used.

- Using allocatable arrays together with the link list is special as the number of cells may change when particles moves around. Therefore it is necessary to reallocate the size of the link list arrays during the time stepping.

B.3.3 Subroutines

There are twenty-two subroutines in the original SPHysics program which are listed and described in Table B.3. A flowchart of the relations between the different subroutines is given in Section B.6. The number and use of the subroutines is not changed in the rewritten F95 version of SPHysics, yet the way the subroutines share the variables with Common.2D is changed. Note that several subroutines share the same CALL name although they are different files.

Table B.3. This is a table of the different files and the subroutines they contain in the 2-D version of SPHysics. In the 3-D version 2D is exchanged with 3D in the filename. [Gesteira et al; 2007]

Subroutine CALL name	Subroutine FILE name(s) *.f	Subroutine properties
Sphysics	Sphysics	Main program
Common.2D	Common.2D	Not a subroutine. Used to share data between the subroutines with the COMMON and INCLUDE commands
Getdata	Getdata_2D	Reading data from input files generated by SPHysicsgen
Energy	Energy_2D	Creates output file with calculated energy
Ini_divide	Inidivide	Initializes the link list
Divide	Divide_2D	Creates the link list
Keep_list	Keep_list	Keeps the list of fixed boundary particles, only called once
Check_limits	Check_limits_2D	Detects particles outside the computational domain and relocates them
Poute	Poute_2D	Records information about the particles in output files
Shepard	Shepard_2D	Uses a Shepard filter on the results
Step	Step_predictor_corrector_2D Step_verlet_2D	Manages the marching procedure

Correct	Correct_2D Correct_sps_2D	Accounts for body forces, XSPH correction and SPS terms
Recover_list	Recover_list	Recovers the list of fixed boundary particles
Variable_time_step	Variable_time_step_2D	Recalculates the time step considering maximum inter particle forces, speed of sound and the viscosity
Ac	Ac	Controls the boundary particle movement and calls SELF / CELIJ
Self	Self_BC_dalrymple_2D Self_BC_monaghan_2D	Controls the interaction between particles inside the same cell
Celij	Celij_BC_dalrymple_2D Celij_BC_monaghan_2D	Controls the interaction between particles inside adjacent cells
Kernel	Kernel_gaussian_2D Kernel_quadratic_2D Kernel_cubic_2D Kernel_wendland5_2D	Calculates the particle – particle interaction according to the chosen kernel
Viscosity	Viscosity_artificial_2D Viscosity_laminar_2D Viscosity_laminar+SPS_2D	Calculates the viscosity terms depending on the chosen type
Monaghanbc	Monghanbc_2D	Accounts for Monaghans repulsive force between fluid and boundary particles
Movingobjects	Movingobjects_2D	Controls the moving objects
Updatenormals	Updatenormals_2D	Calculates the normals to the boundary particles
Movingpaddle	Movingpaddle_2D	Accounts for the paddle movement

When building the SPHysics executable file in the F77 version it is necessary to remove superfluous files from the Fortran workspace. For instance there are four different kernel subroutines all initiated with the same CALL therefore three off the files are removed. Furthermore two subroutine combinations are crucial for the computations depending on the choice of boundary conditions. The two different combinations of files are given in Table B.4 and all the files in a combination are necessary to make it run correctly. Note that it is not possible to model moving objects like a wave flume with dynamic boundary conditions in the current version of the code.

Table B.4. This table lists two different file combinations depending on the choice of boundaries.

Subroutine combination 1 Monaghan boundary conditions	Subroutine combination 2 Dynamic boundary conditions
MonaghanBC	Self_BC_Dalrymple_2D
Self_BC_Monaghan_2D	Celij_BC_Dalrymple_2D
Celij_BC_Monaghan_2D	-
UpdateNormals_2D	-
MovingObjects	-
Movingpaddle	-

The number and function of the subroutines has not been changed in SPHysics F95 and a description is found in Table B.3. The subroutines are now organized into several modules and it is no longer necessary to remove files from the Fortran workspace. Where it is feasible input and output variables have been designated to make the program structure more transparent. The reason this is not done with all subroutines is the large amount of variables being shared throughout the main loop. Furthermore are modules available to share these variables between subroutines.

B.3.4 Modules

Modules became available with F95 and they are used in the rewritten code. A list of the modules and the subroutines they contain are available in Table B.5.

Table B.5. List of modules and the subroutines they contain. The function of the different subroutines has not been changed although some subroutines has been combined into one using the IF statement

Module name	Subroutine file name *.f	Description
Parameters	-	Used to define and share variables between the modules. It also contains short descriptions of each variable
Getdata_2D	Getdata_2D	Data is read from input files
Linklist	Ini_divide	All the subroutines used to generate the link list
	Divide_2D	
	Keep_list	
	Recover_ list	
	Check_limits_2D	
Output	Energy_2D	All output subroutines
	Poute_2D	

Kernels	Kernel_gaussian_2d Kernel_quadratic_2d Kernel_cubic_2d Kernel_wendland5_2d	All particle – particle interaction is calculated in this module. Collected in one IF block the variable is *i_kernel*.
Calculations	Variable_time_step_2D UpdateNormals_2D Correct_2D Shepard_2D MonaghanBC_2D	Mix of different calculations necessary to compute the problem
CalcViscosity	Viscosity_artificial_2d Viscosity_laminar_2d Viscosity_laminar+sps_2d	Viscosity calculation collected in one IF block. The controlling variable is *i_viscos*.
Marching	Step_predictor_corrector_2D Step_verlet_2D Ac_2D	Time stepping and control of particle movement. Choice of time-stepping is collected in one IF block. The controlling variable is *i_algorithm*.
MovingObj	Movingobjects_2D Movingpaddle_2D	Moving objects
Particleinteraction	Self_bc_dalrymple_2d Self_bc_monaghan_2d Celij_bc_dalrymple_2d Celij_bc_monaghan_2d	The mutual relations between particles are calculated using the link list. Collected in two IF blocks. The controlling variable is *IBC*.

B.4 AAU Wave flume SPH model

It was necessary to make a custom set of files able to generate the wave flume used in the experiments. This collection of subroutines combined in the SPHysicsgen module Waveflume_AAU are capable of modelling the flume with a paddle and fill it with particles. The output is fully compatible with SPHysics F95 but will not work on the older F77 version. Apart from building the geometry of the flume it was necessary to alter the new subroutine movingpaddle to enable it to read the sampled paddle movements and use them as base for the computation in SPHysics.

Wave flume geometry

The geometry is build using the four subroutines listed as part of the Waveflume_AAU module in Table B.2. The size and shape of the flume is described in Section C.4 and in Figure C.8. The boundaries of the SPHysics model is build in nine pieces as it is specified in Figure B.3 and filled with water (particles) Figure B.4.

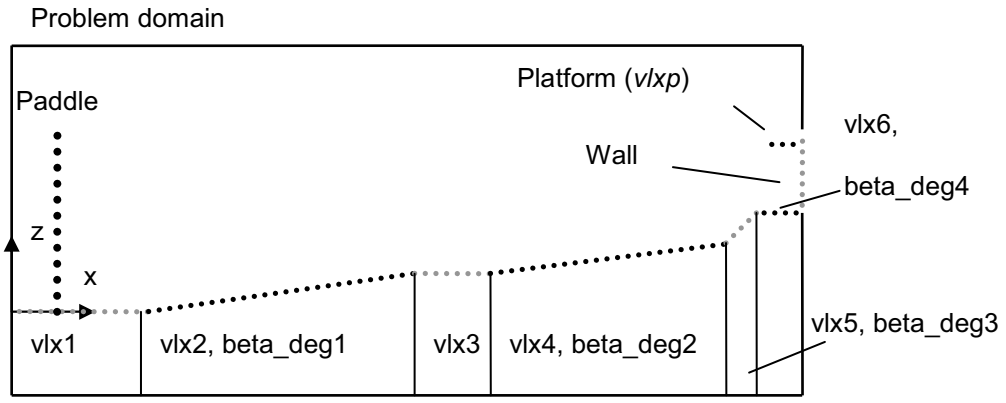


Figure B.3. Division of the wave flume into eight boundary pieces (vlx1-6, vlxp, wall) and one paddle. The annotation is corresponding to the one used in Waveflume_AAU.

