SPH Modeling of Solitary Waves and Resulting Hydrodynamic Forces on Vertical and Sloping Walls

Safinaz El-Solh

Submitted under the supervision of

Dr. Ioan Nistor

In partial fulfillment of the requirements
for the degree of

Master of Applied Science in Civil Engineering

Department of Civil Engineering

University of Ottawa

October 2012

Acknowledgements

First and foremost, I would like to thank God for giving me the strength and patience that have allowed me to persevere. I sincerely thank my family for always showing me their unconditional support and love in all aspects of my life: my mother and father, Rashiqa and Wesam, for raising me to be who I am today and my brother and sister, Abdel-Aziz and Dana, for showing me that I can do anything I set my mind to. I love you all.

My deepest and heartfelt appreciation is extended to Dr. Ioan Nistor for introducing me to the world of Coastal Engineering and continuing to show me how to unlock all my potential as a professional. Ever since the first course I have taken with Dr. Nistor, he has never failed to deliver great advice, extensive knowledge and an understanding of what being an exemplary professor is. Thank you for all your support.

I would also like to thank Phil, my "SPH team", for kindly sharing with me his knowledge of using SPHysics and providing me with technical guidance whenever I needed it.

I would like to acknowledge the kind assistance and data that I have received from Dr. Miguel Esteban and Dr. Andreas Kortenhaus.

I would like to thank all the professors and staff of the Faculty of Civil Engineering at the University of Ottawa for all their guidance and great efforts. I would also like to extend my gratitude to all the graduate students of water resources that I have had the pleasure of meeting who have made my university experience richer.

Last, but most definitely not least, I would like to express how grateful I am to have a wonderfully dynamic group of friends that have shown me endless support, love and encouragement: the 'Ottawa Gang' for always believing in me and my 'Internationally Scattered Gang' for being so involved in my life that it never feels like you are away. I would have never been able to accomplish this without all of you.

Abstract

Currently, the accurate prediction of the impact of an extreme wave on infrastructure located near shore is difficult to assess. There is a lack of established methods to accurately quantify these impacts. Extreme waves, such as tsunamis generate, through breaking, extremely powerful hydraulic bores that impact and significantly damage coastal structures and buildings located close to the shoreline. The damage induced by such hydraulic bores is often due to structural failure. Examples of devastating coastal disasters are the 2004 Indian Ocean Tsunami, 2005 Hurricane Katrina and most recently, the 2011 Tohoku Japan Tsunami. As a result, more advanced research is needed to estimate the magnitude of forces exerted on structures by such bores.

This research presents results of a numerical model based on the Smoothed Particle Hydrodynamics (SPH) method which is used to simulate the impact of extreme hydrodynamic forces on shore protection walls. Typically, fluids are modeled numerically based on a Lagrangian approach, an Eulerian approach or a combination of the two. Many of the common problems that arise from using more traditional techniques can be avoided through the use of SPH-based models. Such challenges include the model computational efficiency in terms of complexity of implementation. The SPH method allows water particles to be individually modeled, each with their own characteristics, which then accurately depicts the behavior and properties of the flow field. An open source code, known as SPHysics, was used to run the simulations presented in this thesis. Several cases analysed consist of hydraulic bores impacting a flat vertical wall as well as a sloping seawall. The analysis includes comparisons of the numerical results with published experimental data. The model is shown to accurately reproduce the formation of solitary waves as well as their propagation and breaking. The impacting bore profiles as well as the resulting pressures are also efficiently simulated using the model.

Table of Contents

List of Figures

List of Tables

List of Symbols

a	Amplitude of wave (m)
b	Width of wall (m)
B	Constant in Tait's equation of state
c	Speed of sound in water (m/s)
c_0	Reference speed of sound in water at the reference density ρ_0 (m/s)
c_i	Speed of sound in water at particle i (m/s)
c_w	Wave celerity near wall (m/s)
C	Constant

d	Water depth (m)
d_o	Initial water depth (m)
dx	Particle spacing along x-axis (m)
dz	Particle spacing along z-axis (m)
dA	Tributary area of sampling region (m)
e	Internal energy
f_i	Force per unit mass at particle i
$f(x)$	Function at position x
F_h	Horizontal force (N)
F_l	Normalized force on wall (N) (Ramsden 1996)
$F_{h,imp}$	Horizontal impact force (N) (Cuomo *et al.* 2010)
$F_{h,qs}$	Horizontal quasi-static force (N) (Cuomo *et al.* 2010)
$\boldsymbol{F}$	Acceleration due to external forces in momentum equation (m/s^2 or N/kg)
g	Gravitational acceleration (m/s^2)
$\boldsymbol{g}$	Gravitational acceleration vector (m/s^2)
h	Kernel smoothing length (m)
h_b	Water depth at wave breaking (m)
h_w	Water depth near wall (m)
h_o	Initial water depth (m)
H	Wave height (m)
$H_{m,o}$	Significant wave height (m)
H_o	Initial wave height (m)
$H1$	Impoundment depth (m)
$H2$	Water depth downstream of dam (m)
L	Wavelength (m)
$L_{(hs)}$	Wavelength at toe of structure (m) (Cuomo *et al.* 2010)

L_m	Wavelength in deep-water conditions (m)
m	Mass (kg)
np	Total number of particles in simulation
nb	Number of boundary particles in simulation
P	Fluid pressure (Pa)
$\bar{P}$	Average fluid pressure within sampling area (Pa)
P_k	Fluid pressure at transducer *k* (Pa)
P_{ij}^*	Pressure in star region of Riemann Solver
Q	Overtopping rate (m^3/s/m)
Q^*	Dimensionless overtopping rate (m^3/s/m)
R_c	Free-board clearance (m)
R^*	Dimensionless wave runup (m)
r_{ij}	Distance between two particles i and j (m)
$\boldsymbol{r_{ij}}$	Distance vector between two particles i and j (m)
s	Slope of beach (m/m)
S	Wave board stroke (m)
t	Time (s)
t_f	Stroke duration (s)
T	Wave period (s)
T_m	Wave period at toe of wall (s)
u	Velocity (m/s)
$\boldsymbol{u}$	Velocity vector (m/s)
U	Velocity of bore-front (m/s)
W	Smoothing kernel function
W_{ij}	Smoothing kernel function at particles i and j
x	Position along x-axis (m)

$\boldsymbol{x}$	Position vector (m)
x_b	Wave breaking position along x-axis (m)
x_o	x-coordinate of origin of rotated axis relative to real x-axis (m)
X_o	Wave board displacement (m)
x_2	x-coordinate of particle on rotated x_2-axis (m)
z	Position along z-axis (m)
z_o	z-coordinate of origin of rotated axis relative to real z-axis (m)
z_2	z-coordinate of particle on rotated z_2-axis (m)
α	Empirical coefficient (Cuomo *et al.* 2010) or viscosity value in SPHysics
ς_{smooth}	Coefficient of smoothing length
ς_{sound}	Coefficient of speed of sound
ς_{CFL}	Coefficient for CFL condition ($m^3/s/m$)
β	Slope limiter constant
γ	Specific weight of water (N/m^3) or exponent for equation of state
γ_b	Berm constant
γ_f	Friction constant
γ_β	Wave angle constant
γ_v	Vertical wall constant
ε	XSPH coefficient or wave height ratio
η	Water surface profile (m)
ν	Kinematic viscosity of laminar flow (m^2/s)
ν_{wall}	Wall viscosity (m^2/s)
ρ	Water density (kg/m^3)
ρ_0	Reference density of water (kg/m^3)
σ	Total stress tensor
$\boldsymbol{\tau}$	Sub-particle scale shear stress tensor (N/m^2)

Π_{ij}	Viscosity condition for particles i and j
ω	Vorticity
φ	Angle formed with horizontal by front of seawall (degrees)
Ω	Influence domain of smoothing kernel
ξ	Irribaren number (slope similarity parameter)

List of Abbreviations

2D	Two-dimensional
3D	Three-dimensional
BEM	Boundary element method
BIM	Boundary integral method
CAD	Computer aided design
CFD	Computational fluid dynamics
CFL	Courant Friedrichs Lewy
CIEM	Large wave flume, in Spain (Canal de Investigacion y Experimentacion Maritima)
COBRAS	Cornell Breaking Wave and Structures
COBRAS-UC	Cornell Breaking Wave and Structures-University of California
CPU	Central processing unit
CUDA	Compute unified device architecture
DBC	Dynamic boundary conditions
DNS	Direct numerical simulation
EoS	Equation of state
FEM	Finite element method
FVM	Finite volume method
GPU	Graphical processing unit

GWK	Large wave flume, in Germany (Grosser Wellenkanal)
ISPH	Incompressible smoothed particle hydrodynamics
iVOF	Improved volume of fluid
JONSWAP	Joint North Sea Wave Project
LES	Large eddy simulation
LIM	Maritime Engineering Laboratory, in Spain (Laboratorio de Ingenieria Maritima)
LNEC	Engineering Laboratory in Portugal
MLS	Moving least squares
MPS	Moving particle semi-implicit
NS	Navier-Stokes
NS-VOF	Navier-Stokes volume of fluid
PAU	Pressure aeration unit
PROVERBS	Probabilistic Design Tools for Vertical Breakwaters
RANS	Reynolds-averaged Navier-Stokes
RBC	Repulsive boundary conditions
SPH	Smoothed Particle Hydrodynamics
SPS	Sub-particle scale
SWL	Still water level
VOF	Volume of fluid
WSPH	Weakly-compressible smoothed particle hydrodynamics

1. Introduction

1.1 Significance of the Study

There are many important applications of using protection walls to withstand the immense amounts of forces exerted by waves at a shoreline. Whether it's a seawall protecting a coastal city or a building required to withstand the risk of being struck by tsunami waves or bores, it is crucial to understand wave impact mechanisms. Water waves can exert forces on structures that act through a longer period of time with smaller magnitudes or impact forces that last for a very short period of time but have a very high magnitude. A few major tsunami and hurricane incidents have caused major economic damage and significant loss of life: the 2004 Indian Ocean Tsunami, Hurricane Katrina in 2005, Hurricane Ike in 2008, the 2010 Chile Tsunami and most recently the 2011 Tohoku Japan Tsunami. Figures 1-1 to 1-4 are images taken in Thailand in 2004, in Chile in 2010 and in Japan in 2011, showing only a fraction of the devastation. Table 1 shows estimates of the life and economic losses (NOAA, 2012 and CNN, 2011) concurred by the aforementioned past coastal disasters. Most of these shoreline structures are failing due to insufficient bearing capacity and the fact that the impact forces are difficult to predict. In most widely used engineering guidelines, impact forces are often not included due to the fact that they are difficult to quantify.

Figure 1-1 Catamaran boat logged on top of a building during the 2011 Japan Tsunami

Figure 1-2 Hotel in Japan (Taro) with first three floors washed out by the 2011 Tohoku Tsunami (Nistor, 2011)

Figure 1-3 Flooding of streets of Kesennuma, Chile caused by the 2010 Chile Tsunami

Figure 1-2 Destruction of the lower level of a resort building on the Nang Thong Beach, Thailand in 2004 (Nistor, 2011)

Table 1-1 Past coastal disaster estimates of economic loss and loss of life

Coastal Disaster	Economic Losses (US dollars)	Loss of Life/ Missing (persons)
2004 Indian Ocean Tsunami	9.9b	283,000
2005 Hurricane Katrina	96b to 125b	1,836
2008 Hurricane Ike	37.5b	200
2010 Chile Tsunami	15b to 30b	525
2011 Tohoku Japan Tsunami	250b to 309b	Over 27,000

Although the effects of impact forces usually occur over a short period of time, evidence of structures failing due to unpredictable impacts indicates that design guidelines are required to address their quantification. In most engineering design guidelines, extreme wave impact forces are not specified due to limited knowledge and research regarding this type of loading. Design codes such as the American Society of Civil Engineering ASCE/SEI 7-10 (ASCE 2010) and the Federal Emergency Management Agency FEMA55 (FEMA 2011), do not include direct formulations to calculate the force due to a tsunami impact, and mainly focus on storm flooding. Other guidelines, such as the Japanese Structural Design Method of Buildings for Tsunami Resistance (SMBTR) (Okada *et al.* 2006) and FEMA P646 (FEMA 2006), include force calculations for impacts on structures due to tsunami flooding inland by characterizing the impacts according to their components. The impact forces include hydrodynamic forces due to the rapid motion of the hydraulic bore, impulsive forces as a result of the initial impact of the bore and hydrostatic forces. Although the effects of impact forces generated by waves usually occur over a short period of time, evidence of structural failure due to extreme loading indicates that improved design guidelines are required to address this rather pressing issue.

The type of wave is also a major factor that influences how strong the hydrodynamic impact is and where along the wall the highest value of force and pressure is generated. The shape of the breaker influences the impact pressures such that more localized pressures of high magnitude will occur when the breaking wave is almost parallel to the wall in the moment

of impact. If the wave has already broken, it will entrain air creating a lower impact pressure or a 'cushioning' effect. The same applies when a wave breaks at the wall and the crest overturns trapping air. The bathymetry of the ocean plays a role as to whether or not waves break and where. Studies have shown that waves that break on the wall will exert much higher pressures than those that have not yet broken.

1.2 Objective

The purpose of this research thesis is to contribute to the understanding of the wave-induced forces and loading on near shore, inland vertical structures. This was achieved through the numerical modelling of wave-structure interactions and by comparing the numerical results against collected physical experimental results of other researchers. The numerical model used herein is based on the Smoothed Particle Hydrodynamics (SPH) method and was used to simulate several experimental wave-structure interactions.

1.3 Novelty

Although the most significant advancements in numerical modelling that involve wave impacts on structures have mostly been made using grid-based methods, these methods have proven to have significant computational limits. The mechanics behind wave impact is has been studied in the past and is generally understood. However, because the wave-structure interactions exhibit highly random features, such as the formation of air pockets in the runup wave along the structure, some uncertainties inevitably occur in the design of a coastal structure. There have been few previous studies on the wave-structure interaction using SPH; however, the number of existing studies that include cases of wide structures, such as seawalls, is considered sparse. Hence, there is an acute need of focused, carefully controlled studies focusing on extreme waves-structures interaction. The novelty of this study resides in being one of the first attempts to investigate the use of the SPH method to numerically model the wave-wide structure interaction.

The generation, propagation and eventual breaking of water waves are very complex and difficult processes to model numerically. This is because the turbulence models required to simulate this phenomena are computationally intensive for grid-based methods, especially for large computational domains. Another focus of the present study involved various techniques to model waves, such as the use of wave-paddles and dam-breach cases.

1.4 Thesis Outline

This thesis is organized into eight chapters. The following is a general description briefly outlining the various sections. Chapter **Error! Reference source not found.** introduces the topic of interest, why the research is significant and which objectives were achieved. Chapter **Error! Reference source not found.** provides an overview of the background studies and research techniques that have been used to investigate the impacts of hydrodynamic forces on structures. Various publications are reviewed and then discussed. Chapter 3 reviews the theory behind the Smoothed Particle Hydrodynamics (SPH) method. It also discusses the parameters used in the numerical model and discusses the advantages and disadvantages of using this type of model for the investigated research topic. Chapter 4 describes each physical experiment which was modelled and compared to using the SPH model. The significant results of each comparison are discussed herein. Chapter 5 outlines the numerical modelling domains used for each physical experiment, the corresponding selection of parameters and provides a preliminarily discussion of the results. Chapter 6 provides a detailed analysis and discussion of the physical and numerical experimental results. Chapter 7 outlines a summary of the research findings and provides conclusions and recommendations for further studies. Finally, Chapter 8 contains the recommendations for future work that could potentially be completed to further investigate this topic.

2. Literature Review

2.1 Introduction

In this section, previously published papers will be reviewed along with the presentation of their assumptions, details, and results. For most hydrodynamic studies it is crucial to obtain laboratory experiments that provide the data required to engage in further testing and perform a comparison to numerical results. This is primarily due to the fact that field data is nearly impossible to obtain for cases involving significant wave impact forces on structures. Most papers referred to in this chapter involve wave impacts on vertical structures. Nonetheless, other types of shore protecting structures were also considered. Subsequently, both physical and numerical experiments are analysed and discussed.

2.2 Analytical Modelling of Wave Bore – Structure Interaction

Blackmore and Hewson (1984) explained that at the time, the existing coastal structures were designed according to empirical equations due to a lack of physical experimentation at large scale and because the occurrence of an extreme wave is considered low. This leads to both over-design and under-design which is undesirable in any engineering application. Prior to 1984, most instrumentation had a slow response time and would not be able to record forces with short impact times. The authors compared results from field data and mathematical model experiments. It was shown that forces predicted using model scales are overestimated.

Okamura (1993) investigated the effects of impulsive wave pressures on an inclined wall. Okamura uses the model proposed by Peregrine and Cooker (1990) in order to estimate the impulse pressures on the wall, which assumes an incompressible, inviscid flow with a constant vorticity. It was found that the effects of the high accelerations and pressures acting on the wall were more significant than those caused by gravity, hence the equations for a vertical wall used for the inclines as well. After all the mathematical model derivations

were completed, the author was still not able to identify which of the pressure or the pressure impulse is the most significant when designing for the strength of the wall.

Kirkgoz (1995) classified a plunging breaking wave into three classes: early breaking, late breaking and perfect breaking. The first category wave breaks on the wall and entraps air. The second one breaks on the wall but not in a vertical position with respect to the front face of the wall whereas the last wave class broke vertically on the wall. The author reviewed the effects of a breaking wave on both sloping and vertical structures according to the past literature. Many semi-empirical formulae were developed from the impulse-momentum principle to find the force of impact due to breaking waves however they all seemed to have underestimated the pressure. The experiments showed that the water depth at the wall is very influential on the resulting impact force. There is a range of values above and below the SWL that causes the maximum impact pressure to exceed the critical limits.

Kortenhaus *et al.* (1996) proposed a new method to design a protecting wall's geometry and have used the Hamburg harbour marina walls as an example for the study. The potential loading cases on the Hamburg marina can be classified into three cases: standing waves which are rare but occur in higher depths, plunging breakers which rarely occur but are the most dangerous and broken waves which are the most frequent type. The recommended design flowchart suggests that, first the input parameters of the case in question must be specified. These include the width of the berm, water depth at berm, water depth at wall and free wall height. Next, the breaking criteria and wave transformation needs to be defined. Afterwards, the various loading cases should be found. Finally the changes in the water levels in the area of the coastal protection should be known. Kortenhaus *et al.* (1996) have set up a design procedure such that it is usable for any case of coastal structure with a similar layout.

Peregrine (2003) discussed the impacts of waves on walls in his paper. The author gave a general idea as to what type of pressure profiles are to be expected in the case of the various wave impacts. According to Peregrine, waves are said to vary from smaller regular

waves to intense sea waves that travel in many directions. Some important factors affecting the type of breaking wave are the roughness of the wall and the bed topography. This will in turn affect the shape of the impacting wave hence peak loads. The water depth in front of the wall is another extremely important factor. The peak pressures are mostly due to the inertial or dynamic forces of the wave. The secondary peak load, however, is due to the hydrostatic impact of the wave. The typical wave profile is then observed to form a "church-roof" profile where the peak impact forms and the remaining loads will gradually decrease to zero. In 1939, Bagnold concluded that the most impacting loads are due to the waves with the least amount of air (Bagnold 1939). However, it is considered very difficult to assess the effects of entrapped air because of the scaling effects that are observed in model experiments. In terms of a real world case of a wall being impacted by an ocean wave, the effects of a peak pressure are actually localised and may not create as much damage on the structure. On the other hand, the effect of some other factors can be increased due to the three dimensionality of a real case.

Chanson (2006) considered an ideal dam break that produces a surging wave that travels over a horizontal bed. The front of the wave is originally considered to be a parabola however this is not the real case where the real fluid creates a tip shape that is concave down. This is illustrated in Figure 2-5 below. By applying the suggested conditions for an ideal dam break in a wide rectangular channel (Ritter1892) and simplifying the continuity and dynamic wave equations, the celerity of the wave front, U, for an initial water depth d_o, is obtained as:

$$U = 2 * \sqrt{g * d_o} \qquad \textbf{(2-1)}$$

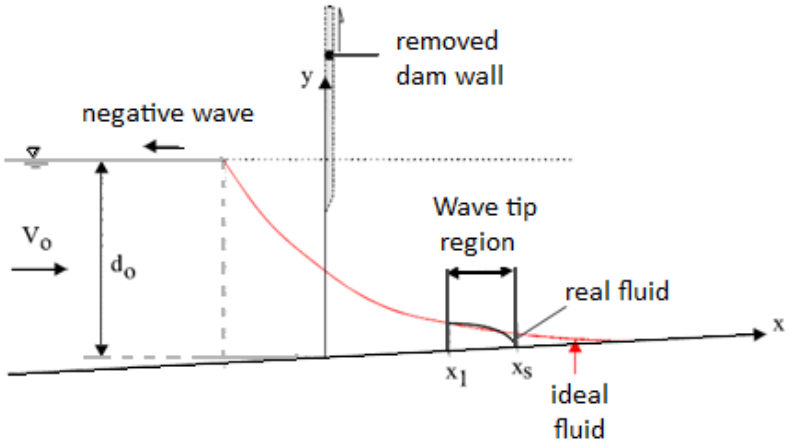

Figure 2-1 Ideal versus real dambreak surface profile (Chanson, 2006)

The region in which the flow stops being ideal starts at x_1. A relationship between the ideal case and the real case was formed by assuming that the mass of the tip should be equal in both cases. The resulting model involving the surface profile was tested according to previous models by Dressler (1954) and Cavaille (1965), showing success. The developed model can be applied to a semi-infinite reservoir to model an advancing tsunami bore. Real-life tsunami data and applications were compared to the model. The case of the 2004 Indian Ocean tsunami was used and showed agreement particularly in the region of Banda Aceh. Chanson claimed that this model is useful for sudden, unexpected emergency situations in order to come up with quick approximations.

2.3 Physical Modelling of Wave Bore – Structure Interaction

Partenscky's (1988) research paper focused on the effects of a breaking wave on the potential sliding, overtopping or foundation failure of a vertical breakwater. The maximum force impact on a wall was shown to be caused by the case where the front of the wave at impact is vertical. Partenscky discussed the guidelines that were available at the time showing that they underestimate wave loading on coastal structures and thus need to be improved. He suggested that since the duration of the force impact is too short to cause any significant damage on the structure, the average of the pressures in the resulting distribution should be taken into account when, for example, designing for a caisson. This should be done with the condition that the mean dynamic pressure duration is considered

over the total impact duration on every point of the caisson. Figure 2-1 below displays how this distribution should be set up.

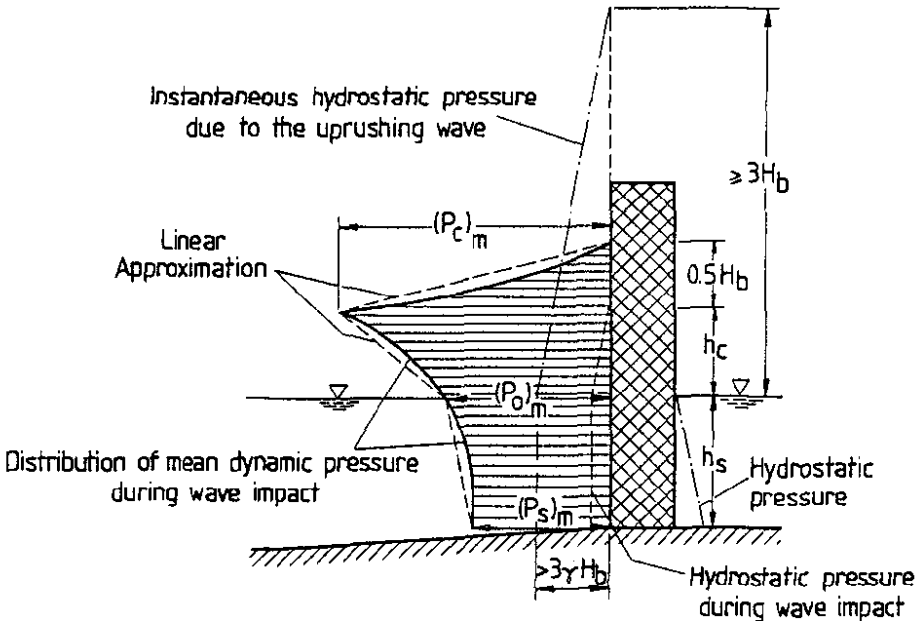

Figure 2-2 Proposed pressure distribution for structures exposed to breaking waves by Partenscky (1988)

Kirkgoz (1990) suggested that the impacts of a broken wave on a vertical wall are much greater than those by an unbroken wave. This is demonstrated through a laboratory experiment where a paddle type wavemaker was used to generate regular waves that travel up a beach slope of 1:10 and impact a 1.0m vertical wall. The water depth was kept at 61cm and the wave heights at 26cm. The plunging waves were set to break on the wall and the maximum forces exerted were located just below the still water level (SWL). The maximum forces were found to be within the range of 7 to 20kN/m. The author concluded that the high magnitude forces would only act in very short instances and only locally affected the wall deflection.

Ramsden and Raichlen (1990) focused on generating solitary waves that would impact a vertical wall in order to assess the bore-structure interaction. The experiments were conducted in a flume of length 40m, width 1.1m and height 0.61m. A paddle-type wavemaker was used to create solitary waves that propagated up a mild beach of slope 1:50. The wall extended the width of the tank and was comprised of a central instrumented

section of width 4.95cm. Two pressure transducers were mounted on the wall at 3.83cm from the bottom and 30.48cm above that. The results showed that the peak forces occurred after the maximum runup.

Schmidt *et al.* (1992) explained that the assumption of not designing for impact loads is not necessarily correct especially in cases where the impact loads may affect the entire structure's stability. The purpose of their paper was to study the effects of impact loads on a vertical structure. Experiments were conducted in the large wave flume (GWK) in Hannover, Germany. A piston-type wavemaker was used to generate plunging-type breaking waves that traveled up a slope of 1:20 at water depths of 3m. Solitary waves were found to produce the best results in terms of creating the typical breaking impact wave. Results showed that as the wave impacted the wall, an air pocket was formed and negative pressures were observed. This was explained by the fact that the air was highly compressed and sent the surrounding water particles in the opposite direction. After examining the peak forces, it was found that the worst case of impact on a vertical structure is that of a plunging breaking wave that breaks on the wall.

Chan (1994) physically modelled the mechanics of a plunging wave impact on a vertical structure. Chan suggests that the failure of the wall could be divided into two main categories: the formation and impact of weak spots on the vertical structure and the overtopping or moving of the entire structure. Experiments were conducted in a wave flume of length 30m, width 0.76m and height 0.9m. The plunging waves were created using a wave-focusing technique meaning that multiple sinusoidal waves are produced with ranging frequencies such that the amplitudes add up at a specific focal point creating a broken wave. In the results, the slight variation in the pressure between separate experimental runs was attributed to the randomness of breaking waves and the dynamics of the entrapped air. The author concluded that the entrapped air poses too many scaling complications, making it difficult to convert the model results to a prototype scale.

Hattori *et al.* (1994) focused on the effects of air pockets created in flip-through or plunging type wave-wall collisions by physically modelling wave impacts. The types of impacts were

related to where the wave breaks in front of the vertical structure. The experiment is conducted in a wave flume of length 20m, width 0.3m and height 0.55m. A flap-type wave generator was used to create regular waves which travel along a beach of slope 1:20, eventually impacting a vertical wall. The wall rested on a mound in order to represent the rubble mound of a composite type breakwater. A total of six pressure transducers were located up the centreline of the wall and 5 cm from the centreline. The water depth at the wall was about 5cm and the incident wave heights ranged from 4cm to 12cm measured at 7m from the wavemaker. Authors classified the impact types into four groups: flip-through, plunging, well-developed plunging and turbulent bores. Negative pressures were observed when the air bubbles expanded. Results supported the fact that the peak pressures occur when the breaking wave impacts the wall with air pockets.

Franco de Gerloni and van der Meer (1994) were investigating the effects of overtopping a vertical breakwater. At the time, Goda's formulas from 1985 were still the most commonly used equations when designing a breakwater. Experiments were performed in a wave flume of length 43m and of depth 1.5m. Random waves were produced under non-breaking conditions with peak periods according to the Joint North Sea Wave Project (JONSWAP) spectra (Hasselmann *et al.* 1973). Results confirmed that the overtopping volume is more important than the mean discharge when designing for the safety of the breakwater. It was found that the overtopping discharges were allowed to be higher than the maximum found by using the manuals which endangered the people walking on or near the structures.

Azarmsa *et al.* (1996b) conducted laboratory experiments in order to investigate the characteristics of a spilling and plunging-type wave impact on a wall. The wave flume used was 54m long, 1m wide and 1m high. A piston-type wavemaker was used to generate the waves in a SWL of 0.15m which propagated down the flume, over a 0.09m high reef, and finally impacting a wall. The wave height ratio for the spilling breaker was 0.24 and 0.55 for the plunging breaker. The physical experiments were compared to a numerical model which was based on a nonlinear boundary integral method (BIM). The computational parameters and other details are described in Azarmsa *et al.* (1996a). The location of the wall was

changed in order to vary the type of breaker. It was found that by varying the position of the wall by as little as 2cm will greatly influence the pressure time history on the wall. The results showed that the air pockets will decrease the values of impact pressures by offering a cushioning effect. The author's conclusion is that the occurrence of a peak impact pressure is not dependent on the type of wave, but rather on whether or not the structure is frequently subjected to a peak wave.

Ramsden (1996) followed up on his paper in 1990 (Ramsden and Raichlen 1990) by continuing to study the effects of a bore impact on a vertical wall. Two additional setups had been introduced where the tank is horizontal and the wall remains at the end. The first case consisted of a piston type wavemaker that produces a solitary wave that breaks and impacts the wall as a bore. The second case involved a dam breach surge that will also propagate down the wave tank and impact the wall. The normalised force on the wall was taken to be as follows:

$$F_l = \frac{1}{2}\gamma b(2H + h_w)^2 \qquad \textbf{(2-2)}$$

where, γ is the water weight per unit volume, $h_w$is the water depth near the wall and b is the width of the wall. F_l is considered to be the force on the wall due to runup reaching two times the wave height in hydrostatic conditions. The experimental results were compared to results from Cross (1967) and were found to be in good agreement until a specific time after which the force seems to be over-predicted. It is important to note that the spike in pressure produced no response in the force and this is attributed to the fact that the data collection frequency was too large. Ramsden then compared the wave results from a turbulent bore and a dry-bed surge. Depending on where the wall is placed, this could drastically change the results. The author explained that as the bore strength increases, the measured force becomes much larger than the linear force F_l and this is because the wall must resist the higher celerities. Ramsden's work led to a number of conclusions. The forces produced during impact were smaller because of the reflected waves that produced vertical accelerations and hence reducing the effect of the impact on the wall. The maximum forces under hydrostatic conditions were found to be larger than those measured. The results

should only be used for the estimation of structural failures under conditions like sliding and overturning.

Bergmann and Oumeraci (1998) investigated the difference between the forces produced by impacts on a vertical wall and a porous vertical wall. They point out that the design guidelines proposed by Goda (1985) do not consider porous walls. The experiment takes place in a wave tank in the GWK in Germany which is 320m long, 5m wide and 7m deep. The SWL was set at 4m and the permeable wall is placed at approximately 96m down the flume. The effects of different types of waves were tested. These include regular waves, solitary waves and random waves with wave heights ranging from 0.5m to 1.5m. The porosity of the wall was varied from impermeable to 40.5 percent permeable. The porosity was defined according to the ratio of the spacing s to the distance between the wall element centrelines. Figure 2-2 below illustrates the wall layout.

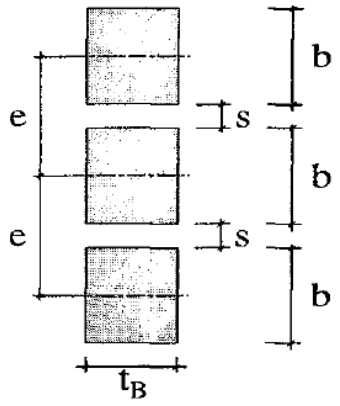

Figure 2-3 Wall porosity defined as P= s/e % (Bergmann and Oumeraci, 1998)

The pressure distributions obtained were compared to Goda's formula (1985) and it was shown that the pressure values by Goda (1985) were higher than those obtained in the experiment. In the case of the impermeable wall, Goda's formulation was found to under-predict the pressure values. In order to account for the wall porosity, the authors suggested the use of a linear factor in Goda's formula; however, this was shown to be unsuccessful. Hence, a non-linear factor was developed.

Bullock *et al.* (2001) investigated wave impacts on vertical walls by comparing data obtained from field measurements, laboratory tests and numerical results. The field test data was obtained from the Admiralty breakwater on the island of Alderney off the northern coast of France. Figure 2-3 below shows the cross section of the Admiralty breakwater as well as the location of the pressure transducers. The measured data showed that there was a variation in the pressure values along the horizontal, suggesting the need for a three-dimensional model as well. Typical pressure distributions can be seen Figure 2-5.

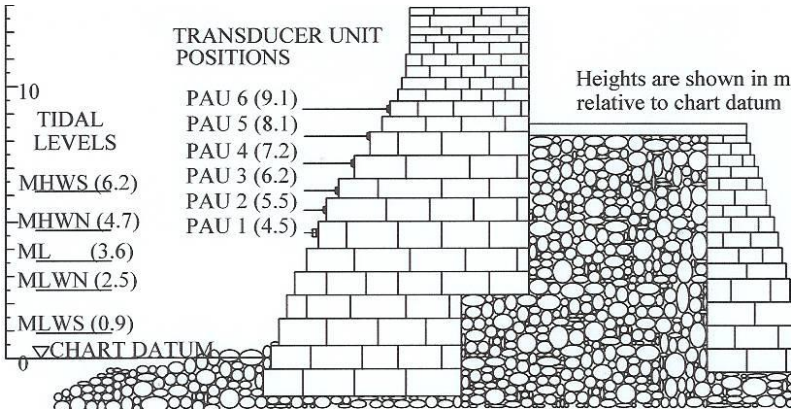

Figure 2-4 Section of Admiralty breakwater and pressure measurement locations (Bullock *et al.,* 2001)

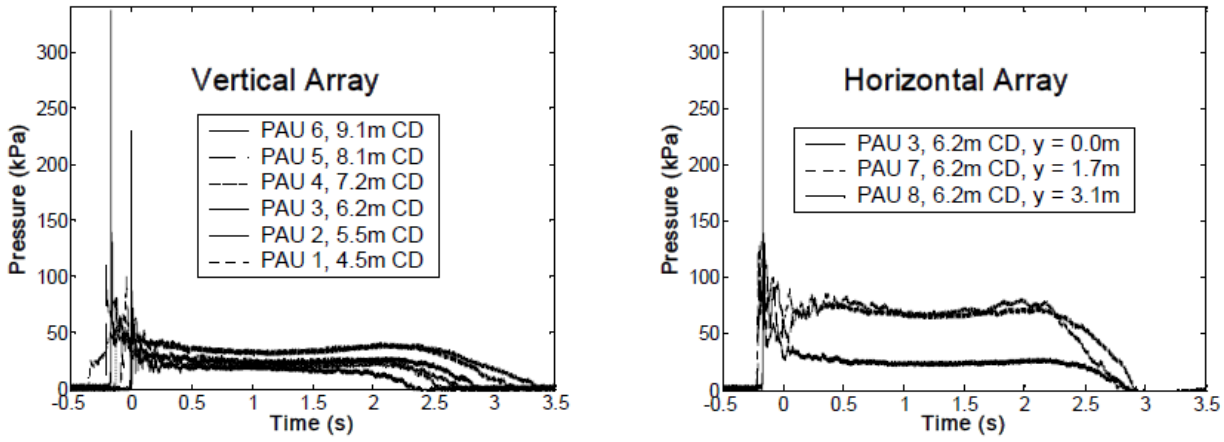

Figure 2-5 Impact pressure time histories from field data (Bullock *et al.,* 2001)

Two sets of laboratory tests were performed in the GWK at Hannover, in Germany and in the wave flume at the University of Plymouth in the United Kingdom. The first was a large scale setup where the breakwater was modelled at a 1:4 scale and the second is a small

scale setup at 1:25. In the large scale tests, the water depth was kept in the range of 0.75m to 1.7m and a wavemaker was used to create regular waves with heights ranging from 0.5m to 1.7m. The small scale tests were conducted in a flume of length 20.7m, width 0.9m and depth 0.9m. Both freshwater and saltwater were tested and it was observed that the breaking waves in seawater tend to allow more aeration and bubbles. The numerical experiments had not yet been developed at the time of publication. The authors concluded that additional work is required and that there should be some improvements in the design procedure suggested by Allsop and Kortenhaus (2001).

Bullock *et al.* (2007) also performed experiments testing the influence of entrapped air in the wave at impact. The experiments took place in the GWK where a wall was constructed out of a 90mm thick reinforced concrete plate with stiffening beams. Four pressure aeration units (PAU) and nine pressure transducers were used to take readings on the wall. At one of the levels on the wall, both a PAU and a transducer were placed one meter apart in order to obtain a comparison in the readings. Distinctions were made between four types of waves to be tested by the following definitions. A wave was defined as 'slightly breaking' if it is 1 to 2.5 times the maximum quasi-hydrostatic load and was referred to as 'broken' if the wave breaks before reaching the wall and the turbulence in the wave entrains a lot of air. A wave was labelled as 'low aeration' if it had a voids ratio of less than or equal to 5% and a high and short duration load. A wave was considered 'high aeration' if the void ratio is high and the increase in pressure takes a relatively long time.