The flume is filled with water using the fill_part_flume subroutine. The flume displayed in Figure B.4 is plotted with a spacing $dx,dz = 0.01$ and means a total of 75,669 particles if they are placed in a SC grid.

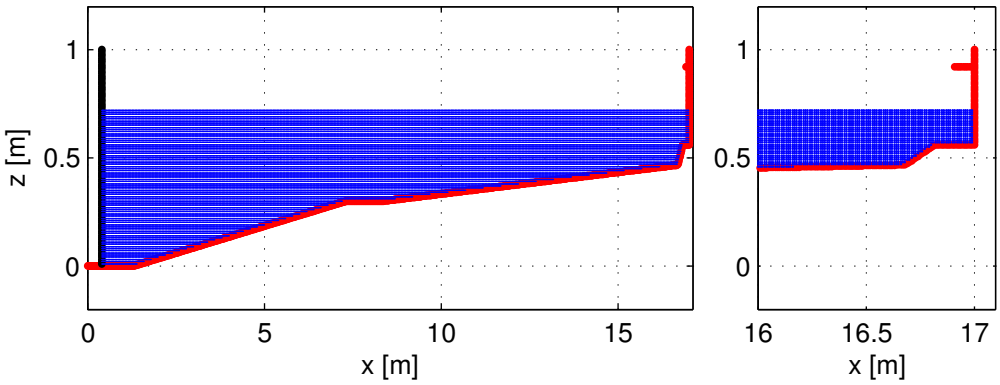


Figure B.4. Example of the generated wave flume to the left and to the right a close up on the end of the flume with the wall and platform.

It has been a priority to limit the number of particles. In order to do this water without any impact on the result has been excluded from the model. The height of the paddle is chosen so that the generated waves will not spill behind the paddle. Particles leaving the flume by spilling across the wall are excluded from the computation.

Paddle movement

The paddle is controlled by input sampled in the experiments presented in Appendix C the sample resembles a sinus wave as depicted in Figure B.5. The sampled signal is read in two different ways depending on whether or not variable time steps have been enabled.

- Variable time step off means that the position *xp* of the paddle through the whole sample may be calculated at once.

- Variable time step on means that *xp* is calculated in each time step by iterating from the measured result i.e. if the time is 5.0015 seconds the mean of the two positions at 5.0010 and 5.0020 is used. (sampled with 1000 Hz)

The subroutine controlling the paddle also needs the speed *up* of each paddle particle. As this was not measured the speed is calculated using a central difference scheme (B.1) when loading the position file in getdata_2D. An example of a signal and its derivate is depicted in Figure B.5.

$$f'(x_0) = \frac{f_1 - f_{-1}}{2dt} \tag{B.1}$$

where

dt is the timestep [s]

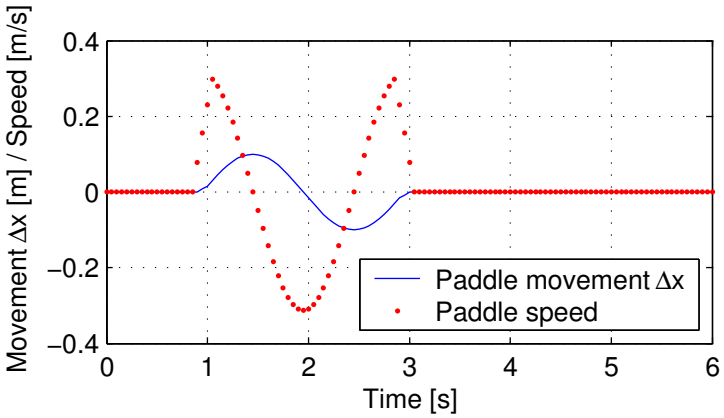


Figure B.5. An example of how a sample of the paddle movement and its time derivative (particle speed) might look. This is a generated example sampled with 20 Hz.

B.5 Conclusion and further work

The translation of SPHysics from F77 to F95 in this project has been performed in order to understand the rudimentary structure of an advanced SPH program and to prepare it for computing a virtual wave flume. It was demonstrated by the results in the main report that the F95 is stable and able to handle the specified problems. One drawback by using modules is the risk of collecting to many/long subroutines in the same module which makes it difficult to work with. With the experience in using Fortran gained during the past six months it would be possible to make it all more streamlined. A great deal of time was spend tracking the different variables and their importance. This work has not been concluded as only the variables shared between several subroutines are defined. But with the new structure of the code further work will be easier.

If more time was available it would be necessary to rid the code of some of the problems mentioned in this chapter and build a more flexible geometry generation system. Furthermore this project has only worked with a 2-D version of SPHysics. A rebuilding of the 3-D version would be easy to perform based on the collected experience and reusing the shared subroutines. Running 3-D computations would demand greater amounts of computational power and a parallelization might speed things up, as described by [Liu; 2003].

B.6 Flowchart SPHysics F77 and F95

The following pages contain flowcharts of the original SPHysics 1.0 code and the rewritten F95 version. The flowcharts are based on the SPHysics user guide [Gesteira et. al; 2007] and investigations of the code. The following flowcharts are included:

- **Flowchart page I** – The original SPHysics 1.0 code structure written in F77 together with the created output (files) and important lists (arrays).
- **Flowchart page II** – An updated version of SPHysics 1.0 written in F95 with redundant and obsolete commands removed. The whole code is now organized into modules described in Section B.3.4.
- **Flowchart page III** – The original SPHysicsgen 1.0 code structure written in F77 together with the created output (files). When the geometry is a box.
- **Flowchart page IV** – The original SPHysicsgen 1.0 code structure written in F77 together with the created output (files). When the geometry is a beach.

The signatures used on the flowcharts are explained in Figure B.6. Note that the necessary input files generated by SPHysicsgen are not presented here.

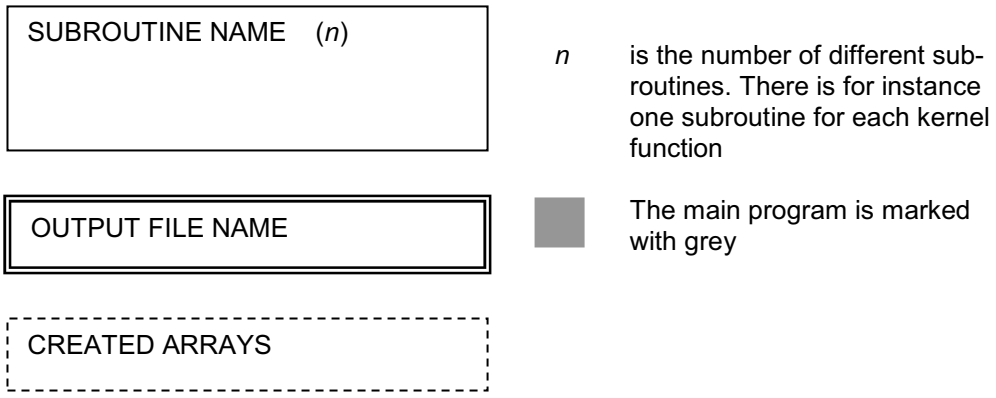
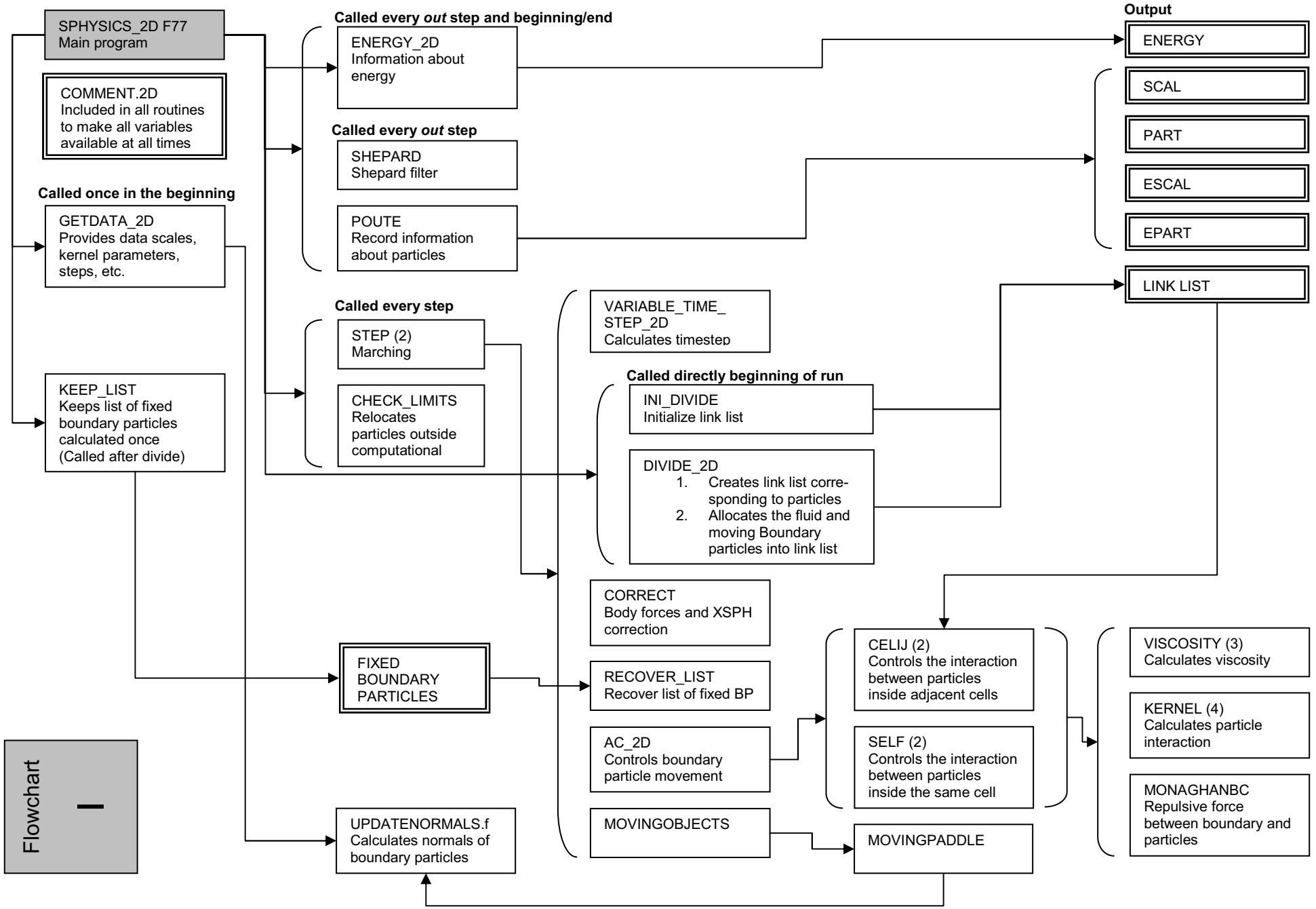
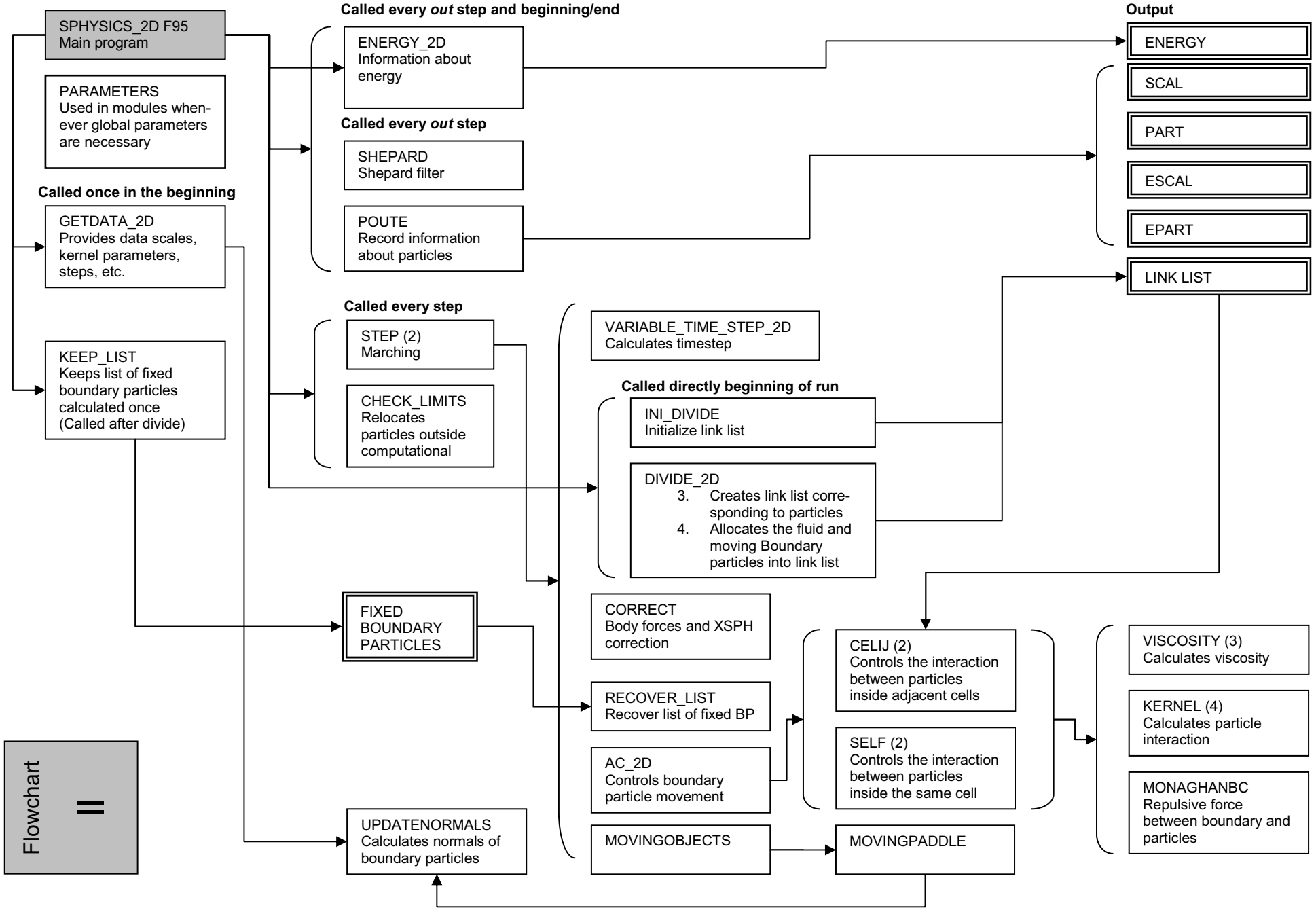
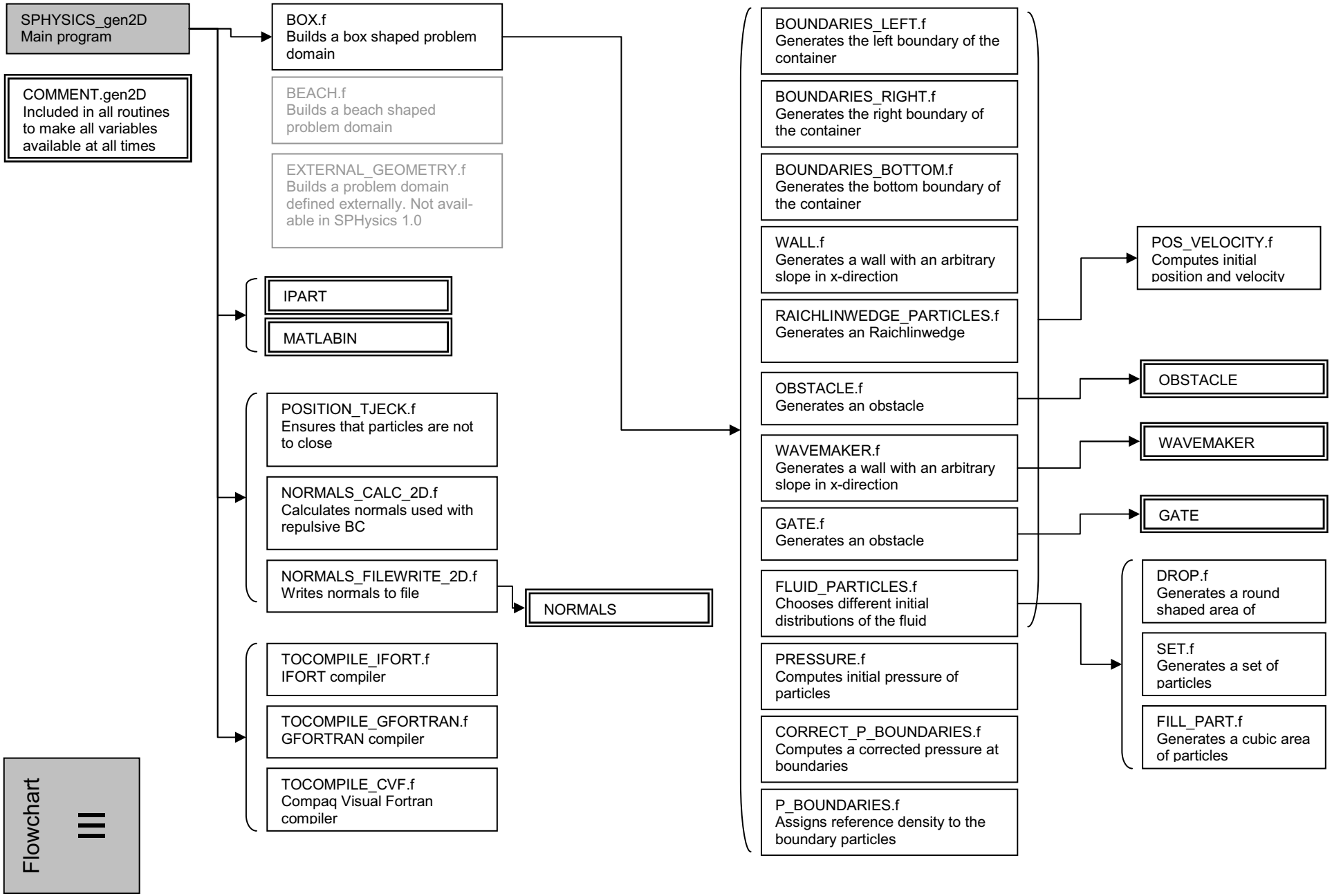
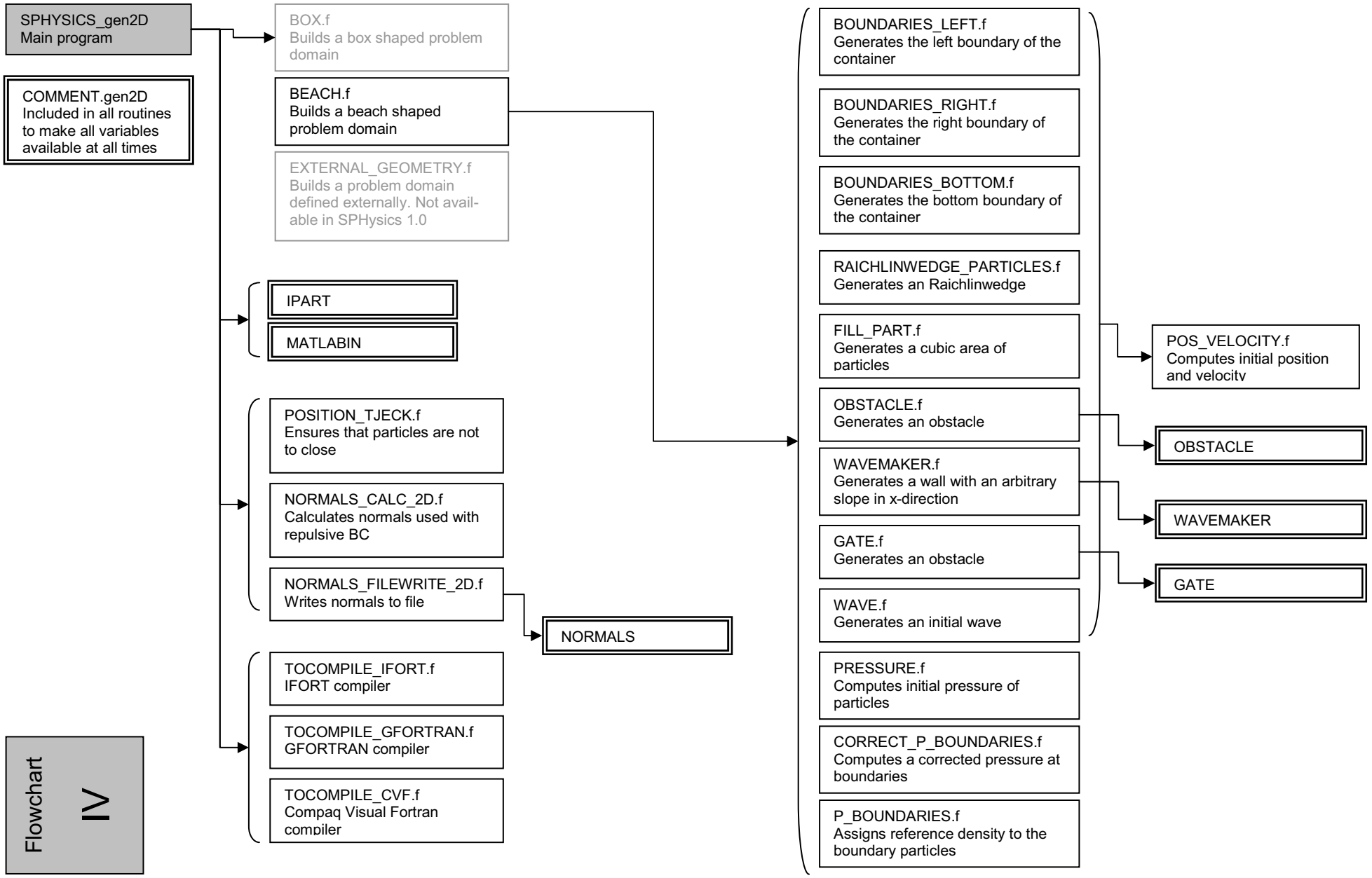