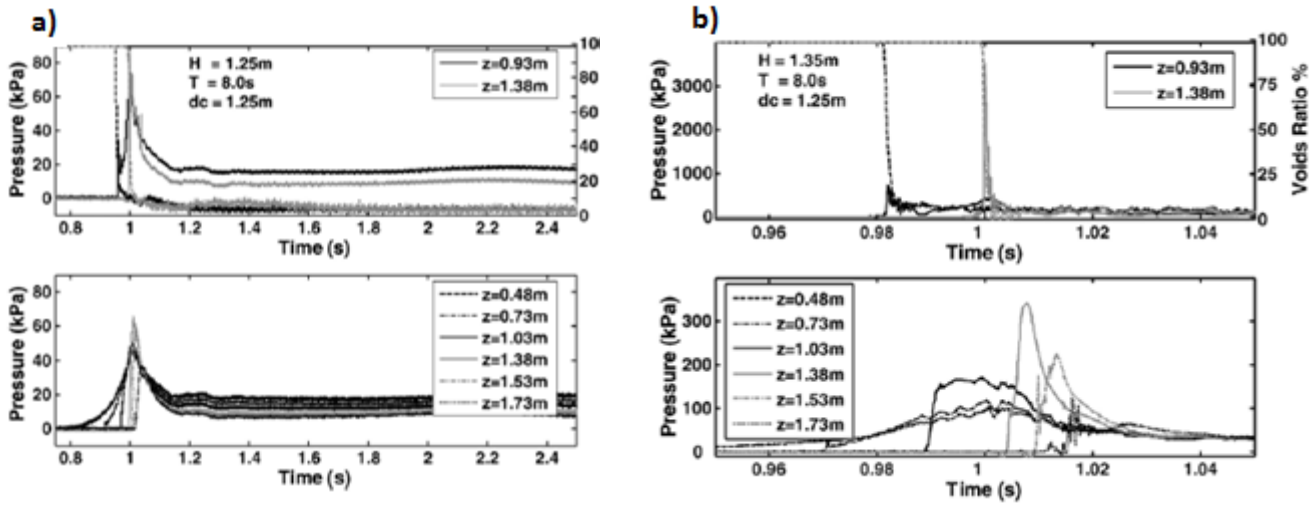

Figure 2-6 Pressures measured for a) slightly breaking and b) low aeration waves (Bullock *et al.,* 2007)

In Figures 2-6 (a) and (b), the top panels show the pressures measured by the PAUs and the bottom panels show the pressure measured by the transducers for a slightly breaking wave and low aeration waves. The value of z describes where the measuring instrument was placed above the mound crest. Figure 2-6 (a) shows a spike in the pressure for a slightly breaking wave at around the same time of 1.0s. The void ratio rose to about 100% at the impact and then dropped back down to a steady low rate. The drop and consistent level suggests that no additional air was entrapped in the wave. The maximum pressure was measured to be 65kPa by the PAUs and 75kPa by the transducers. For the case of a low aeration wave, the spikes in the initial impact pressure were detected at different times by the PAUs but did have very short rise times. The PAU's located higher up the wall recorded maximum pressures of 3500kPa with rise times of 1.2ms. The comparison PAU recorded a peak pressure of 720kPa through 2.5ms while the transducer recorded a maximum pressure of 341kPa with a rise time of 3.8ms. This is an indication that even at 1m apart there can be a different reading, demonstrating the sensitivity behind the placement of the instruments. In the case of a high-aeration wave, the top panel shows that the PAU at the higher position recorded a pressure before the lower one. This indicates that an air pocket has been trapped in the wave and caused this type of profile. The pressures recorded are very low at approximately sub-atmospheric levels. If this case occurs in the field, for instance on a stone wall, the wave will exert a negative seaward force on the wall, possibly

removing bricks from the wall if the pressure builds in the spaces. From the differences in the recorded pressures for low and high aeration, it can be concluded that the entrained air caused lower force values but higher impulses on structures. For a slightly breaking and low aeration wave, the peaks in pressure tend to occur just above the SWL. In the case of a high aeration and broken wave, the peaks occur along the SWL. According to Hull *et al.* (2002), the location of the peak pressure is usually above SWL for flip through impacts, down to the SWL for breakers that have trapped air, and below the SWL for broken waves. The results were considered to follow this proposed pattern.

In Cuomo *et al.* (2010), a comparison was made between short duration impact loading and quasi-static loading on a wall. The experiments took place in the large wave flume (LIM) in Barcelona, Spain. This flume is 3m wide, 4m deep and 100m long. There is a 1:13 concrete beach approaching the wall where pressures were recorded by eight transducers, logging at frequency of 2000Hz, placed on the front face of the wall at a spacing of 0.2m. Horizontal forces on the front of the wall were calculated as follows:

$$F_h = \sum_{k=1}^{8} P_k \, \Delta z \qquad \textbf{(2-3)}$$

where, P_k is the pressure recorded at each k transducer and Δz is the distance between the transducers (0.2m). A wave loading was considered 'impact' loading if it has a very short duration in the range of the natural period of vibration of the structure. On the other hand, a 'quasi-static' loading had a duration that is greater than two times the natural period of vibration. Pressures measured in this experiment were compared with results that were predicted using models by Goda (1974), Sainflou (1928), Hiroi (1920) and many others, for both impact and quasi-static loading. In Figure 2-6, the data points that lie on the solid line are considered the best predicted. Predictions for the quasi-static loading were the best using Goda's equations. The Sainflou method under-predicted the measured values and Hiroi gave a fairly conservative result. For the methods that are being compared to, there is a wide range of scatter of impact load predictions. It was observed that the methods

established can be used for smaller values of impact loads; however, they may produce very unreliable results for larger and more extreme impacts.

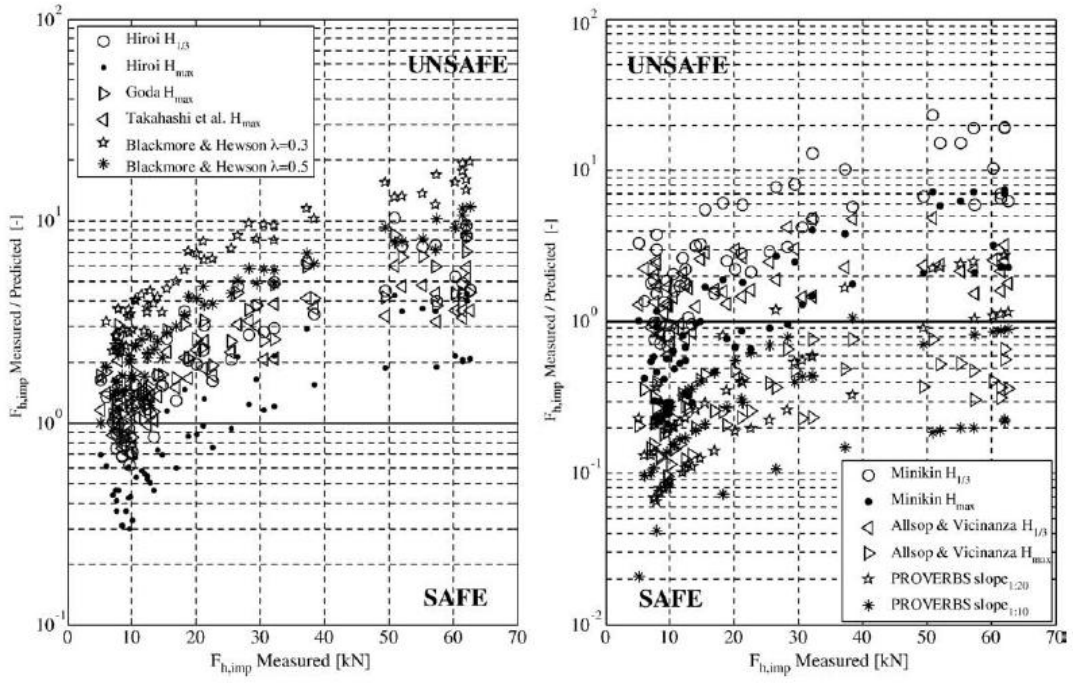

Figure 2-7 Comparison of wave impact forces between various proposed models (Cuomo *et al.,* 2010)

Goda and Takahashi's predictions greatly underestimated the values and this was because those equations were developed to estimate longer duration impacts. Other methods such as those by Minikin (1963) and PROVERBS, Probabilistic Design Tools for Vertical Breakwaters, (McConnell and Kortenhaus 1997) showed a general pattern that averages around the solid line meaning they were able to sufficiently make a prediction. Cuomo *et al.* proposed new equations to calculate the forces due to impacts. The equation for impact loading is as follows:

$$F_{h,imp} = \rho g H_{m0} L(h_s) \left(1 - \frac{|h_b - d|}{d}\right) \quad \textbf{(2-4)}$$

where, $L(h_s)$ is the wavelength at the toe of the structure, H_{m0} is the significant wave height, h_b is the breaking depth. The proposed equation for quasi-static loading is:

$$F_{h,qs} = \alpha \rho g {H_{m0}}^2 \quad \textbf{(2-5)}$$

where, α is an empirical coefficient taken as 4.76. It was found that the equation seemed to predict more accurately the quasi-static loads than the impact loads. Cuomo *et al.* (2010) concluded that although the new formulations seem simple, they were able to satisfactorily make predictions of the forces. However, because they are empirical and derived from this specific case, they must only be used to make predictions for vertical structures on steep approach slopes.

Hofland *et al.* (2010) performed large scale tests with a high temporal and spatial resolution. The experiments were conducted in a flume of length 200m, width 5m and depth 7m. Broken waves were generated using a method proposed by Van den Boomgaard (2003), where multiple waves were focused at a predetermined point in order to add up their energies and cause the wave to break. A 9.0m long wall included an instrumented section which contains 23 pressure sensors with a sampling rate of up to 50 kHz. Results showed that the highest pressures were observed when a flip-through impact occurs. The peak impacts occurred above the SWL, which was not in agreement with Goda (2000). The model was not considered completely representative of the real world case since freshwater rather than salt water was used. This plays a role on the compressibility of the air, which will change the values of the pressures obtained for the air entrained waves.

2.4 Numerical Modeling of Wave Bore – Structure Interaction

Xiao and Huang (2008) modelled a broken solitary wave's runup on a model beachfront house. They achieved this by using a Reynolds Averaged Navier-Stokes (RANS) equation alongside a k-ε equation proposed by Lin and Liu (1998). The k-ε model is used to represent the turbulence in the wave breaking mechanism. In order to track the free-flowing surface, the volume of Fluid (VOF) function was used. They compared the results to an experimental model generated by Synolakis (1986) in order to validate their model. Solitary waves of wave height to water depth ratio of 0.28 were generated and broke onto a beach slope of 1:20. According to the graphical comparison, the values of the surface profile by the

proposed numerical model are extremely close to the experiments by Synolakis (1986). The model was then used to compare analytical solutions of impacts of forces on a vertical wall using results from Fenton and Rienecker (1982). The normalised force versus height graph indicates a very close agreement. Finally, the model is compared to experiments involving a 1:20 beach and idealised houses placed at various elevations along the slope. The maximum forces at impact due to the broken solitary wave were observed at the lowest elevation where the house was partially submerged. This was expected as the wave energy has not dissipated as much as if it were to travel inland as a bore. Xiao and Huang observed that as the house was moved up the shore by a distance of a quarter of the runup, the impact forces would almost half every time.

Tanizawa and Yue (1992) focused on the effects of air pockets when a plunging wave impacts a vertical wall. The authors modeled the impacts using a mathematical interpretation of an irrotational flow of a homogenous, incompressible and inviscid fluid flowing in two-dimension. The air pockets were simulated by assuming uniform, adiabatic air compression. The governing equation used to achieve this was a polytropic gas law. The waves are generated using a piston wavemaker. The numerical methodology was based on a two-stage process of fitting and smoothing then re-gridding the domain using a grid function. Their numerical model was compared to experimental results from Chan and Melville (1988). It was shown that the dimensions of the simulated air pocket and maximum impact pressure were very close to the experimental values. The duration of impact, however, was found to be at least double the experimental value. Also, they discussed that because the values of the pressures for air impacts were smaller and over a larger area than the water only impacts, the air pressure impulse was found to be greater and was considered more important in the design of coastal structures.

Peregrine and Topliss (1994) investigated the pressure field exerted by a wave impact on a vertical wall numerically by modelling the overturning wave using the model proposed by Dold and Peregrine (1986) and Dold (1992). The model assumes that the fluid is incompressible and irrotational. The time stepping of the model was done using a truncated

Taylor series. The numerical results are compared to physical experiments by Hattori and Arami (1992). It was found that the larger the entrapped air pockets, the lower the resulting pressures obtained on the wall. The equation provided originally by Peregrine (1994) to calculate the maximum pressures was compared to the new proposed equation and found to be within 30 percent agreement, hence suggested as a means to obtain a rough approximation.

Liu and Al-Banaa (2004) executed a numerical simulation of a solitary wave impacting a marine barrier using a model called COBRAS (Cornell Breaking Wave and Structures) (Lin and Liu, 1998) that can solve RANS equations. The wave turbulence was incorporated using the k-ε turbulence model. The turbulent kinetic energy is represented by k and the rate of dissipation of k is given by ε. The results were used to develop formulae to estimate maximum runup and maximum force on the vertical barrier. They showed that the numerical force-time history was in excellent agreement with the experimental one. The authors were able to derive two useful formulae in order to approximate the maximum values of wave runup due to a solitary wave collision and maximum values for the force on the marine barrier.

Kirkgoz and Mamak (2004) used a theoretical approach that involves a boundary element method to numerically solve the pressures induced by the impact of a wave on a vertical wall. The model was developed to work around the setbacks of using the impulse-momentum principle which has failed due to practicality (Cooker and Peregrine, 1995). The results showed reasonable agreement when compared to the experimental ones, however some wave cases showed under- or over-prediction in the values of impact pressure. The model works based on the condition that the rise time of the wave is known, which is considered a major limitation.

Wemmenhove *et al.* (2006) aimed to numerically simulate a loading case from Bullock *et al.* (2001) in a two-phase flow and test the theory proposed. The results from Bullock *et al.* (2001) showed that the peak pressures due to trapped air in an impacting wave may decrease but will usually have a longer duration. In the past, wave impact problems had

only been modelled using one-phased flow which neglects the effect of air entrainment. The authors recommend using a two-phase flow when the impacts are more violent. Since air bubbles and air pockets have a diameter of relatively small order of magnitude (1mm – 100mm), this introduces computational problems to the scale and temporal domain. The authors applied an improved VOF (iVOF) model to track the surface and model the flow using a 3D VOF solver called ComFLOW (Garrits and Veldman 2003). The case of a dambreak wave impacting a wall was considered and the results showed that the air entrainment had no significant effect on the values of the pressures.

Mokrani *et al.* (2010) numerically modelled the effects of a plunging breaker on a vertical wall using a two-dimensional NS model for a two-phase flow. The surface of the flow and the interface were tracked using a VOF equation. The turbulence in the plunging breaker was modelled using a mixed scale sub-grid model. The result is a system of linear equations that are solved using a multi-frontal massively parallel sparse direct solver (Amestoy *et al.,* 2000). The solver calculates the average water volume in each cell and uses it to find the new densities and viscosities involved in the next time step. When compared to the BEM model proposed by Vinje *et al.* in 1981, the NS-VOF model proposed showed a delay in predicting the wave-breaking surface profile. The reason for this difference was attributed to numerical diffusion. The maximum obtained forces, when compared to experiments by Schmidt *et al.* (1992), were showed to be under-predicted by a factor of four.

Hsiao and Lin (2010) investigated the effects of a solitary wave impacting and overtopping an impermeable seawall by using the numerical solver called COBRAS (Lin and Liu, 1998). The authors conducted both numerical and physical experiments in order to validate the capabilities of the COBRAS model. The physical experiments were conducted in a flume of length 22m, width of 0.5m and a height of 0.75m in the Tainan Hydraulic Laboratory. Solitary waves were generated using a piston type wavemaker by applying Goring's mathematical model from 1978. As previously explained, COBRAS employs a two-dimensional RANS model and the turbulence is modelled using a k-ε model. The numerical domain was shortened with the appropriate scaling in order to reduce the computational

time. The numerical model was not able to adequately model the impinging bore front with an error of up to 20 percent. Results show that the entrapped air pocket behind the wall was simulated in agreement with the physical tests.

2.5 Structure Interactions using Smoothed Particle Hydrodynamics

Randles and Libersky (1996) discussed the improvements and changes in SPH for both fluids and solids. Focusing mainly on the section on fluids, the weaknesses of SPH, such as instability and poor accuracy, have been overcome through the use of a kernel renormalization and by using a conservative smoothing method. The SPH method was found to be favourable because it is relatively simple to apply, making the fluid-structure interaction model more robust and incorporates the void treatment for multiphase flow.

Lo and Shao (2002) used an SPH model with large eddy simulation (LES) in order to generate solitary waves. LES is a model that involves aspects from both direct numerical simulation (DNS) and RANS in order to model the turbulence of breaking waves. Through the use of SPH, the authors were able to avoid the Navier-Stokes (NS) equations in the Eulerian form which is found to introduce computational problems for wave-breaking cases. SPH and moving particle semi-implicit (MPS) methods are the most popular non-grid-based simulation techniques in use. The paper provided results of using a two-dimensional incompressible SPH model. The authors claimed that the incompressibility allows for easier tracking of the free surface. There must also be a large number of particles in order to minimize discrepancies resulting from particles flowing as the fluid would. In order to achieve the suitable amount of particles, the number of particles should be doubled until a point where no more changes in the results occur and the behaviour converges. The Boussinesq (1972) equation was used to model the solitary wave action where the surface profile is modelled as:

$$\eta(x,t) = a\,\mathrm{sech}^2\left[\sqrt{\frac{3a}{4d^3}}(x-ct)\right] \quad \textbf{(2-6)}$$

where, a is the amplitude, d is the water depth and c is the wave celerity. The model was verified using a water depth of 0.1m, ratio a/d of 0.15 and particle spacing of 0.01m. This generated approximately 12,000 particles in the numerical domain. When compared to Synolakis' experimental waves from 1987, there was good agreement in the wave generation, shoaling and breaking processes. However, the turbulence results were in partial agreement and further research is required.

Gomez-Gesteira and Dalrymple (2004) used a three-dimensional SPH method to model the impact of a dam breach case on a tall column for both the cases of a circular and square column. The physical experiments compared are those done by Yeh and Petroff from the University of Washington. The setup can be seen below in Figure 2-7.

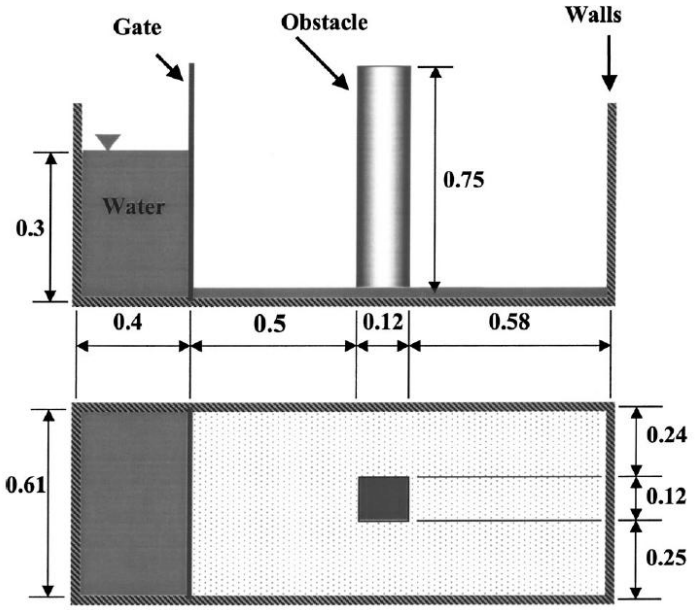

Figure 2-8 "Bore in a Box" experimental setup of Yeh and Petroff (units in meters) (Gomez-Gesteira and Dalrymple 2004)

The equation of state used to model the relationship between the pressure and the density was that proposed by Batchelor in 1974. The kernel function was chosen to be a normalized

cubic spline function and the XSPH variant (Monaghan 1989) was used to govern the movement of the particles. The XSPH variant allows the particles to move with the same average velocity as the fluid particles surrounding it. Results showed a good agreement between the forces on the structure and the velocities of the particles for the experimental data and the numerical SPH model.