Figure B.6.The signatures used in the flowcharts to describe SPHysics code structure and how it has been rewritten in F95.


SPHYSICS_2D F77
Main program
COMMENT.2D
Included in all routines to make all variables available at all times
Called once in the beginning
GETDATA_2D
Provides data scales, kernel parameters, steps, etc.
KEEP_LIST
Keeps list of fixed boundary particles calculated once (Called after divide)
Called every out step and beginning/end
ENERGY_2D
Information about energy
Called every out step
SHEPARD
Shepard filter
POUTE
Record information about particles
Called every step
STEP (2)
Marching
CHECK_LIMITS
Relocates particles outside computational
VARIABLE_TIME_STEP_2D
Calculates timestep
Called directly beginning of run
INI_DIVIDE
Initialize link list
DIVIDE_2D
1. Creates link list corresponding to particles
2. Allocates the fluid and moving Boundary particles into link list
CORRECT
Body forces and XSPH correction
FIXED BOUNDARY PARTICLES
RECOVER_LIST
Recover list of fixed BP
AC_2D
Controls boundary particle movement
MOVINGOBJECTS
UPDATENORMALS.f
Calculates normals of boundary particles
MOVINGPADDLE
CELIJ (2)
Controls the interaction between particles inside adjacent cells
SELF (2)
Controls the interaction between particles inside the same cell
VISCOSITY (3)
Calculates viscosity
KERNEL (4)
Calculates particle interaction
MONAGHANBC
Repulsive force between boundary and particles
Output
ENERGY
SCAL
PART
ESCAL
EPART
LINK LIST
Flowchart –


SPHYSICS_2D F95
Main program
PARAMETERS
Used in modules whenever global parameters are necessary
Called once in the beginning
GETDATA_2D
Provides data scales, kernel parameters, steps, etc.
KEEP_LIST
Keeps list of fixed boundary particles calculated once
(Called after divide)
Called every out step and beginning/end
ENERGY_2D
Information about energy
Called every out step
SHEPARD
Shepard filter
POUTE
Record information about particles
Called every step
STEP (2)
Marching
CHECK_LIMITS
Relocates particles outside computational
VARIABLE_TIME_STEP_2D
Calculates timestep
Called directly beginning of run
INI_DIVIDE
Initialize link list
DIVIDE_2D
3. Creates link list corresponding to particles
4. Allocates the fluid and moving Boundary particles into link list
FIXED BOUNDARY PARTICLES
CORRECT
Body forces and XSPH correction
RECOVER_LIST
Recover list of fixed BP
AC_2D
Controls boundary particle movement
MOVINGOBJECTS
MOVINGPADDLE
UPDATENORMALS
Calculates normals of boundary particles
Output
ENERGY
SCAL
PART
ESCAL
EPART
LINK LIST
CELIJ (2)
Controls the interaction between particles inside adjacent cells
SELF (2)
Controls the interaction between particles inside the same cell
VISCOSITY (3)
Calculates viscosity
KERNEL (4)
Calculates particle interaction
MONAGHANBC
Repulsive force between boundary and particles
Flowchart
=


SPHYSICS_gen2D
Main program
COMMENT.gen2D
Included in all routines to make all variables available at all times
BOX.f
Builds a box shaped problem domain
BEACH.f
Builds a beach shaped problem domain
EXTERNAL_GEOMETRY.f
Builds a problem domain defined externally. Not available in SPHysics 1.0
IPART
MATLABIN
POSITION_TJECK.f
Ensures that particles are not to close
NORMALS_CALC_2D.f
Calculates normals used with repulsive BC
NORMALS_FILEWRITE_2D.f
Writes normals to file
NORMALS
TOCOMPILE_IFORT.f
IFORT compiler
TOCOMPILE_GFORTRAN.f
GFORTRAN compiler
TOCOMPILE_CVF.f
Compaq Visual Fortran compiler
BOUNDARIES_LEFT.f
Generates the left boundary of the container
BOUNDARIES_RIGHT.f
Generates the right boundary of the container
BOUNDARIES_BOTTOM.f
Generates the bottom boundary of the container
WALL.f
Generates a wall with an arbitrary slope in x-direction
RAICHLINWEDGE_PARTICLES.f
Generates an Raichlinwedge
OBSTACLE.f
Generates an obstacle
WAVEMAKER.f
Generates a wall with an arbitrary slope in x-direction
GATE.f
Generates an obstacle
FLUID_PARTICLES.f
Chooses different initial distributions of the fluid
PRESSURE.f
Computes initial pressure of particles
CORRECT_P_BOUNDARIES.f
Computes a corrected pressure at boundaries
P_BOUNDARIES.f
Assigns reference density to the boundary particles
POS_VELOCITY.f
Computes initial position and velocity
OBSTACLE
WAVEMAKER
GATE
DROP.f
Generates a round shaped area of
SET.f
Generates a set of particles
FILL_PART.f
Generates a cubic area of particles
Flowchart


SPHYSICS_gen2D
Main program
COMMENT.gen2D
Included in all routines to make all variables available at all times
BOX.f
Builds a box shaped problem domain
BEACH.f
Builds a beach shaped problem domain
EXTERNAL_GEOMETRY.f
Builds a problem domain defined externally. Not available in SPHysics 1.0
IPART
MATLABIN
POSITION_TJECK.f
Ensures that particles are not to close
NORMALS_CALC_2D.f
Calculates normals used with repulsive BC
NORMALS_FILEWRITE_2D.f
Writes normals to file
NORMALS
TOCOMPILE_IFORT.f
IFORT compiler
TOCOMPILE_GFORTRAN.f
GFORTRAN compiler
TOCOMPILE_CVF.f
Compaq Visual Fortran compiler
BOUNDARIES_LEFT.f
Generates the left boundary of the container
BOUNDARIES_RIGHT.f
Generates the right boundary of the container
BOUNDARIES_BOTTOM.f
Generates the bottom boundary of the container
RAICHLINWEDGE_PARTICLES.f
Generates an Raichlinwedge
FILL_PART.f
Generates a cubic area of particles
OBSTACLE.f
Generates an obstacle
WAVEMAKER.f
Generates a wall with an arbitrary slope in x-direction
GATE.f
Generates an obstacle
WAVE.f
Generates an initial wave
PRESSURE.f
Computes initial pressure of particles
CORRECT_P_BOUNDARIES.f
Computes a corrected pressure at boundaries
P_BOUNDARIES.f
Assigns reference density to the boundary particles
POS_VELOCITY.f
Computes initial position and velocity
OBSTACLE
WAVEMAKER
GATE
Flowchart IV

Appendix C

Wave flume experiments

The wave flume experiments were conducted in the period from October 30 to November 9 2007. The experiments were conducted at Aalborg University in the wave laboratory room no. L-133. The experiments and the used equipment are described in detail on the following pages.

The real life inspiration is the access platforms placed on offshore wind turbines to allow workers access to the machinery, cf. Figure C.1. The platforms are subject to impact from waves breaking against the substructure.

Figure C.1. To the left is an offshore wind turbine from Farm1 (2007) and to the right a close-up of the landing stage and the platform running around the exterior of the turbine substructure from Oceanatlas (2007).

The purpose of the experiments was to get a source of reference when using the program SPHysics F95 to compute a virtual wave flume. Therefore it is the intention of the experiments to generate a similar situation in a wave flume with waves breaking against a vertical structure and hitting a platform. The work to build a virtual SPH model of the flume and comparison with the experiment will expose

the current limits of the SPH method and clarify how well the method is able to model two-dimensional (2-D) situations with distortion of the free surface, impact against structures and turbulence.

C.1 Experiment expectations

The experiment is a pure test study and the size of the structure is chosen to fit the available wave flume and generate a situation where waves are breaking against a horizontal platform as depicted in Figure C.2. The setup is compared to a SPH model in scale 1:1 and only regular waves are generated at the paddle. To keep track of the wave it is necessary to measure the wave height in the flume with wave gauges. In order to measure the impact the wall must be equipped with pressure transducers. The problem is simplified into a 2-D model because only a 2-D version of SPHysics F95 is currently available and in two dimensions it is possible to work with a greater numerical discretization than currently possible in 3D.

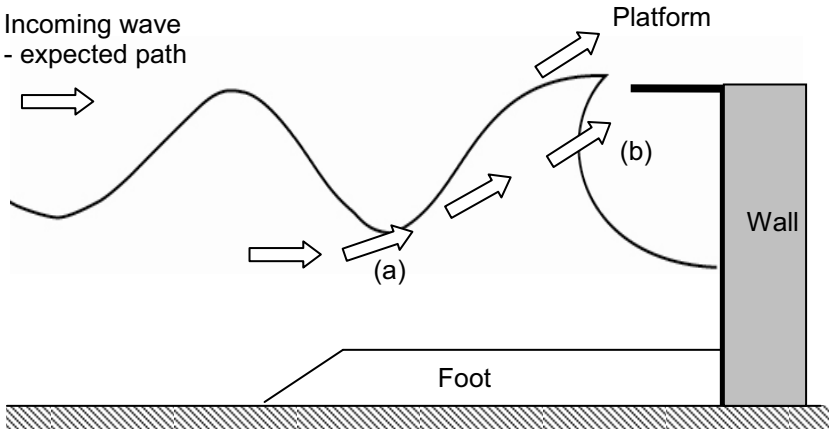


Figure C.2. In this figure, the situation desired in the experiment and the SPH model is depicted. The waves are chosen so they rise just before the wall (a) as they suddenly enter shallow water and hit the platform (b) from beneath. The pressure transducers are placed to measure this.