Dalrymple and Rogers (2005) described the main issues of using SPH to numerically model the wave-breaking process. Other mentioned methods are the boundary element method by Grilli *et al.* (2000), the direct numerical simulation (DNS) by Lin and Li (1998) and the RANS model with LES by Wu (2004). In the SPH model, the true speed of sound is not used, because it would cause the time steps to be much smaller, hence greatly increasing computational time. The speed of sound is modelled as ten times the peak celerity of the wave. It was found that using the Verlet (1967) algorithm would halve the computational time required to complete the simulation.

Silvester and Cleary (2006) also modelled the laboratory experiments conducted by Yeh and Petrov where a dam break case takes place in a rectangular tank whose dimensions are 0.61m wide, 1.6m long and 0.75m high. The structure that was impacted was a square column of 0.12m side dimension. The column was placed at 0.9m from the end of the tank. Refer to Figure 2-7 for a schematic diagram. The water was stored in a reservoir behind a gate which is then opened to release a wave. In the SPH experimental model, when the fluid is released from its initial state, the water rushes down the path and creates a breaking wave that tends to propagate like a bore. As shown in Figure 2-8, the predicted forces obtained using the SPH model are in good agreement with those measured by Yeh and Petrov.

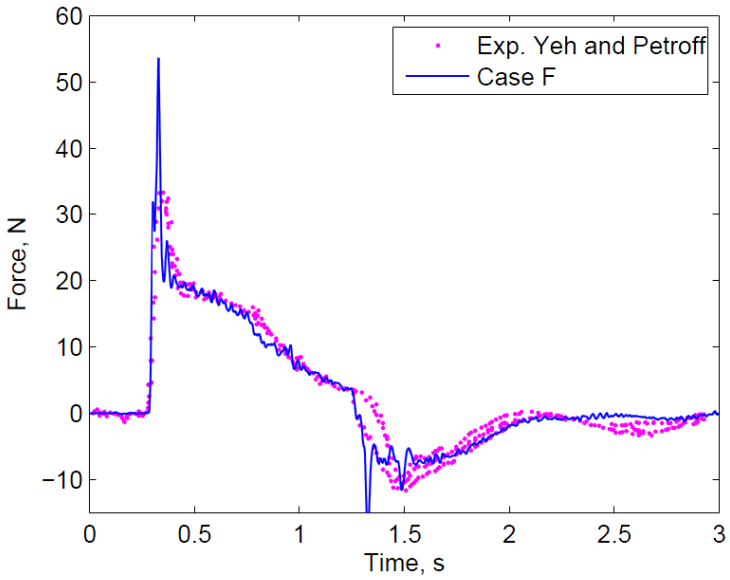

Figure 2-9 Numerical and experimental horizontal forces on column (Silvester and Cleary 2006)

The time at which the first force spike occurs, caused by the initial impact with the column, was also in agreement. The peaks, however, are over-predicted by the SPH model and this was attributed to the fact that the column is modelled as infinitely stiff as opposed to the experimental results that assumed the column had a specific stiffness causing there to be a response time when taking a reading using the load cell. Also, the sampling times for the SPH model were higher than those in the experimental model. The authors also concluded that the turbulence experienced in the SPH model was not really accounted for through air entrainment. This could affect where and when the forces spike in the experimental results, possibly creating a discrepancy.

Crespo *et al.* (2008) used a three-dimensional SPH model to validate the case of dike mitigation used in order to protect a tall structure such as a column. The layout used for this model is shown in Figure 2-9. The model was used to generate the forces produced by a dambreak bore impact on the column. Three cases of dikes; tall, intermediate and short, were considered and the resulting forces were compared to the dike-free case. It was shown that the impact forces on the column increased as the height of the dike decreased. Also, as the distance between the dike and the column was increased, the forces also increased due to the flow reforming beyond the dike and impacting the structure.

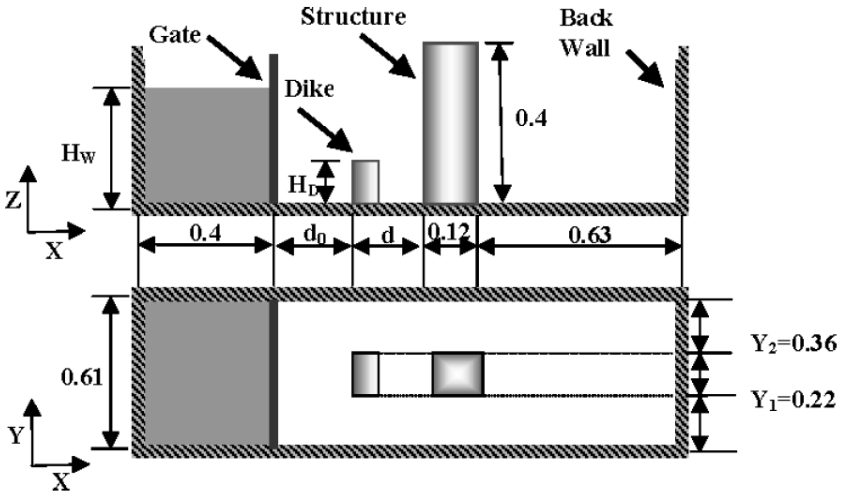

Figure 2-10 Configuration of numerical experiment (Crespo *et al.* 2008)

Viccionne *et al.* (2008) described that one of the most significant obstacles when using a semi-empirical model in design, is that they are developed for each specific setting, which may not always be the case depending on the structure in question. In this paper, the authors simulated the impact pressures on a seawall caused by sinusoidal waves produced by a paddle wavemaker. The predictor-corrector scheme was used to advance the time steps of the SPH model. There was a total of 70,000 particles with a repulsive boundary condition. The results were compared to another numerical model called Flow 3D. Flow 3D uses an Eulerian-based grid modelling technique. As shown in Figure 2-10 below, the surface profile produced by SPH in panel (a) and that by Flow3D in panel (b) shows a good agreement with some minor surface disturbances, which are likely due to the resolution of the SPH model. The pressure distribution on the wall due to the wave impact was also shown to be in good agreement.

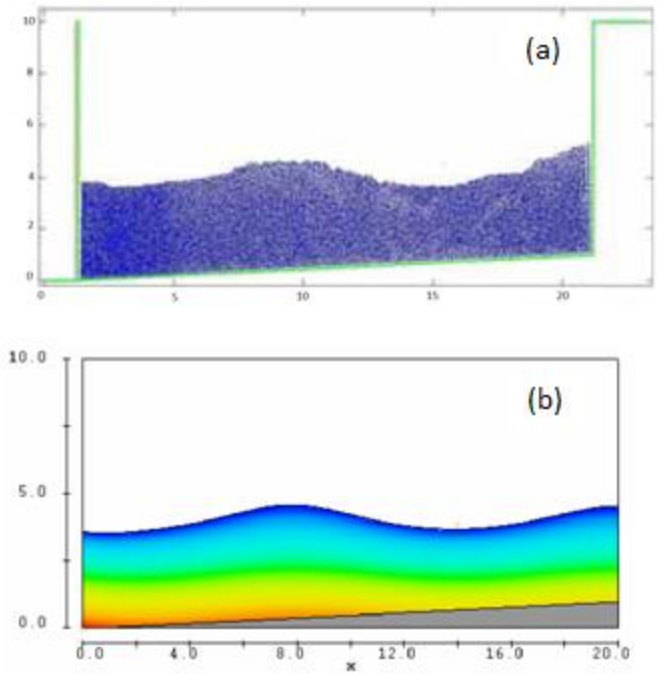

Figure 2-11 Comparison of numerical models at t = 20s (a) SPH (b) Flow 3D (Viccionne *et al.* 2008)

Didier and Neves (2010) conducted research on wave-structure interaction using SPH. The authors used a physical experiment performed in the large wave flume of the National Civil Engineering Laboratory in Portugal (LNEC). The flume is 73m long, 3m wide and 2m deep. A piston-type wavemaker was used to generate regular waves that propagated up a beach of 1:20 slope and impacted a seawall. The layout can be seen in Figure 2-11. The impermeable seawall has a slope of 2:3 and its crest is located at 1.684m from the bottom with a free board clearance [Rc] of 0.543m. The free board distance is that from the top of the seawall to the still water level. The regular waves had a period T of 3.79s, wave height H of 0.40m, wavelength l of 12.04m.

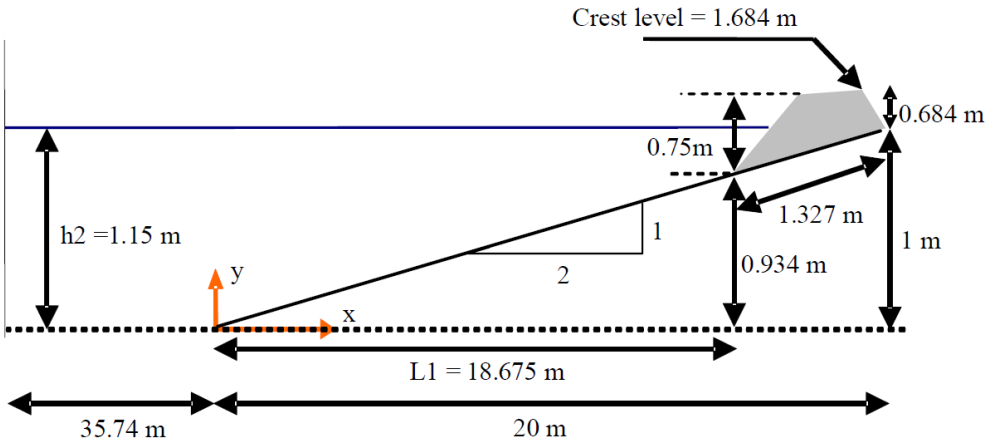

Figure 2-12 Schematic of experimental layout (Didier and Neves 2010)

In the SPH model, the key parameters used were a quadratic kernel for a weakly compressible fluid. This allowed the use of an equation of state for the pressure and compressibility was adjusted in order to slow the speed of sound so that the time step is reasonable. The predictor-corrector model (Monaghan 1994) was used to integrate the time. The boundary particles were chosen to behave under a repulsive boundary condition. The resolution chosen in the SPH model was not very high in order to allow for a reasonable computing time. Hence, the domain was reduced to a region that is 10m long. Figure 2-12 shows the free surface elevation of the SPH model compared to the physical experimental ones. The results show a very good agreement, especially in terms of the point at which the peaks start to emerge i.e. the wave period. There are very little differences in the values of the amplitudes and this is attributed to the difference in the phase between the harmonic and the fundamental frequency.

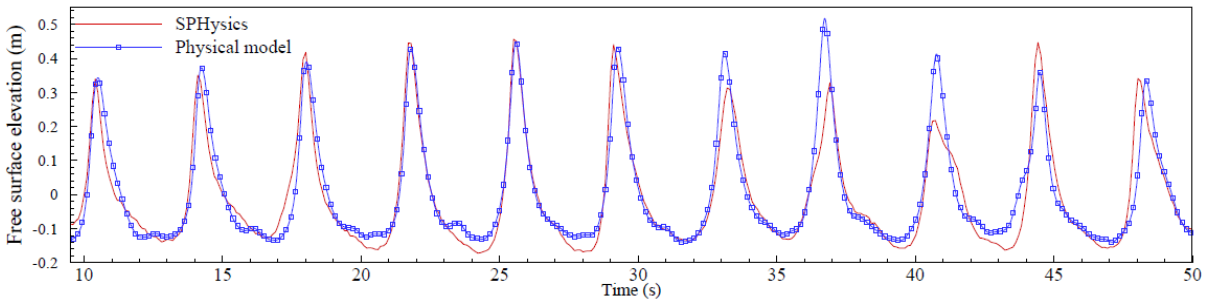

Figure 2-13 Free surface elevations from SPH model compared to physical model at distance x=11m (Didier and Neves 2010)

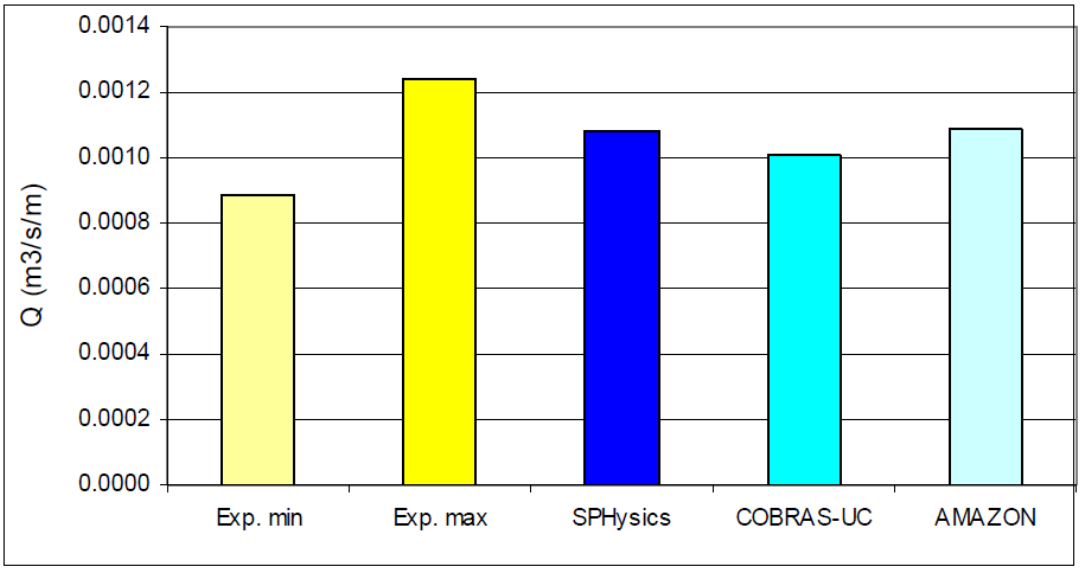

Figure 2-14 Overtopping rates from physical experiment compared to SPHysics, COBRAS-UC and AMAZON (Didier and Neves 2010)

The authors showed that the wave propagation was less sensitive to discretization than particle dimension. The conclusion was that, in general, there was a good agreement between the SPH results and the experimental ones. The mean overtopping flow rate was compared to other numerical models' results as well, such as AMAZON and COBRAS-UC, and the results are shown in Figure 2-13 above. All the models slightly under-predicted the maximum overtopping results from the LNEC experimental results and the conclusion was that it still fell within the range considered to be in good agreement.

Rogers *et al.* (2010) modelled the case of a caisson breakwater under the effect of impacting waves using SPH. The issue of the movement of the caisson due to the impact has not been researched enough and it is a major concern in the case of failure due to movement under extreme wave impacts. Rogers *et al.* (2010) suggested that there was no means to estimate how much a structure is expected to move under the impact of an extreme wave. The SPH model is compared to the physical experiments performed by Wang *et al.* (2006) for which the setup can be seen in Figure 2-14.

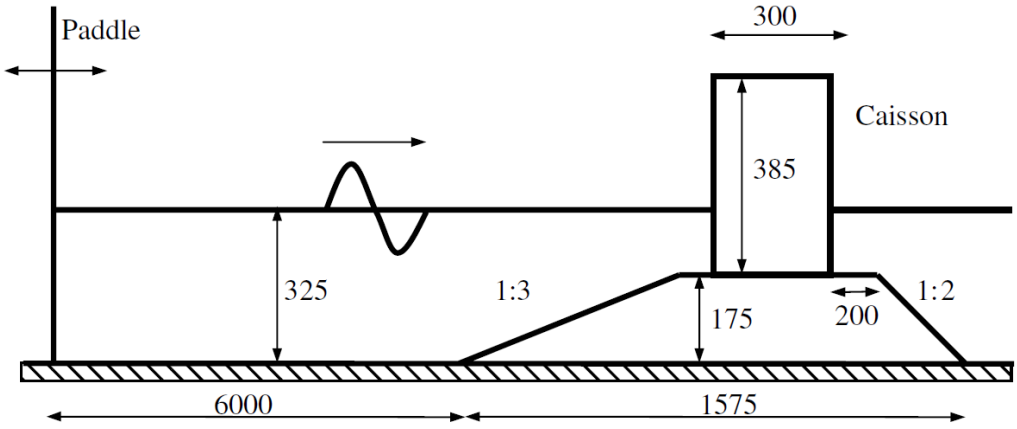

Figure 2-15 Simulation domain setup (units: mm) (Rogers *et al.* 2010)

Regular waves with a period of 1.3s were produced using the paddle such that they break right before reaching the caisson. The SPH model showed that the maximum forces were not registered at first, however with increasing resolution, the problem was resolved and agreement was seen. The displacements of the caisson detected by the SPH model were not as sudden as the ones in the actual data.

Shao (2010) used SPH to model the impacts of a solitary wave approaching a curtain-type breakwater. The computational domain is 10m long with a water depth of 0.2m. The selected particle spacing was 0.01m such that 20,000 particles were produced. The solitary wave was produced by a piton-type wavemaker with a wave height ratio ε of 0.3 and 0.6. A schematic layout of the curtain-type breakwater can be seen in Figure 2-15.

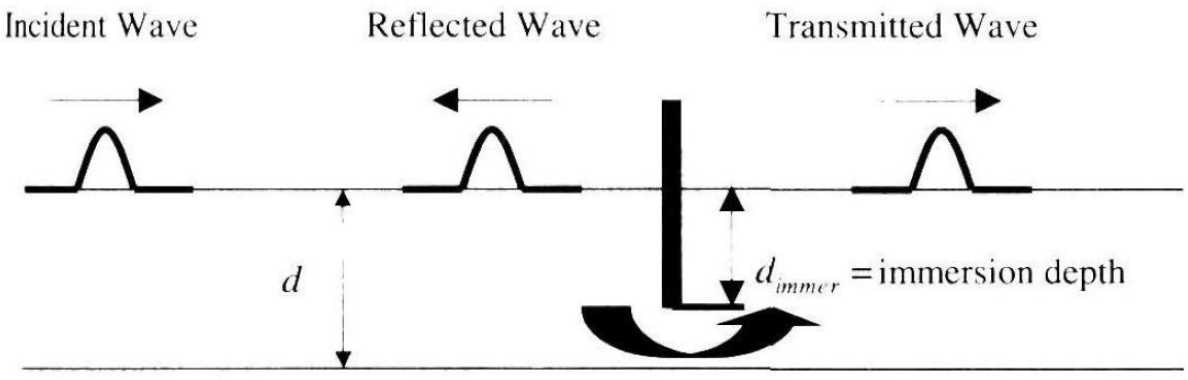

Figure 2-16 Sketch of a curtain-type breakwater (Shao 2010)

The author concluded that the curtain is capable of dissipating the energy in the wave when the curtain is immersed in over half the depth of the water. Also, the force-time history shows that the force peaked once for the smaller waves with ε of 0.3 but peaked twice for larger solitary waves with ε of 0.6. Shao (2010) attributed this to the fact that there may be an imbalance in the hydrostatic and dynamic forces as the wave impacts the curtain.