The wave impact on the platform will be instantaneous and might involve air being trapped between the structure and the water. Due to the declining water depth and the reflection of waves, the waves just before the platform rise and run up the wall. To track the whole movement it is desirable to place pressure transducers on the wall and platform.

The combination of wave properties necessary for a wave to travel through the wave flume and hit the platform is found with a test series with the chosen struc-

ture size and a water depth. The parameters are furthermore chosen to produce waves that do not break before they reach the structure in order to ease the comparison between experiment and model. As the experiment is started from a still water situation, the first waves dissipates and it is necessary to allow the wave generator time to gain up and generate a series of linear waves to get the required size of each wave.

The waves are generated at a paddle approximately 17 m from the structure and the greater part of the wave energy will be reflected when it reaches the wall. This is due to the vertical wall and the choice to make it a 2-D experiment and the same behaviour is expected of the SPH model. Because of the reflection, the wave gauges are placed in groups of three in order to enable them to distinguish incoming and reflected waves. In spite of this precaution it is not prudent to run the experiment longer than the time it takes the first waves to travel the flume thrice (i.e. between 30-60 seconds). Beyond this point the new waves together with the first and second group of waves will interact and create an unpredictable wave pattern.

C.2 Experiment types

The following cases are run as experiments for the chosen wave properties presented in Section C.5:

- Experiment no. 1: An experimental study of the variance and correlation of the sampled data
- Experiment no. 2: Train of regular waves rising and hitting the wall and platform with difference in water wave height H and period T.
- Experiment no. 3: Series of regular waves rising and breaking just before the platform (1-2 metres)

C.3 Experiment vs. SPH method

The following is a discussion of the differences between the experimental and the numerical model that may influence the final results.

The greatest difference is brought by the shear time it takes to perform experimental studies in the laboratory. The time consuming is partly due to the time it takes between each experiment before the flume is ready again and all prior generated waves have dissipated away. With the numerical model it is possible to build and

run a large number of tests hindered only by imagination and the computers available when the numerical model is built.

Geometry and discretization

It is possible to make a numerical model that has the same geometry and paddle as the experiment although the areas behind the wall and behind the paddle are removed as it makes no sense to model water that has no impact on the experiment. The amount of water spilling across the wall is small and does not change the water level in the flume. Furthermore as a time series of the paddle movement is sampled in the experiment it is possible to duplicate it exactly when computing the numerical solution.

Due to the shear size of the flume there is a limit on how many particles may be used and this will influence the impact readings. The generated waves in the flume are linear and non breaking until just before the structure and it is expected that even a relatively rough discretization will be able to model this situation. When the waves break and hit the structure, a huge number of particles are required to model the waves in every detail. This is not possible due to limits in computational power and one of the questions is how good an estimation of the real thing the rough SPH model is.

In the experiment it is only possible to measure the pressure in a limited number of points and a high sampling frequency is necessary to sample the impact. Furthermore, any measurements with the pressure gauges depend on their size as it must be an average of the pressure on the piece of structure they are covering. The SPH model has similar limitations, however with respect to the chosen time step and discretization.

Impact and water/air interaction

The most obvious difference between the numerical model and the experiment is the lack of air in the computational model. It would be possible to use the SPH method to model both air and water as the only difference between the materials are the mass and the speed of sound. This it not supported by the present edition of SPHysics but experiments have been conducted by [Colagrossi & Landrini, 2003] These experiments show that when modelling impact problems with air/water interaction a maximum pressure is expected as the trapped air tries to escape.

C.4 The experiment

The experiment was conducted at Aalborg University using the equipment available in the wave laboratories at the Department of Civil Engineering. The following is a description of the setup and the different types of equipment used during the generation and sampling of results.

C.4.1 Setup and equipment

In the experiment the problem is simplified into a 2-D situation depicted in Figure C.3. The size of the structure determines the choice of water depth and wave properties. A high wall will need deep water to generate the situation from Figure C.2 and increase the number of particles in the numerical model. On the other hand a low structure and low water depth will make it difficult not to generate waves that break long before they reach the structure. To address this problem it is chosen to place the platform 0.45 m above the bottom and place a shallow foot of pebbles that will help the waves to rise. Depending on the progress of the tests it will be possible to remove the pebbles and/or change the water depth to generate the requested situation and try alternatives. The wall and platform themselves are hard to alter as they must be immobile and are securely fastened with bolts to the wall and bottom of the flume.

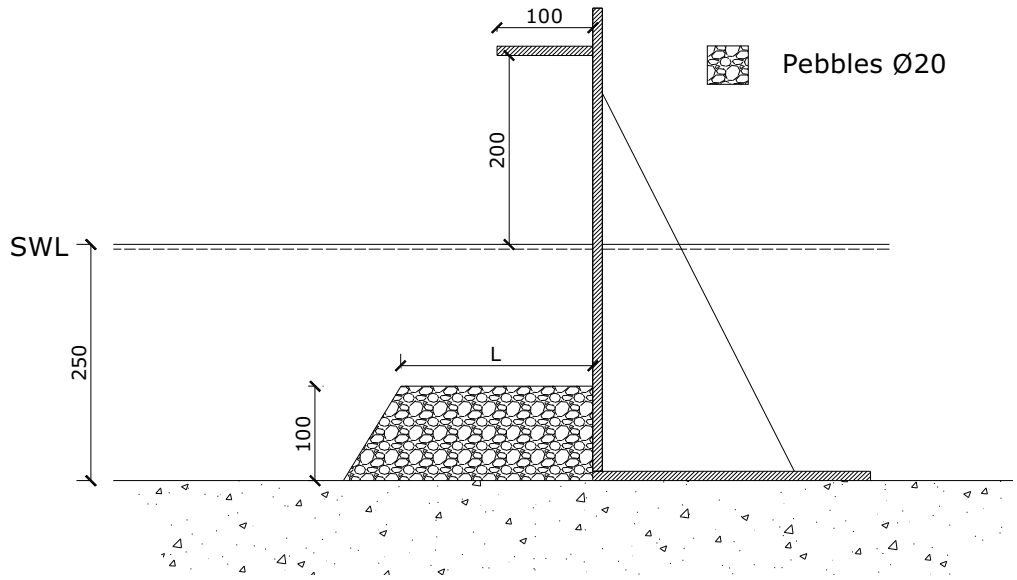


Figure C.3. A conceptual sketch of the experimental setup as placed in the wave flume. The structure consists of three pieces: a toe of pebbles to control wave breaking, a wall to break against and a platform above the SWL.

The model was built in the laboratory based on Figure C.3 and the following specifications.

- As SPHysics is unable to work with deformable solids the wall and the platform must be made of a material stiff enough to resist the wave forces.
- The wall and the platform must be safely secured in the channel.
- The pressure transducers are securely attached and present a smooth surface on the wall and platform for the waves to interact with.
- The toe of pebbles must be stable i.e. it is not moved in any significant way by the waves of a single sample.

The finished model with pressure transducers attached is shown in Figure C.5. The model was build of waterproof plywood boards (20 mm) secured with fittings to the walls and the bottom of the wave flume. The toe of pebbles was initially built with a length L of 0.30 m and a mean stone size of approximately Ø20 mm.

Figure C.4. Picture showing the wall placed in the wave flume and the two rows of strain gauges AA (left) and BB (right). To the right is a close up of three pressure transducers showing how they fit neatly into the wall surface.

Figure C.5. To the left a picture of the cross section before water and pebbles are added. To the right a picture of the experimental setup as it was placed in the wave flume with water and the toe of pebbles in place.

C.4.2 Wave flume and Paddle

The wave flume in room L-133 is 23 m long and depicted in Figure C.6. The flume generates waves with a piston paddle and is constantly rebuilt for new experiments. The geometry when running this chain of experiments is given in Figure C.8 and as an AutoCAD drawing on the CD-ROM.

Figure C.6. Pictures of the wave flume in room L-333 used for the experiments.

The wave flume operates with a piston paddle that generates waves by moving a piston in a horizontal movement pushing the water in front of it, cf. Figure C.7. The movement of the paddle is measured to be used in the computational models of the flume. The generated waves are measured at three locations in the flume shown in Figure C.8 in order to determine the size of the generated waves when they hit the wall.