2.6 Discussion

Throughout section 2-1 to 2-5, a review of the state-of-the-art literature involving analytical, physical and numerical modelling of wave-structure interactions has been completed. It has been shown that there has been a significant amount of research conducted on the complex topic of wave-structure interactions; however, the results and conclusions of these publications are not yet satisfactory nor can they solely contribute to the improvement of existing design equations.

Most physical experiments involved using a wavemaker to generate regular waves that impact vertical walls [Kirkgoz (1990), Chan (1994), Azarmsa *et al.* (1996b)] as well as solitary waves that turn into bores and subsequently impact vertical walls [Ramsden and Raichlen (1990), Ramsden (1996), Schmidt *et al.* (1992)]. Other physical experiments investigated the effects of air entrainment in the wave front on the wave-structure interaction [Hattori *et al.* (1994), Bullock *et al.* (2007)]. The topic of waves overtopping a vertical wall is also not as common and was covered in research by Franco de Gerloni and van der Meer (1994) where they concluded that the most damage that occurs due to a wave-structure interaction is caused by overtopping. The majority of the physical experiments were conducted in wave flumes; however, they still cannot be considered truly “large scale” due to the use of only centimeters to tens of centimeters of water depth. This brings into the picture the need for more numerical model research in order to decrease the high costs associated with real large scale physical experiments. Moreover, using properly verified and calibrated numerical models, one can investigate wave-structure impact scenarios otherwise difficult to achieve in laboratory conditions.

A large amount of numerical experiments that involve wave-structure interactions are based on Navier-Stokes equations and they are applied in combination with varying turbulence models and surface tracking equations. The VOF method is widely used to track the surface profile of a numerical experiment such as in Xiao and Huang (2008) and Mokrani *et al.* (2010) and even iVOF in Wemmenhove *et al.* (2006). A major concern in the wave formation and structure interaction process is the amount of air entrainment in the wave front that requires a numerical model capable of handling multi-phase flows [Tanizawa and Yue (1992), Peregrine and Topliss (1994), Mokrani *et al.* (2010)]. A common model that was used to numerically model the wave-structure interaction is COBRAS [Liu and Al-Banaa (2004), Hsiao and Lin (2010)].

There have been several studies using SPH that focus on simulating a dam-breach case to produce a bore that impacts a slender vertical column or a vertical wall [Gomez-Gesteira and Dalrymple (2004), Silvester and Cleary (2006), Crespo *et al.* (2008)]. Lo and Shao (2002) showed that using LES alongside SPH was sufficient to model the shoaling and wave breaking process however it did not efficiently capture the turbulence in a wave bore. Other SPH experiments involved the use of a wavemaker to generate waves that impact vertical walls [Didier and Neves (2010), Rogers *et al.* (2010), Shao (2010)]. In most of these cases, the waves were accurately generated and the surface profiles were deemed acceptable.

As outlined in Chapter 1, the current research will attempt to add to the previously discussed body of research work by providing additional studies of cases of wave-structure interaction using a two-dimensional SPH model.

3. Introduction to Smoothed Particle Hydrodynamics in Hydraulic Modelling

3.1 Background

Smoothed particle hydrodynamics, developed by Gingold and Monaghan (1977) and Lucy (1977), was originally used to model astrophysical problems. In 1994, Monaghan showed its use in free surface flows. SPH utilizes a Lagrangian approach which is used to represent a fluid through individual particles each of which has assigned its own properties such as mass, density and velocity. Due to this arrangement, the method does not require a computational grid. Due to an initial pressure gradient, particles will be forced to move accordingly in time. This means that the values for velocity, pressure, etc., will change with time. The model obtains these values between assigned particles by interpolation. It can easily trace material interfaces, free surfaces and moving boundaries. The method has gained popularity in solving complex fluid dynamic problems because it is able to overcome some of the computational difficulties that are experienced when using mesh or grid-based Eulerian models. Many of the complications of the more traditional techniques, such as finite element method (FEM) or boundary element method (BEM) can make the numerical simulation expensive or, at times, inefficient. As with any method, there are drawbacks depending on the case that is considered. Before introducing these issues, an overview of the basic formulation behind SPH will be discussed.

3.2 Formulation

In a typical CFD problem, the most common method to model the process of wave breaking and wave-structure interaction employs the Navier-Stokes (NS) equations. The NS equations require the conservation of mass, conservation of momentum and conservation of energy. For the case of a Langrangian approach, as stated in Lin and Liu (2003), the NS equations for conservation of mass, momentum and energy can be respectively written as:

$$\frac{D\rho}{Dt} = -\rho \frac{\partial v^{\beta}}{\partial x^{\beta}} \qquad \textit{(3-1)}$$

$$\frac{Dv^{\alpha}}{Dt} = \frac{1}{\rho}\frac{\partial \sigma^{\alpha\beta}}{\partial x^{\beta}} \qquad \textit{(3-2)}$$

$$\frac{De}{Dt} = \frac{\sigma^{\alpha\beta}}{\rho}\frac{\partial v^{\alpha}}{\partial x^{\beta}} \qquad \textit{(3-3)}$$

where, v is the velocity, ρ is the fluid density, σ is the total stress tensor composed of the isotropic pressure and viscous stress, e is the internal energy and α and β denote the coordinate directions.

In the SPH method, each fluid particle is considered to be an interpolation node at which a function $[f]$ can be approximated according to the following:

$$f(x_i, t) = \int_j f(x_j) W(x_i - x_j, h)\, dx \qquad \textit{(3-4)}$$

where, x is the particle position, W is a weighting function more commonly known as a kernel and h is the kernel smoothing length. The subscripts i and j indicate the particle of interest and the particle surrounding it respectively. The weighting function $W(x_i - x_j, h)$will be denoted as W_{ij} in the remainder of the equations. The smoothing length determines to what extent the smoothing function will be effective in the domain around each particle.

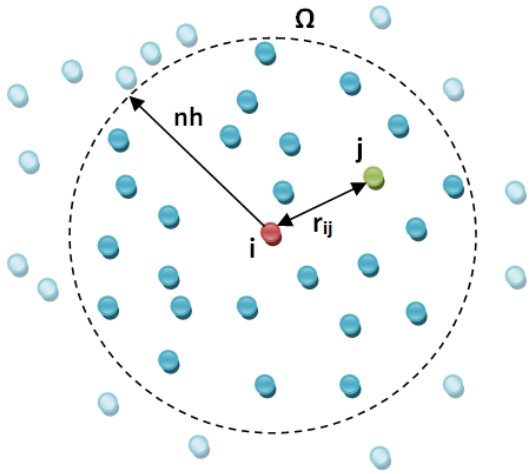

Figure 3-1 Illustration of particle influence domain in two-dimensional SPH

This influence domain, called Ω, is a circle of radius length *nh* for a two-dimensional model and a sphere for a three-dimensional model, with the same radius. These terms are illustrated in Figure 3-1 above. The dark blue circles represent water particles inside the influence domain. The red circle represents the particle of interest i and the green circle a neighbouring particle j. The kernel function must satisfy a few important conditions: its integral over the entire influence domain equals to one, it becomes equal to the Dirac delta function as h tends to a value of zero, it remains compact in that it equals zero outside the influence domain and positive on the inside and it maintains a monotonically decreasing behaviour. The Dirac delta function [δ] is given in equation (3-5).

$$\delta(\boldsymbol{x_i} - \boldsymbol{x_j}) = \begin{cases} 1, & \boldsymbol{x_i} = \boldsymbol{x_j} \\ 0, & \boldsymbol{x_i} \neq \boldsymbol{x_j} \end{cases} \quad \textbf{(3-5)}$$

The previously stated kernel function conditions are expressed in equations (3-6) to (3-10) for an influence domain of *nh*.

$$\lim_{h \to 0} W_{ij} = \delta(\boldsymbol{x_i} - \boldsymbol{x_j}) \quad \textbf{(3-6)}$$

$$\int_{\Omega} W_{ij}\, d\Omega = 1 \quad \textbf{(3-7)}$$

$$W_{ij} = 0 \quad for \quad |\boldsymbol{x_i} - \boldsymbol{x_j}| > nh \quad \textbf{(3-8)}$$

$$W_{ij} > 0 \quad for \quad |\boldsymbol{x_i} - \boldsymbol{x_j}| \leq nh \quad \textbf{(3-9)}$$

$$W_{ij} > W_{ik} \quad for \quad |\boldsymbol{x_i} - \boldsymbol{x_j}| < |\boldsymbol{x_i} - \boldsymbol{x_k}| \quad \textbf{(3-10)}$$

Additional details on the properties of different types of kernel are reviewed in Section 3.3. For the numerical approximation, the integral equation (3-4) is converted to the discretized form as the following:

$$f(x_i, t) = \sum_j W_{ij} f_j(x_j, t) \frac{m_j}{\rho_j} \quad \textbf{(3-11)}$$

where $m_j$and ρ_j are the mass and density of particle j, respectively. In SPH, the mass of the particle is held constant but the density is allowed to vary. Density variations are calculated

according to equation (3-12) below. In order to simulate the fluid in terms of the NS equations, the SPH approximation equation (3-4), is applied to the conservation of mass and momentum as follows:

$$\left(\frac{1}{\rho}\frac{d\rho}{dt}\right)_i = \sum_j \frac{m_j}{\rho_j}\boldsymbol{u}_j \nabla_i W_{ij} \tag{3-12}$$

$$\left(\frac{d\boldsymbol{u}}{dt}\right)_i = \sum_j m_j \left(\frac{p_j}{\rho_j^2} + \frac{p_i}{\rho_i^2}\right)\nabla_i W_{ij} - \boldsymbol{g} \tag{3-13}$$

where, $\boldsymbol{u}$ is the velocity vector, p is the pressure and $\boldsymbol{g}$ is the gravitational acceleration vector. A laminar viscosity condition is used in the simulations in order to model flows with low Reynolds numbers. In the case of wave breaking, the momentum equation must include a turbulence calculation. As originally proposed by Gotoh *et al.* (2001), the LES theory can be incorporated to the laminar viscosity terms such that the turbulence can be included (Shao *et al.,* 2005). This condition is known as sub-particle scale (SPS) and hence the final form of the conservation of momentum is shown in the following equation:

$$\left(\frac{d\boldsymbol{u}}{dt}\right)_i = \sum_j m_j \left(\frac{p_j}{\rho_j^2} + \frac{p_i}{\rho_i^2}\right)\nabla_i W_{ij} - \boldsymbol{g} + \sum_{\boldsymbol{j}} m_{\boldsymbol{j}} \left(\frac{\boldsymbol{4v}(\boldsymbol{x}-\boldsymbol{s})\nabla_i W_{ij})}{(\boldsymbol{\rho_j}+\boldsymbol{\rho_i})(\boldsymbol{r_{ij}^2})}\right)\boldsymbol{u_{ij}} + \sum_{\boldsymbol{j}} m_{\boldsymbol{j}} \left(\frac{\boldsymbol{\tau_j}}{\boldsymbol{\rho_j^2}} + \frac{\boldsymbol{\tau_i}}{\boldsymbol{\rho_i^2}}\right)\nabla_i W_{ij} \tag{3-14}$$

where $\boldsymbol{v}$ is the viscosity of laminar flow, $\boldsymbol{r_{ij}}$ is the distance between i and j and $\boldsymbol{\tau}$ is the sub-particle stress tensor. The first part of the right-hand side of equation (3-9), up till and including [$-\boldsymbol{g}$], represents the pressure gradient. The next term represents the laminar viscosity according to Morris *et al.* (1997) and finally, the third term accounts for the SPS turbulence model as proposed by Shao *et al.* (2005). After all equations are set up in the model, they are solved using a time-stepping method that either considers a single step, such as Verlet (Verlet 1967), or two steps such as the predictor-corrector model.

3.3 Parameter Description

3.3.1 Time Stepping

In SPH, the model can be progressed through time using four different techniques. The predictor-corrector algorithm, as proposed by Monaghan in 1989, such that the velocity, density, position and energy are first calculated at every time step and corrected with the forces at the half-time step. From these predicted values, the pressure can then be calculated from the resulting density at each time step. Monaghan (1989) suggested that the computational time for a numerical model could be halved if the midpoint values for the calculations are used to predict the next time step. The Verlet theorem (Verlet 1967) can also be used at the time steps. It involves using two sets of equations based on a third-order Taylor expansion series as shown in equations (3-15) to (3-18).

$$u_a^{n+1} = u_a^{n-1} + 2\Delta t F_a^n \quad \textit{(3-15)}$$

$$\rho_a^{n+1} = \rho_a^{n-1} + 2\Delta t D_a^n \quad \textit{(3-16)}$$

$$r_a^{n+1} = r_a^n + \Delta t V_a^n + 0.5\Delta t^2 F_a^n \quad \textit{(3-17)}$$

$$e_a^{n+1} = e_a^{n-1} + 2\Delta t E_a^n \quad \textit{(3-18)}$$

Using these equations, the variables are evaluated at each time step; however, for every M number of steps of order 50, the variables are found using equations (3-19) to (3-22) in order to avoid divergence of the time integration.

$$v_a^{n+1} = v_a^n + \Delta t F_a^n \quad \textbf{(3-19)}$$

$$\rho_a^{n+1} = \rho_a^n + \Delta t D_a^n \quad \textbf{(3-20)}$$

$$r_a^{n+1} = r_a^n + \Delta t V_a^n + 0.5\Delta t^2 F_a^n \quad \textbf{(3-21)}$$

$$e_a^{n+1} = e_a^n + \Delta t E_a^n \quad \textbf{(3-22)}$$

The third method of time stepping is the Symplectic algorithm proposed by Leimkhuler in 1997. Finally, the last method of time stepping is using the Beeman algorithm (Beeman 1976). The Beeman algorithm is essentially the same as the Verlet (Verlet 1967) method; however, the velocity is interpolated using a different equation. The reader is directed to the mentioned sources for more details and equations that are not provided here.

The value of the time step must be calculated by taking into account the force terms and the viscous diffusion term using the Courant Friedrichs Lewy (CFL) condition due to the explicit nature of the approximation equations. According to Monaghan and Kos (1999) the CFL condition is applied to the time step as follows:

$$\Delta t = \varsigma_{CFL} \times \min(\Delta t_f, \Delta t_{cu}) \quad \textbf{(3-23)}$$

$$\Delta t_f = \min_i\left(\sqrt{h/|f_i|}\right) \quad \textbf{(3-24)}$$

$$\Delta t_{cu} = \min_i \frac{h}{c_i + \max_j \left|\frac{h\mathbf{u_{ij}} \cdot \mathbf{x_{ij}}}{{r_{ij}}^2}\right|} \quad \textbf{(3-25)}$$

where $\varsigma_{CFL}$is the CFL constant taken of the order of magnitude of 0.1, Δt_f is the time step depending on the force per unit mass, Δt_{cu} is the combined Courant viscosity time step, $|f_i|$ is the force per unit mass, c_i is the reference speed of sound at particle i, $\mathbf{u_{ij}}$ is the velocity vector between particles i and j and $\mathbf{x_{ij}}$ is the distance vector between the particles.

3.3.2 Boundary Conditions

Since there is no computational grid, the numerical model must take place inside a specific boundary or region of particles that exhibit special properties. There are various conditions that can be placed on a boundary particle such as the dynamic boundary condition and the repulsive boundary condition. As suggested by Crespo *et al.* (2007) and Dalrymple and Knio (2000), the dynamic boundary condition (DBC) sets the particles to behave the same way as fluid particles; however, they can either remain in fixed positions or move following a

separate function designed for specific purpose such as the particles of a wavemaker paddle or a dam breach gate. The repulsive boundary condition (RBC), as proposed by Monaghan and Kos (1999) and Rogers and Dalrymple (2008), was developed such that the fluid particles remain inside the boundaries. This is achieved by setting the force exerted by the boundary particle on an incoming fluid particle to be equal and opposite in direction of the force by that incoming particle.

3.3.3 Density Filter

In order to assure that the density calculation at each time step stays within a realistic range, a filter can be used to re-calculate the density every few time steps (Colagrossi and Landrini 2003). This effect is caused by the density oscillations due to the progression of the numerical model. There are two types of filter formulations that can be used. The first is Shepard's equation which is of zero-order and in which the density of each particle is re-calculated every thirty time steps. The Moving Least Squares (MLS) method is another filter that can be used at every thirty or so time steps. It is a first-order correction and was initially developed by Dilts (1999) and later used in SPH by Colagrossi and Landrini (2003).

3.3.4 Kernel Correction

The kernel corrector and kernel gradient corrector can be applied to the SPH simulation in order to avoid the failure of a kernel function, hence renormalizing it. The particle in question is not always completely surrounded by particles at the boundary regions. As a result, the kernel becomes shortened and loses its correct interpolating capabilities. Failure occurs mainly in the cases of free surface flows where the surface particles are not completely surrounded by other fluid particles. The kernel correction was originally introduced by Li and Liu (1996) in the form of a Fourier transform function. This involves maintaining the kernel condition of unity by dividing equation (3-11) by equation (3-7). The kernel gradient correction was first introduced by Bonet and Lok (1999). A correction matrix is involved in the calculation of the kernel gradient equation in order to satisfy conservation of momentum. This is known to slightly increase the computational time due to the additional calculations introduced by the matrix at each time step. The reader is directed

towards the papers by Bonet and Lok (1999) and Li and Liu (1996) for additional details in the mathematical formulation of the correction.

3.3.5 Viscosity Treatment

The viscosity of the fluid to be modeled is a main factor involved in the conservation of momentum. Depending on which viscosity treatment, the equation for conservation of momentum will have different diffusion terms. As previously mentioned, there can be different viscosity considerations depending on the case to be modeled. These are artificial viscosity (Monaghan 1992), laminar viscosity (Lo and Shao 2002) and laminar with SPS viscosity (Gotoh *et al.* 2001). Generally the three cases are incorporated respectively into the momentum equation as follows:

$$\left(\frac{d\boldsymbol{u}}{dt}\right)_i = \sum_j m_j \left(\frac{p_j}{\rho_j^2} + \frac{p_i}{\rho_i^2} + \Pi_{ij}\right) \nabla_i W_{ij} - \boldsymbol{g} \tag{3-26}$$

$$\left(\frac{d\boldsymbol{u}}{dt}\right)_i = \sum_j m_j \left(\frac{p_j}{\rho_j^2} + \frac{p_i}{\rho_i^2}\right) \nabla_i W_{ij} - \boldsymbol{g} + \sum_{\boldsymbol{j}} m_{\boldsymbol{j}} \left(\frac{4\boldsymbol{v}(\boldsymbol{r_{ij}}) \nabla_i W_{ij})}{(\rho_j + \rho_i)(\boldsymbol{r_{ij}^2})}\right) \boldsymbol{u_{ij}} \tag{3-27}$$

$$\left(\frac{d\boldsymbol{u}}{dt}\right)_i = \sum_j m_j \left(\frac{p_j}{\rho_j^2} + \frac{p_i}{\rho_i^2}\right) \nabla_i W_{ij} - \boldsymbol{g} + \sum_{\boldsymbol{j}} m_{\boldsymbol{j}} \left(\frac{4\boldsymbol{v}(\boldsymbol{r_{ij}}) \nabla_i W_{ij})}{(\rho_j + \rho_i)(\boldsymbol{r_{ij}^2})}\right) \boldsymbol{u_{ij}} + \sum_{\boldsymbol{j}} m_{\boldsymbol{j}} \left(\frac{\boldsymbol{\tau_j}}{\rho_j^2} + \frac{\boldsymbol{\tau_i}}{\rho_i^2}\right) \nabla_i W_{ij} \tag{3-28}$$

where the term Π_{ij} is defined as:

$$\Pi_{ij} = \begin{cases} \dfrac{-\alpha \overline{c_{ij}} \rho_{ij}}{\overline{\rho_{ij}}}, & \boldsymbol{v_{ij} r_{ij}} < 0 \\ 0, & \boldsymbol{v_{ij} r_{ij}} \geq \boldsymbol{0} \end{cases} \tag{3-29}$$

The term α refers to a constant that can be defined depending on the numerical case. In the SPH model, α is called the 'viscosity value'.

3.3.6 Vorticity

The vorticity in fluid dynamics is generally defined as the local angular rate of rotation of the fluid particles. According to Monaghan (1992), the vorticity, ω of a particle i, in SPH can be calculated using the following formula:

$$\omega_i = \sum_j m_j \left(\frac{\boldsymbol{v}_{ij}}{\rho_i}\right) \nabla W_{ij} \qquad \textbf{(3-30)}$$

The significance of the vorticity is to be able to model the rotational motion of the fluid particles in specific applications such as wave-breaking and turbulent bore propagation. The vorticity is considered negative when the rotation is counter-clockwise or away from the direction of motion of the propagating fluid.

3.3.7 Equation of State

In order to solve the given SPH formulation, there must be a relationship between the density of a particle and its pressure. This relation can be in the form of the Tait state equation (1888), the Ideal gas equation or Poisson equation. The equation of state (EoS) proposed by Batchelor (1974) and Monaghan (1994) and based on Tait's equation (Neece and Squire, 1968) for compressibility, is as follows:

$$p = B\left[\left(\frac{\rho}{\rho_o}\right)^{\gamma} - 1\right] \qquad \textbf{(3-31)}$$

$$B = \frac{c_o^2 \rho_o}{\gamma} \qquad \textbf{(3-32)}$$

$$c_o = \sqrt{b^2 gH} \qquad \textbf{(3-33)}$$

where γ is a constant assumed to be 7, ρ_o is the reference water density (1000kg/m^3), c_o is the reference speed of sound, b is a value between 10 and 40 dependant on the case and H is the water depth. This relation allows for the pressure to be calculated directly from the densities of the fluid particles at each time step and greatly simplifies the calculations, as opposed to using the Poisson equations which are partial differential equations. It is recommended that the speed of sound is set to around ten times the maximum expected wave velocity in the simulation (Monaghan 1994). This is done by setting a value for b and

hence B such that the aforementioned condition holds true. The fluid is assumed to follow a weakly compressible SPH model (WSPH) when using the Tait EoS. For an incompressible fluid used in incompressible SPH models (ISPH), the Poisson equation may be used. For details of the derivation of an incompressible EoS, the reader is referred to research done by Lee *et al.* (2008). Lee *et al.* (2008) conducted comparisons between using an ISPH and WSPH model and found that in some cases the ISPH model gave less of an error.

3.3.8 The Riemann Solver

During a WSPH simulation, the particles' velocities and pressures tend to fluctuate causing inaccuracies in the interpolations. This can be avoided using a Riemann Solver (Gomez-Gesteira *et al.* 2010b). The basic idea behind a Riemann problem, in the case of SPH, is that the pressures and/or velocities of fluid particles become inconsistent from one location to the next which would then introduce errors in the calculation. Hence, the Riemann solver can be used to neutralize this effect. Various types of Riemann solvers can be used in SPH such as a conservative model (Vila 1999) or a non-conservative model (Parshikov *et al.* 1999). Gesteira *et al.* (2010) define the region between the two locations over which a discontinuity in the pressure or velocity occurs as the "star" region. The pressure in the star region P_{ij}*, is defined to be the average of the pressures of the particles on either side. In the non-conservative model, the only difference that using a Reimann solver makes is in the equation of momentum conservation where the pressures P_i and P_j are replaced by P_{ij}*. A parameter called the slope limiter, or β-limiter, is shown to significantly affect the level of disturbances in the surface of the fluid during the computation. If the value of the β-limiter is decreased and approaches one, it tends to cause diffusive results that will in turn reduce the values of the force on a structure. On the other hand, if the value of the β-limiter is increased and approaches two, an increase in the wave height will occur possibly causing an unrealistic behaviour of the fluid particles.

3.4 Advantages and Disadvantages of Smoothed Particle Hydrodynamics Method

Every modelling theory has both its advantages and disadvantages which depend on what type of problem is being modelled. Features of SPH that are beneficial to one application may be a source of errors for another. An advantage of SPH is that it can guarantee mass conservation without any additional computation. This is because the conservation of mass is one of the basic formulations from which SPH equations are derived. Unlike the more common methods such as FEM or finite volume method (FVM), SPH can approximate the particle pressures from the neighbouring particles and does not require solving of linear systems of equations. The pressures can easily be obtained from a direct equation representing the weighted contribution of neighbouring particles. It has proven to be more robust and easy to implement comparing to many other methods. In some cases, a considerably accurate result can be achieved even with a lower resolution of particles. This allows for shorter computing time. It has been shown that SPH can easily handle the wave-breaking process such as the formation of a breaking roller. It is able to efficiently resolve problems with a fluid media that experiences highly non-linear deformations. According to Monaghan (2005), the turbulence in a fluid model can be introduced through the use of a Lagrangian equation or even an LES. The introduction of higher order kernels will produce more accurate and elaborate time steps, however this may result in further inaccuracies depending on the kernel chosen. A very important step is choosing the correct kernel approximation function. For example, the most popular splines tend to be a cubic or quadratic. The derivative of a cubic spline kernel function increases as distance between the particles decreases; however, it begins to decrease due to a local maximum. This doesn't fit the assumption in SPH which is based on less influence with increasing distance. This is shown in Figure 3-2 below, which displays how the functions and their derivatives vary with the distance between particles.

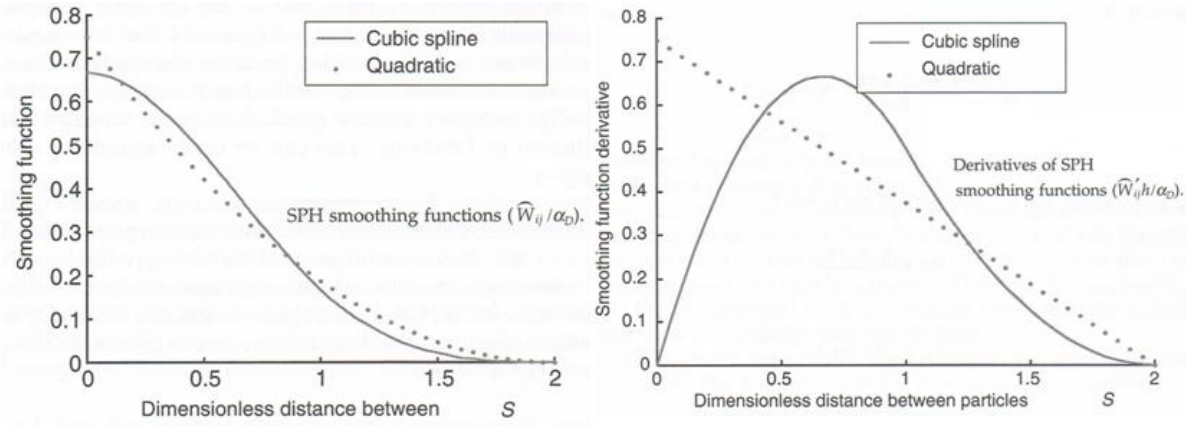

Figure 3-2 Schematic examples of a cubic and quadratic spline functions (Liu and Liu 2003)

One of the main disadvantages of SPH is that at high particle resolutions in large domains, it requires a higher computational time due to the addition of particles per time step. This can be resolved through the use of parallel processers or graphical processing units (GPU) as it will be discussed in the next section. Also, if the smoothing length is too small, then not enough particles are created. Conversely, if the smoothing length is too large then all the details of the particles will be smoothed or averaged out. Additional details of the effects of using the features of SPH will be discussed in the next chapters as the results of the experiments are presented.

3.5 Open Source Code

Through the collaboration of three different universities; University of Vigo in Spain, John Hopkins in the United States, and Manchester University in the United Kingdom, open source codes have been developed which allow the use of the SPH theory to solve various types of free-surface flow problems. There are various versions of these source codes each involving a different processing method and allowing for different features. A brief description of these versions is given in the following subsections. The general execution methodology for SPHysics is given in the flow chart shown in Figure 3-3.

3.5.1 SPHysics

The most basic version of the open source code, called SPHysics, is written in FORTRAN programming language for both a two-dimensional (2D) and a three-dimensional (3D) type

problem. This code requires only a central processing unit (CPU) in order to execute. Parallel SPHysics is another option of the code that allows for the use of multiple CPU's in tandem to perform the simulations.

The SPHysics model allows the introduction of obstacles, such as trapezoidal seawalls, through the definition of their coordinates with respect to the origin. The user can also add a slope in the bottom of the domain to create a beach-type geometry. Waves can be modelled by various methods such as using paddle- or piston-type wave-makers with a prescribed motion input file or simply a sine wave equation. Other features include the modelling of floating objects and dam-breach cases using gates. The experimental case can be run in both 2D and 3D although the 3D model will require more running time due to the significantly higher number of particles required to model the same domain. A visualization of the case with a wave formed by a paddle on a beach using SPHysics can be seen in both 2D and 3D in Figure 3-4. For additional details about the code's structure and implementation, the reader is referred to the SPHysics user manual (Gomez-Geistera *et al.* 2010a).

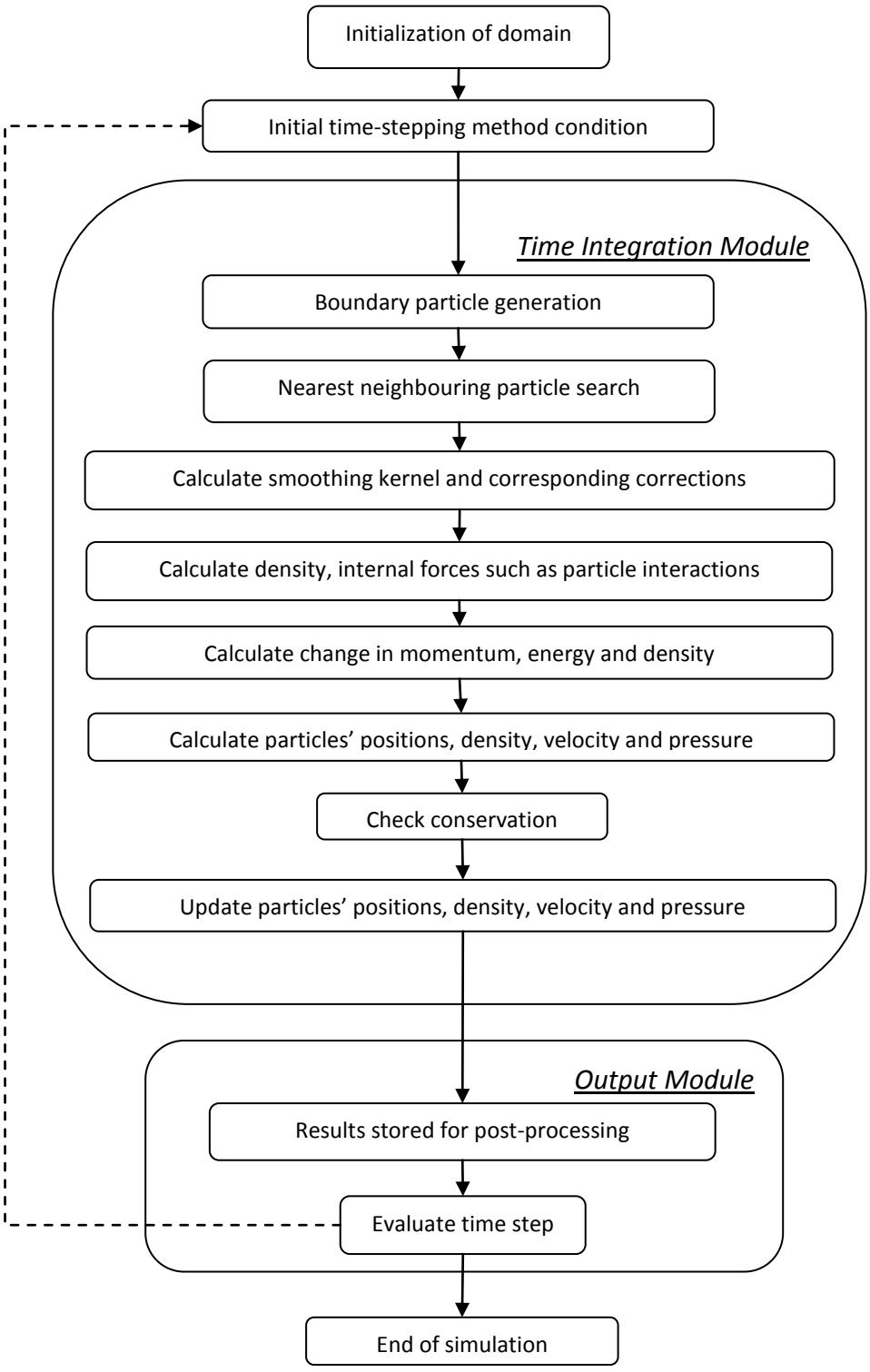

Figure 3-3 General execution methodology of SPHysics

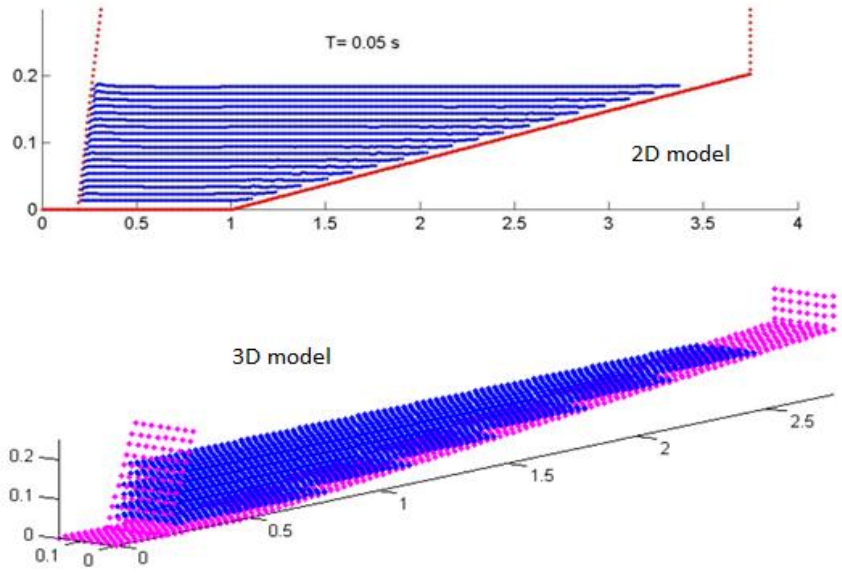

Figure 3-4 Case example from SPHysics for a wave paddle on a beach (adapted from Gomez-Geistera *et al.* 2010)

3.5.2 DualSPHysics

This version of the code requires a CPU and a graphical processing unit (GPU) running simultaneously. The advancement in the performance of GPU's has made high performance computing more readily available as an option for CFD specially in the case of extremely large experimental domains. DualSPHysics can be applied to produce results of a two- or three-dimensional case and is coded in C++ and Compute Unified Device Architecture (CUDA) programming languages (Crespo *et al.* 2011). This allows the user to take advantage of using a GPU which significantly speeds up the simulations. This version of the open source code also allows for modelling of more realistic engineering applications by incorporating the real spatial geometry in three-dimension. A sample output of DualSPHysics for the 3D case of a dam breach wave impacting a wide vertical wall is shown in Figure 3-5.

DualSPHysics allows for the simulation of the same sample cases as in SPHysics. Most SPH formulations utilized are the same for both SPHysics and DualSPHysics with the exception of a few that are currently being developed such as the Riemann solver and periodic open boundaries. DualSPHysics enables the user to import a Computer Aided Design (CAD) file to

be used as a numerical domain for the experiment to be run. For additional details about the code's structure and implementation, the reader is referred to the SPHysics user manual (Gomez-Geistera *et al.* 2012a, 2012b).

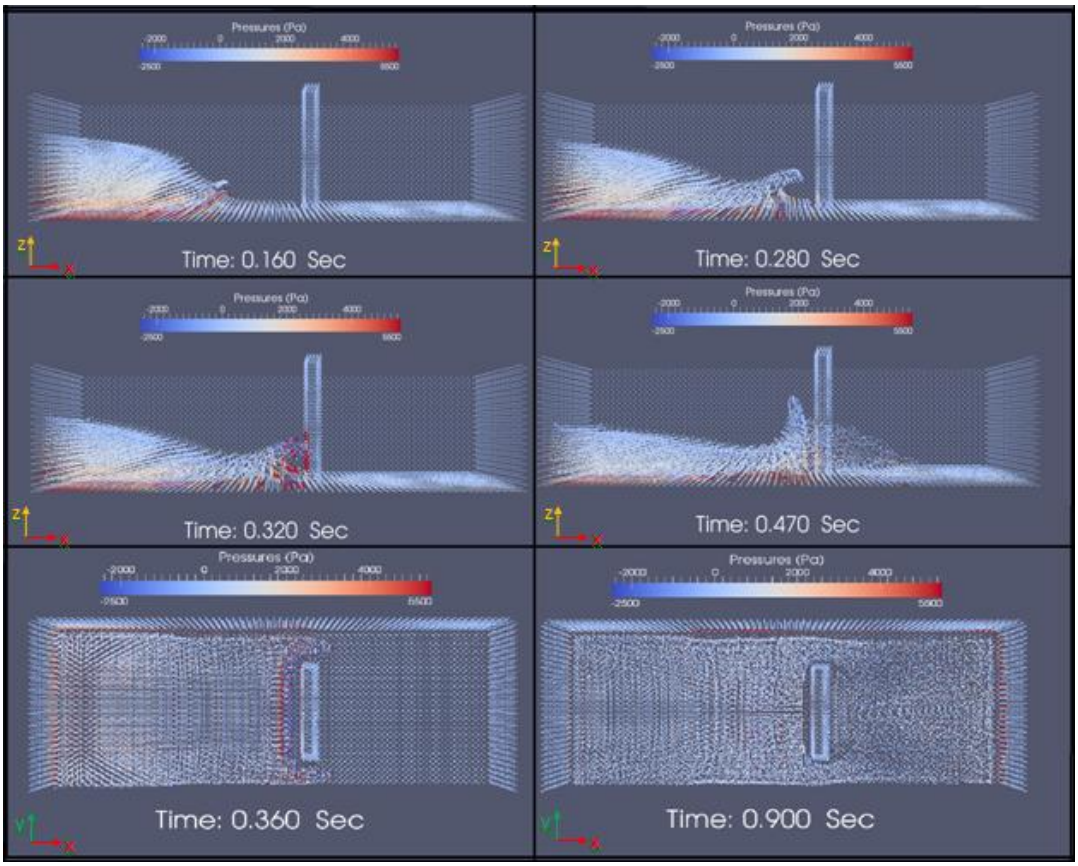

Figure 3-5 Several views of a sample case of dam breach impact on wall using DualSPHysics

3.5.3 GPUSPH

Another useful version of the SPHysics code is GPUSPH developed by Hérault *et al.* (2010). This code is also written in C++ and CUDA and is developed for Linux and MacOS operating systems. As with DualSPHysics, an NVIDIA graphics card that is compatible with CUDA is required in order to use the GPUSPH code. GPUSPH includes example cases simulating a variety of scenarios such as dam breaks and solitary wave formation using wave paddles. Currently, the developers are working on the option of uploading and using a digital elevation model image as the experimental domain. A sample case involving a dam breach

case that generates a bore that impacts a wall was simulated using GPUSPH and can be seen in Figure 3-6.