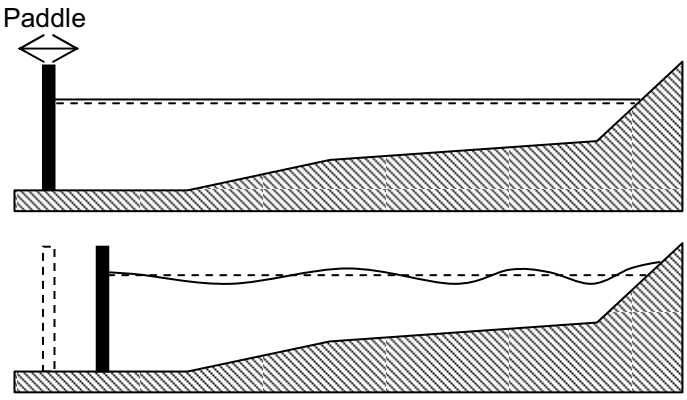


Figure C.7. The piston paddle generates waves by horizontal movement of the paddle.

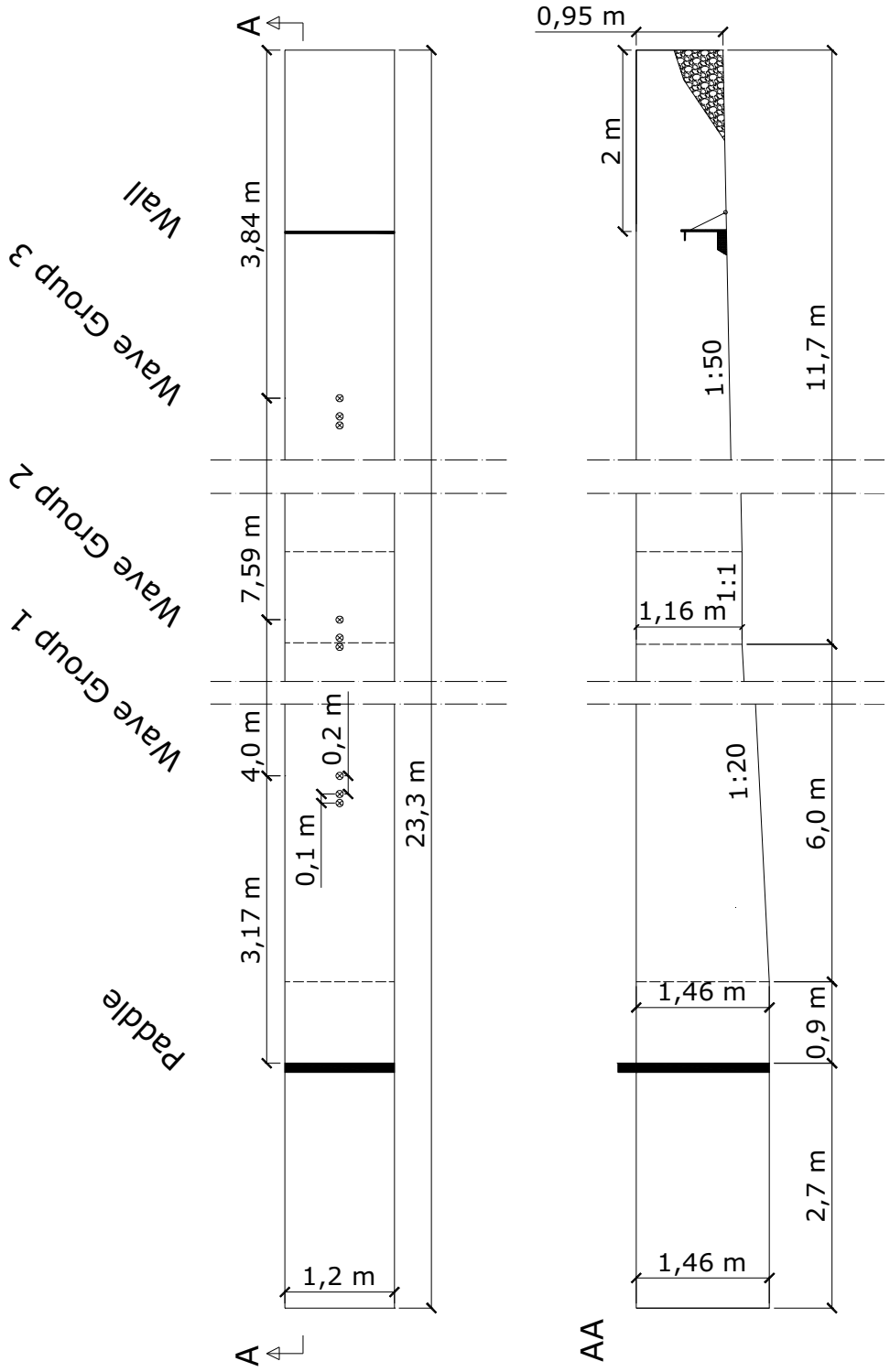


Figure C.8. A sketch of the wave flume with its dimensions and the locations of the wall and the three groups of wave gauges. The original is available as an AutoCAD drawing on the CD-ROM.

C.4.3 Pressure transducers

Pressure transducers are used to measure the wave impact at different points of the structure. Two different sizes are available in the laboratory Ø19 and Ø8 they are both shown in Figure C.9 and henceforward referred to as Transducer Ø19 and Transducer Ø8.

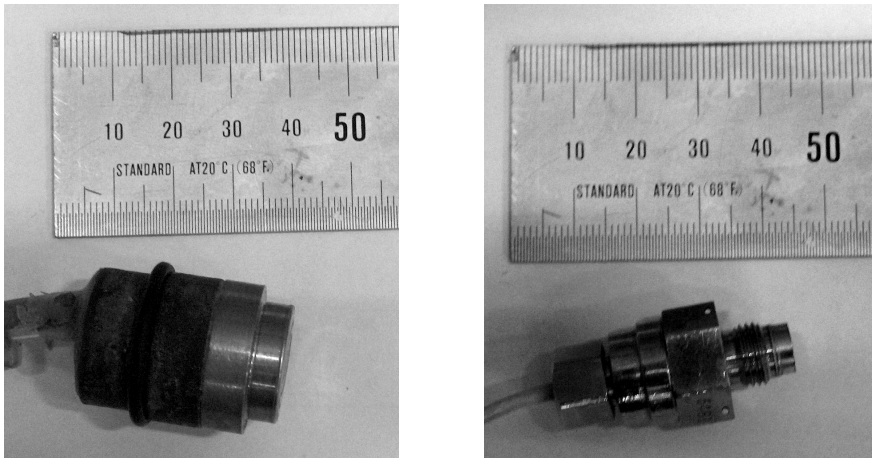


Figure C.9. The two different types of transducers used in the experiments. To the left is Transducer Ø18 and to the right is Transducer Ø7.

Table C.1. Technical specifications of the two different transducers

Size	Manufacturer	Model
Ø8	Kulite Semiconductor Products	HKM-134-375M- 1 Bar VG
Ø19	Unknown	Unknown

Transducer Ø19 measures pressure relatively to the atmospheric pressure while Transducer Ø8 measures against a reference in a chamber inside the pressure transducer. Because of the difference in size and measuring method it is decided to use two rows of strain gauges placed at similar points on the structure. The transducers are placed in the middle of the wave flume well away from any disturbances at the walls, cf. Figure C.10.

The number of pressure transducers is concentrated above SWL and close to the platform. This pattern is chosen to get a detailed measurement of the wave when it pushes up the wall and hits the platform. The number of the transducers and their corresponding channels is given in Table C.2.