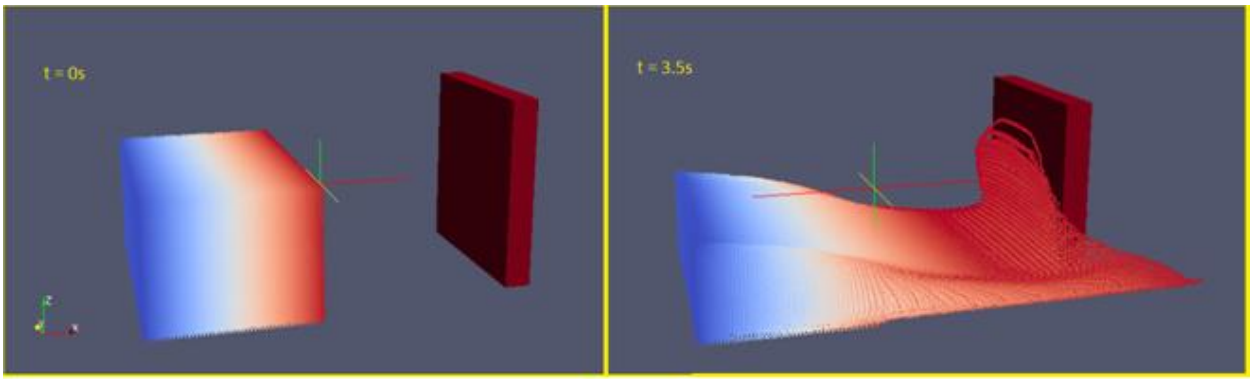

Figure 3-6 Sample model output of dam breach bore impact on wall using GPUSPH

3.5.4 Discussion

Three open source codes, all based on the SPH model, have been introduced and briefly summarized in order to provide the reader with some insight regarding the options available for this study. Since the experimental cases that will be presented in the coming chapters involve wide coastal protection structures attacked by oblique waves, the wave/bore impact is assumed to be symmetric along the width of the front face of the walls allowing the numerical modelling to be completed in 2D. This means that a 2D model should be sufficient to complete this study without the unnecessary additional computational time resulting from the use of a 3D model. As a result, the most suitable version, SPHysics, was used to simulate the physical experiments for this study.

4. Physical Models Description

In order to test the accuracy and robustness of the SPHysics model, three physical model experiments were numerically simulated using this model. The results of the numerical model were then compared with their corresponding physical results obtained in each experiment. Although these physical experiments were already presented in the literature review of this study, a more detailed description of each of them is provided in sections 4.1 through 4.3.

4.1 Test Case 1: Ramsden (1993)

For his PhD thesis in 1993, Ramsden presented results of laboratory experiments investigating the impact of an instrumented wall with solitary waves, surges and turbulent bores in a wave tank. It was possible to set the tank such as to accommodate both horizontal and tilted tank floors. The study aimed to discuss the effects of translational waves rather than breaking waves.

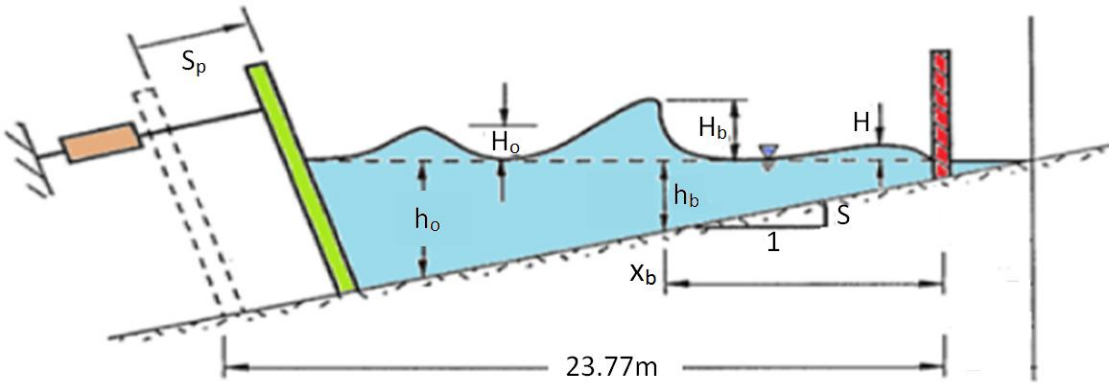

Figure 4-1 Experimental setup for titling tank for Ramsden (1993) (adapted from Ramsden, 1993)

A piston wave maker was used to generate solitary waves that travelled up on a beach with a mild slope, S = 0.02. The wave tank has a width of 1.10m, length of 40m and height of 0.61m. As shown in Figure 4-1, the wave had an initial wave height, H_o, water depth, h_o, and breaks at a distance, x_b, from the wall. Numerous wave height conditions were used but for

the purposes of this research, only the case with an initial relative wave height [H_o/h_o] of 0.288m is considered. The initial experimental conditions are shown in Table 4-1.

Table 4-1 Wave and wave-induced bore conditions according to Ramsden (1993)

H_o/h_o (m/m)	H_b (cm)	H_b/h_b (m/m)	x_b (m)	H (cm)	c_w (cm/s)
0.288	20.3	1.21	15.6	4.9	126

The wave-induced turbulent bore propagates towards the wall with a celerity, c_w, and impacts the wall with a maximum bore height, H. The vertical wall extends along the entire width of the flume and the SWL at the wall was 5mm. In order to maintain a water depth of 5mm at the wall with a tank bed slope of 0.02, the initial water depth near the piston [h_o] is 0.47m. Three force transducers were installed on the wall; two at an elevation of 3.83 cm and one at 34.31cm from the bottom. Figure 4-2 below shows the layout of the instrumented wall. The time-history of the wave profile and motion was captured using high speed video cameras. The same experiment was performed in a horizontal wave tank where the waves and bores were generated either by using a wavemaker or by releasing a volume of water impounded behind a pneumatic gate.

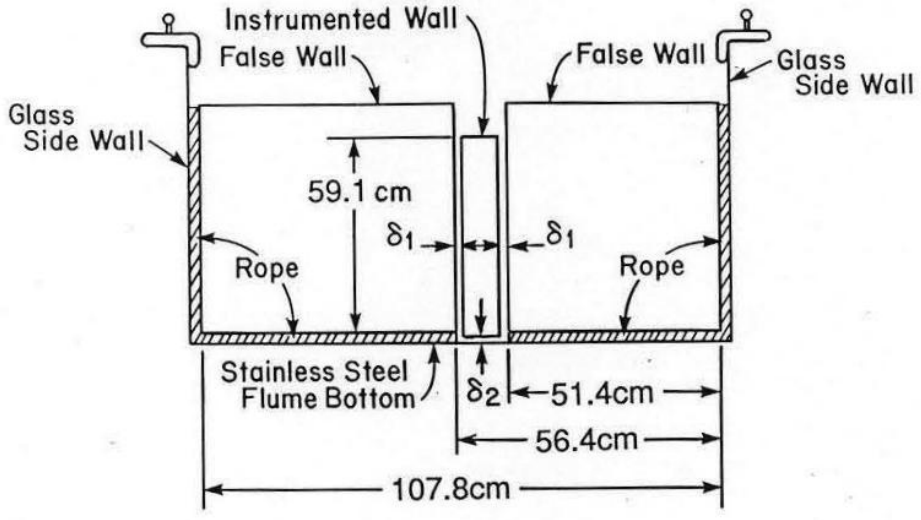

Figure 4-2 Schematic drawing showing dimensions of instrumented wall from Ramsden (1993)

4.2 Test Case 2: Esteban *et al.* (2008)

In 2008, Esteban *et al.* conducted research on the effects of various types of solitary waves on the failure mechanisms of armoured caisson layouts. A series of experiments were carried out at Yokohama National University, Japan, in a wave flume that is 15.3m long, 0.6m wide and 0.55m wide. The solitary-type waves were generated by releasing water from a dam capable of holding varying impoundment depths. In the downstream section, the bed slopes at an angle of 1:10 followed by a trapezoidal gravel mound on top of which a rectangular caisson rests. The details of the dimensions of the wave flume are shown in Figures 4-3 to 4-5 below.

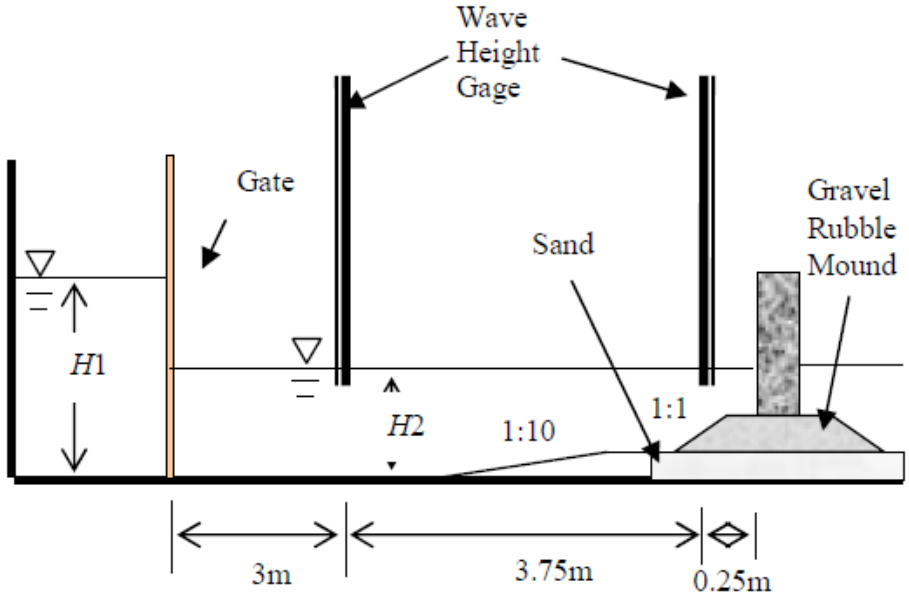

Figure 4-3 Experimental wave tank layout from Esteban *et al.* (2008) (Esteban *et al.* 2008)

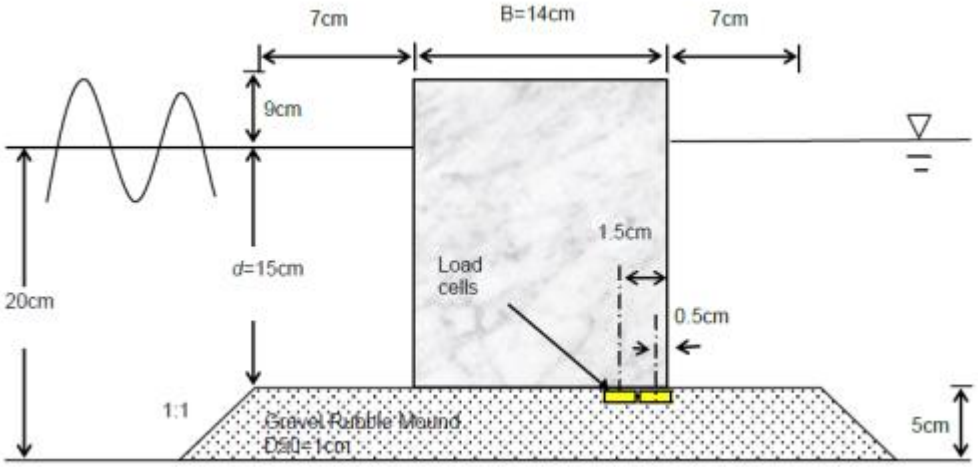

Figure 4-4 Close-up of front face of caisson setup from Esteban *et al.* (2008) (Esteban *et al.* 2008)

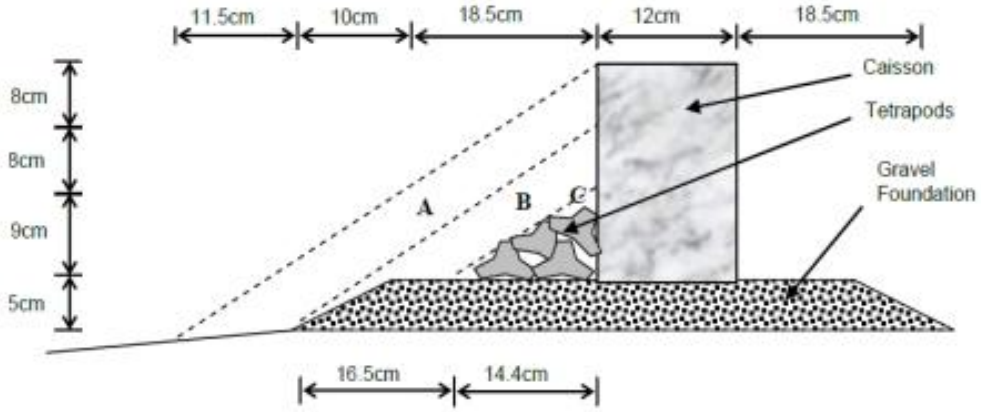

Figure 4-5 Close-up of side view of caisson and armouring setup from Esteban *et al.* (2008) (Esteban *et al.* 2008)

Since the overturning of the caisson is one of the main concerns, a load cell was placed at the heel of the caisson in order to measure the forces that would induce an overturning moment. Essentially, the vertical critical load was measured. The armour setup, as shown in Figure 4-5, ranges from layout A, with the largest amount of Tetrapods placed on top of the gravel mound, to layout D, with no armouring units. Three wave impact cases were tested: a bore type wave, one that breaks on the caisson and one that almost breaks near the caisson. In order to achieve these three cases, the impoundment depth was changed from 15cm to 17cm and finally to 19cm. The SWL was kept constant at 15cm. In this study, only the case of impoundment depth of 17cm was considered with layout D.

4.3 Test Case 3: Hsiao and Lin (2010)

Hsiao and Lin (2010) conducted a series of laboratory experiments in order to test the effects of a solitary wave impacting and overtopping an impermeable seawall through the use of a numerical solver called COBRAS (Hsiao and Lin 2010). The length of the flume used was 22m with a width of 0.5m and a height of 0.75m and the tests were performed in the Tainan Hydraulic Laboratory in China. A piston–type wave maker was used to generate solitary waves through the application of Goring's mathematical model from 1978 (Goring 1978). During a typical experimental run, the solitary wave approaches a beach of slope 1:20 and impinges a seawall caisson of seaward slope 1:4 and landward slope 1:1.8. Pressure transducers were placed along both the seaward and landward sides of the caisson in order to measure the impact loads due to the oncoming waves. A sketch of the layout and locations of the transducers as well as the experimental domain is shown in Figure 4-6.

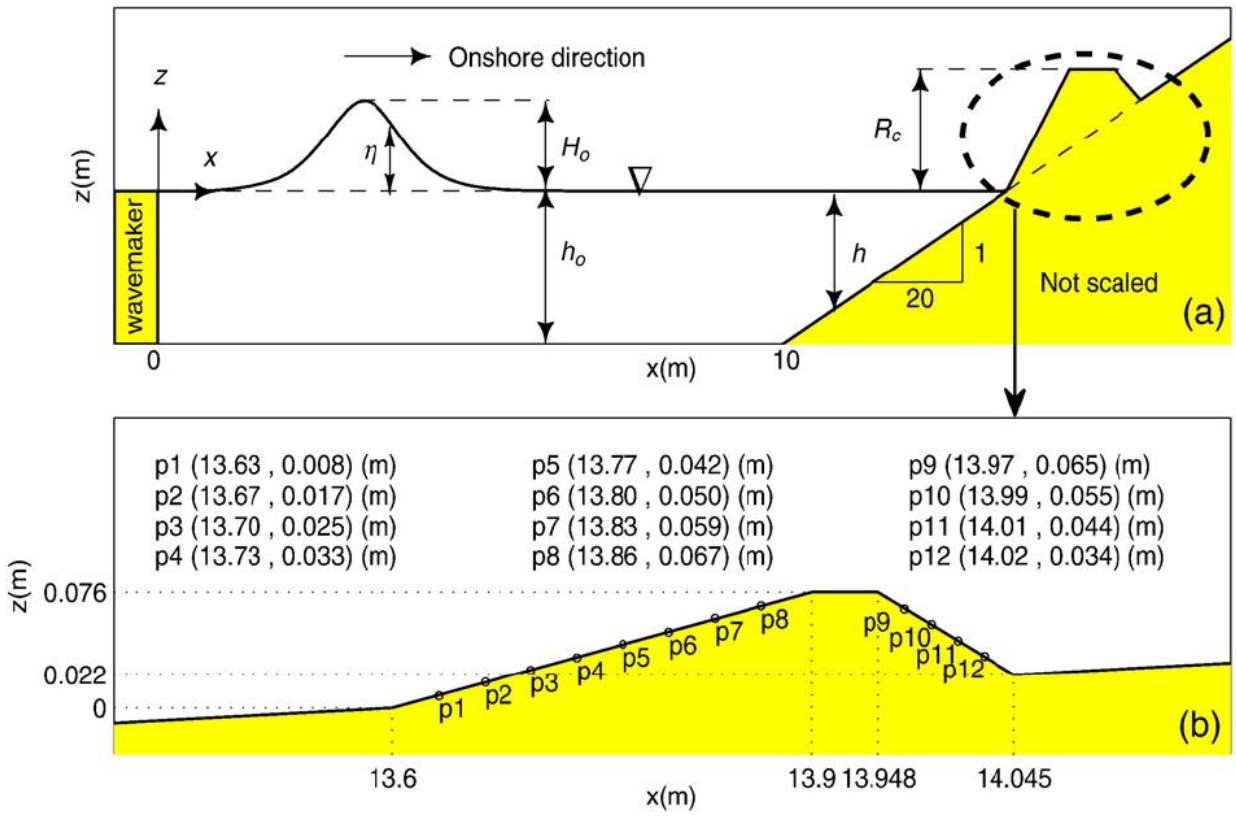

Figure 4-6 Panel (a) Experimental layout; Panel (b) Pressure transducer locations from Hsiao and Lin (2010); (Hsiao and Lin, 2010)

The authors tested the most influential types of wave conditions: a turbulent bore impacting and overtopping the seawall, a wave breaking directly on the seawall and a wave directly overtopping the wall without breaking prior to impact. These are considered the most influential waves because they exhibit similar properties, as well as, behave the closest to a real tsunami wave (Yamamoto *et al.* 2006). The SWL ranges from 0.2m, 0.22m and to 0.256m which allow for the previously mentioned tsunami cases to be investigated.