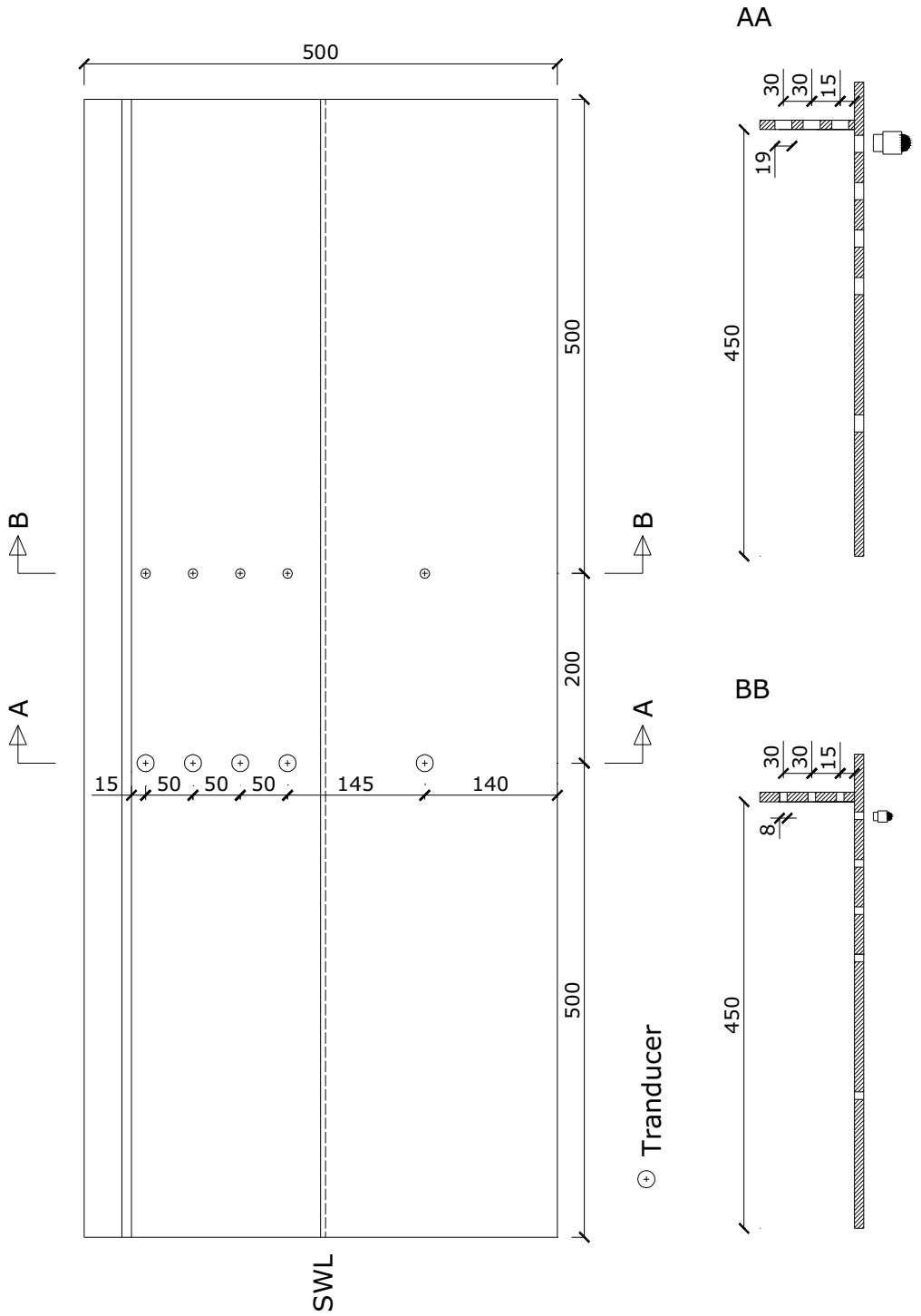


Figure C.10. This figure shows the location of the pressure transducers on the wall and the platform. Transducer Ø19 is placed in a row at AA and Transducer Ø8 is placed in a row at BB.

Table C.2. Transducer numbering and the corresponding channels they have when collecting the data. The transducer level is measured from the bottom and the transducers on the platform are placed from the wall and outwards i.e. Transducer 009 and 475 are closest to the wall.

Level	Row AA - Transducer Ø19		Row BB - Transducer Ø8	
[mm]	Channel No.	Transducer No.	Channel No.	Transducer No.
140	11	002	19	463
285	12	003	20	465
335	13	006	21	470
385	14	007	22	471
435	15	008	23	473
450	16	009	24	475
450	17	011	25	477
450	18	012	26	479

C.4.4 Wave gauges

Waves are measured at three separate locations specified in Figure C.8. The wave gauges at one location and the distance between them is depicted in Figure C.11. The wave gauges measures the change in currents between two rods and are thus dependent on the temperature and purity of the water. Furthermore, the submergence of the rods has an impact on the quality of the measurement. The wave gauges are therefore calibrated before every third sample.

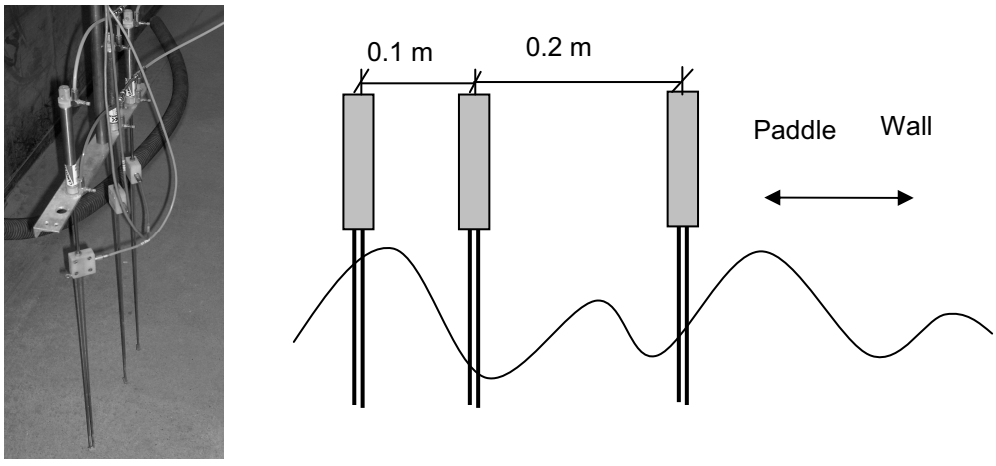


Figure C.11. A group of wave gauges is shown to the left and the distance between them is depicted to the right. Using the gauges together in a group makes it possible to separate generated and reflected waves in the post processing.

Table C.3. Gauge channels used when collecting wave data in the flume. They are placed in the middle of the flume and the position is measured from the upper end of the flume and down towards the paddle.

	Group No. 1		Group No. 2		Group No. 3	
	Channel No. [-]	**Position** [m]	**Channel No.** [-]	**Position** [m]	**Channel No.** [-]	**Position** [m]
A	01	15.73	04	11.73	07	4.14
B	02	15.63	05	11.63	08	4.04
C	03	15.43	06	11.43	09	3.84

C.4.5 Sampling equipment

There are a total of 26 channels involved in the experiments, are all listed in Table C.2 and Table C.3 except channel 10 which is connected to the paddle. All data sampling is conducted with WaveLab. The pressure transducers are connected to an amplifier and a data acquisition unit while the wave gauges are connected to a wave recorder, low-pass filter and a data acquisition unit. The following is a list of the equipment, what it was used for and the chosen sampling settings.

WaveLab 2.961 (Software)

WaveLab is a Windows® program for data acquisition and analysis in wave laboratories developed at Aalborg University. The program is used to sample all channels with a frequency of 1000 Hz and a range of ±10 V.

The sample frequency was chosen this high to measure the impact forces on the wall. A lower frequency would be enough for measuring the wave gauges (20 Hz recommended for wave measuring) but it is only possible to choose one frequency if simultaneous sampling is wanted on all channels. Post processing of the samples on page 52 showed that although the sample rate was adequate for the majority of wave gauges there was a problem with the peak measurement of gauge 009, 008, 473 and 475. It is advised that higher sampling frequencies is used in future experiments.

The range of ±10 V is the maximum possible and the amplification was chosen to use this range as effective as possible. The decisive factor considering the amplification for the pressure transducers was the high impact forces on the platform although they only happened a few times during each measurement. [Wavelab2; 2007]