5. Numerical Model Description

5.1 Computational Domain

Since all three experiments to be investigated are two-dimensional problems, each physical experiment is modelled numerically using the SPHysics model version developed for the two-dimensional case. Details of each numerical setup are provided in the following sub-sections.

5.1.1 Ramsden (1993)

Ramsden's (1993) laboratory experiment is modeled numerically using the layout shown in Figure 5-1. The green vertical line represents the piston-wave paddle while the red vertical line represents the wall that was impacted by the broken wave. Identical to the laboratory experiment, the length of the flume in the computational domain is 23.77m and the height of the wall is 1.0m. Analogous to the physical experiment, the initial water depth [h_o] is chosen as 0.47m in order to maintain the water depth [h_w] near the wall at approximately 5mm. Utilizing Goring's (1978) mathematical model for a solitary wave, the wavemaker was programmed to generate solitary waves in the numerical model.

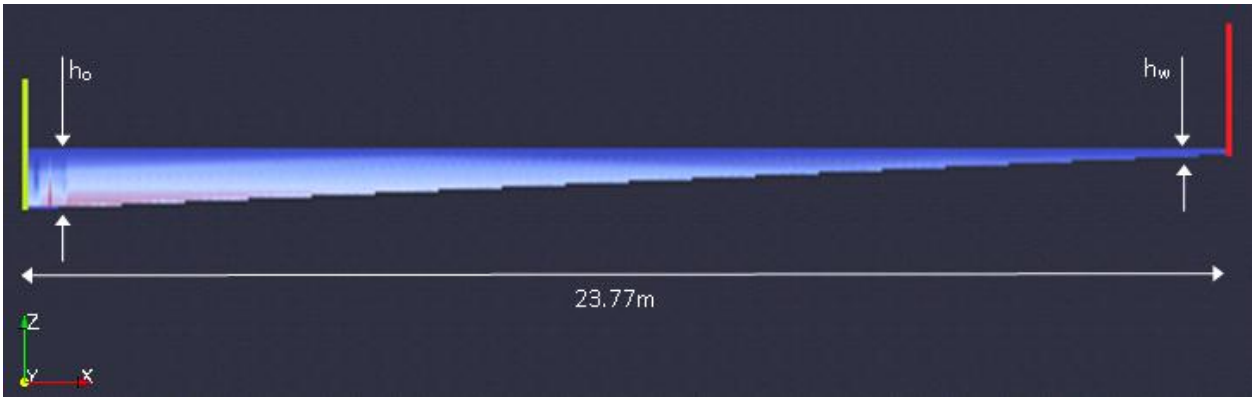

Figure 5-1 Numerical domain setup for Ramsden (1993)

5.1.2 Esteban *et al.* (2008)

The location of the caisson in the numerical domain for the physical experiments of Esteban *et al.* (2008) is shown in Figure 5-2. The dimensions in the computational domain are the same as the ones in the physical experiment. The inundation depth [H1] varies depending on the case to be modelled and was taken to be 17cm for case T2. Throughout all trials, the SWL [H2] was kept at 15cm. The dam break wave is produced by releasing the water in the impoundment reservoir, allowing it to propagate down the flume and up against the caisson shown in red in Figure 5-2. The entire domain length is set at 10.5m and the height is 0.75m. The computational layout carries minor differences from the original physical experiment, which can be found just before the front of the caisson. This is where the 10cm flat section in the original case was simplified into merely a continuation of the slope. This is assumed reasonable since the approach slope is very mild and the length of the flat area is very small in proportion to the domain.

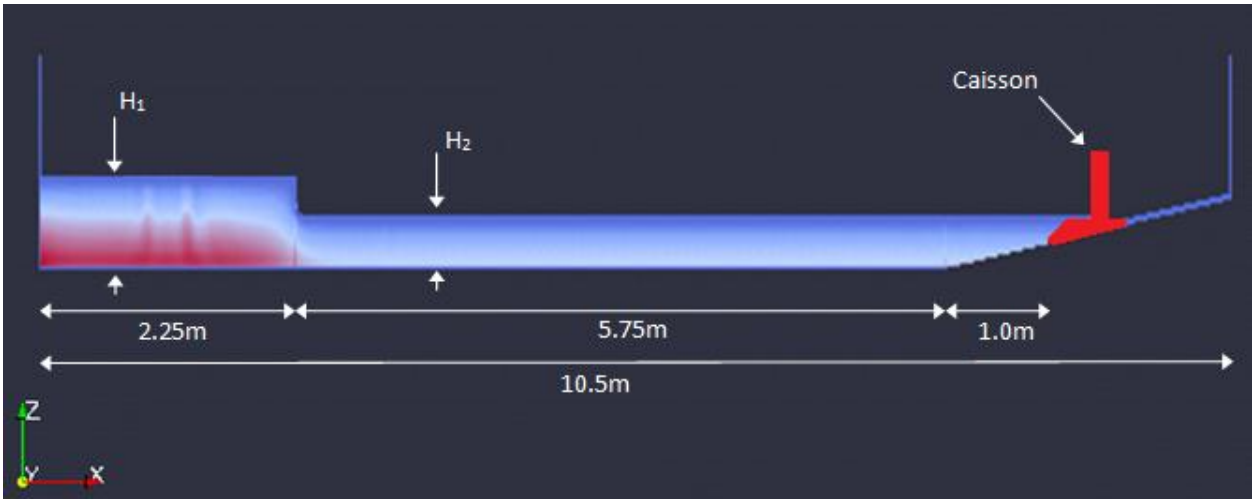

Figure 5-2 Numerical domain setup for Esteban *et al.* (2008)

5.1.3 Hsiao and Lin (2010)

The physical experiments performed by Hsiao and Lin (2010) were also simulated using the SPHysics two-dimensional model. The numerical domain for the tested cases is shown in Figure 5-3. In the figure, the green line on the left represents the piston-type wavemaker and the trapezoidal-shaped seawall is shown in red. The length of the domain is 15.0m and

the height set at 1.0m. Goring's (1978) mathematical model is used to simulate the laboratory estimation of a solitary wave. This means that the input for this experiment included a file describing the positions of the wavemaker and at what velocity it is to move in order to create a certain solitary wave with a specified wave height to water depth ratio (H_o/h_o) in deep conditions. Additional details pertaining to the numerical wave-model will be discussed in section 5.3.

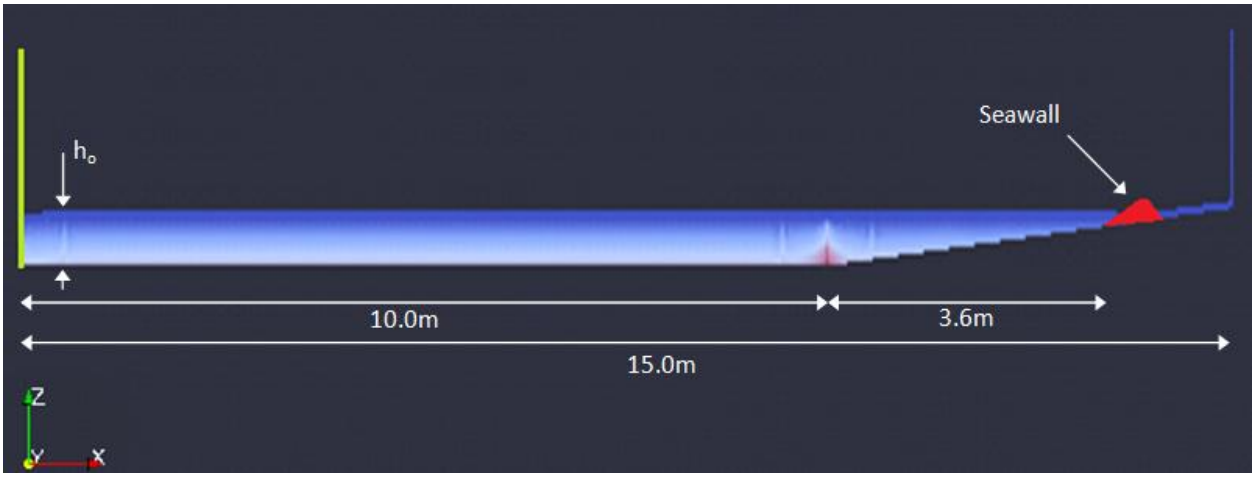

Figure 5-3 Numerical domain setup for Hsiao and Lin (2010)

5.2 Determination of Pressure and Forces

Since the SPHysics code does not allow the direct extraction of the results from the output files, a numerical procedure was implemented in order to compare the numerical results to the experimental ones. This procedure involves calculating the forces exerted on the wall from the direct outputs of the SPHysics code. This chosen method was adopted based on the code developed throughout Philippe St-Germain's Master's thesis (2012). St-Germain's code (St-Germain, 2012) was re-developed for the two-dimensional case of SPHysics. A detailed set of codes is provided in the Appendix of this thesis. As water particles approach the wall, the model will calculate the time-history of the pressure for each particle at each time step. If a sample area in the x-z plane with known dimensions Δx by Δz is considered, then the arithmetic average of the pressures of all the particles in that region can be calculated. The layout of the sampling areas is shown in Figure 5-4. It is then assumed that

the force contribution (per meter width of wall), generated by the water particles inside a sampling area, is obtained as the product of their average pressure and the corresponding tributary height (Δz) of this sampling area on the wall.

The average force acting on the wall along stream-wise direction [F_x] can be calculated using equations (5-1) and (5-2) below where the average pressure of the particles in the sampling area [dA] is given by $\bar{P}$.

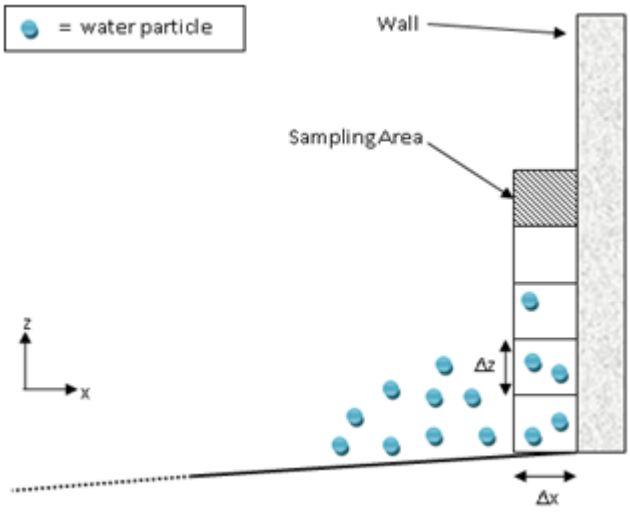

Figure 5-4 Sampling volumes used for force extraction on vertical wall

$$F_x = \int_A \bar{P}\, dA \qquad \textit{(5-1)}$$

$$dA = \Delta x\, \Delta z \qquad \textit{(5-2)}$$

For the case of a non-vertical (sloping) structure, such as in the case of Hsiao and Lin (2010), the force extraction code was modified to take into account the slope of the structure and was implemented to calculate the forces accordingly. Figure 5-5 shows a schematic of this procedure. The code modification involves describing the coordinates of the particles in terms of an imaginary rotated axis by using transformation equations. This imaginary axis lies along the slope of the front face of the seawall and perpendicular to it, with the origin at the seaward bottom corner of the structure. A simple schematic describing the axis is shown in Figure 5-6.

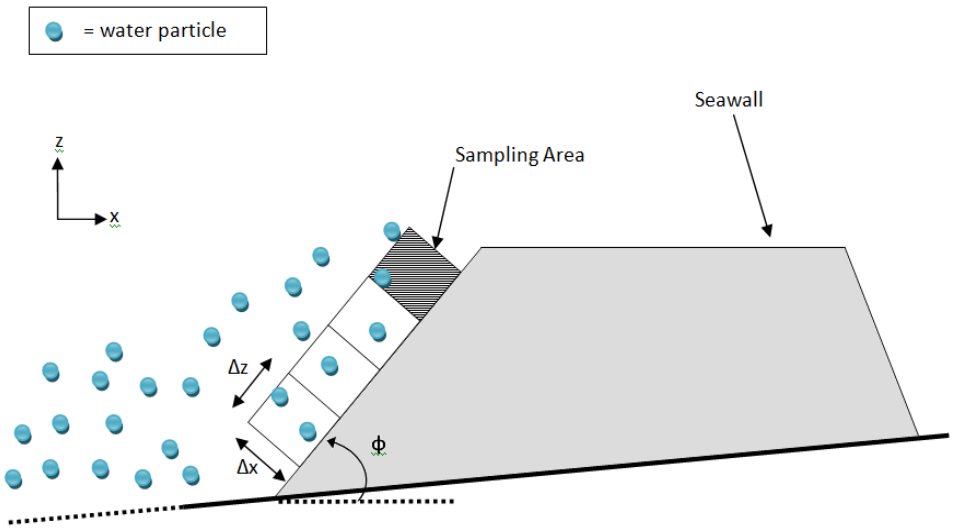

Figure 5-5 Sampling volumes used for force extraction on sloping structure

This simple transformation allows for the same force extraction code to be used without any major modifications. This method was tested by considering a still water sample case on the same structure and comparing the theoretical values of pressure to those calculated using the modified code

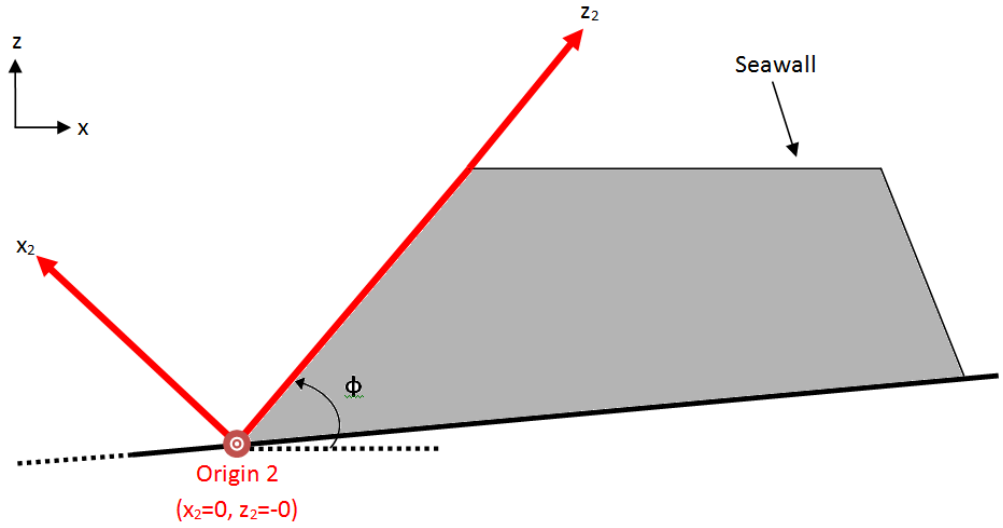

Figure 5-6 Schematic of location of rotated axis

The calculations are based on the equation for static fluid pressure on a submerged surface and the relation is given in equation (5-3). The pressure at a certain point on a flat submerged surface is independent of the angle the surface makes with the horizontal and

only depends on the specific density of the fluid [γ], as well as the length of the fluid column above that point [d].

$$p = \gamma d \tag{5-3}$$

The 2-D transformation of the axes is based on two mathematical equations (5-4) and (5-5):

$$x_2 = -(x - x_o)\, sin\, \varphi + (z - z_o)\, cos\, \varphi \tag{5-4}$$

$$z_2 = (x - x_o)\, cos\, \varphi + (z - z_o)\, sin\, \varphi \tag{5-5}$$

where variables x_2 and z_2 are the coordinates of the particle in the rotated axis, x and z are the coordinates of the particle in the real axis, x_o and z_o are the coordinates of the origin of the rotated axis with respect to the real axis and ϕ is the angle that the front face of the seawall makes with the horizontal. After the coordinates are converted, the rest of the extraction code will continue to calculate the average pressures per sampling area as per equation (5-1).

When using the above explained method of pressure and force calculation, a sensitivity analysis was carried out to select the appropriate location of these sampling areas. This is because in the SPHysics model, it has been observed that particles spacing with respect to the structure can generate discrepancies in calculating the pressure in the region closest to structure. An example of this is shown in Figure 5-7 where the layer of particles closest to the structure unrealistically exhibits very low pressures. This means that the sampling areas should not be taken at zero distance from the structure. The simplest method to obtain accurate pressures is to initiate calculations at the zero-distance location and incrementally and iteratively move the sampling volume away from the structure. The optimal distance away from the structure can be thus estimated.

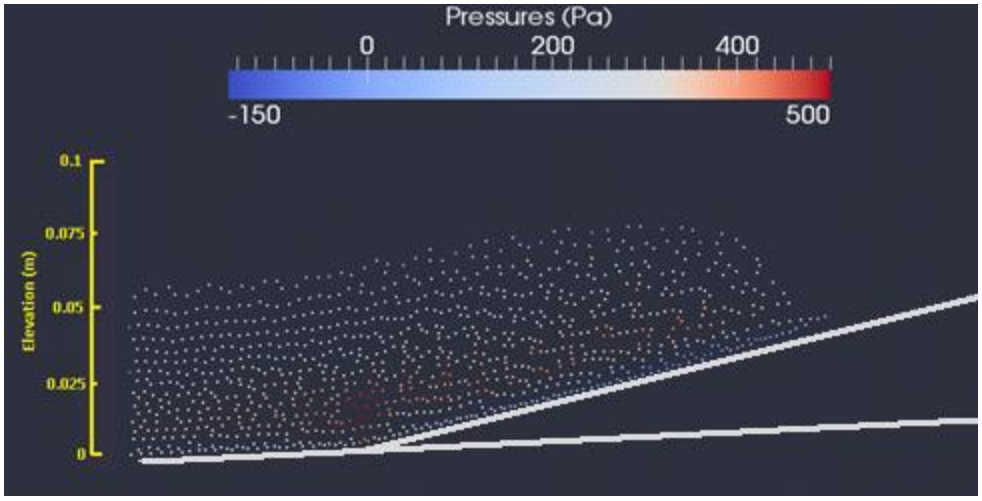

Figure 5-7 Pressure distribution of bore advancing over seawall from SPH model of Hsiao and Lin (2010)

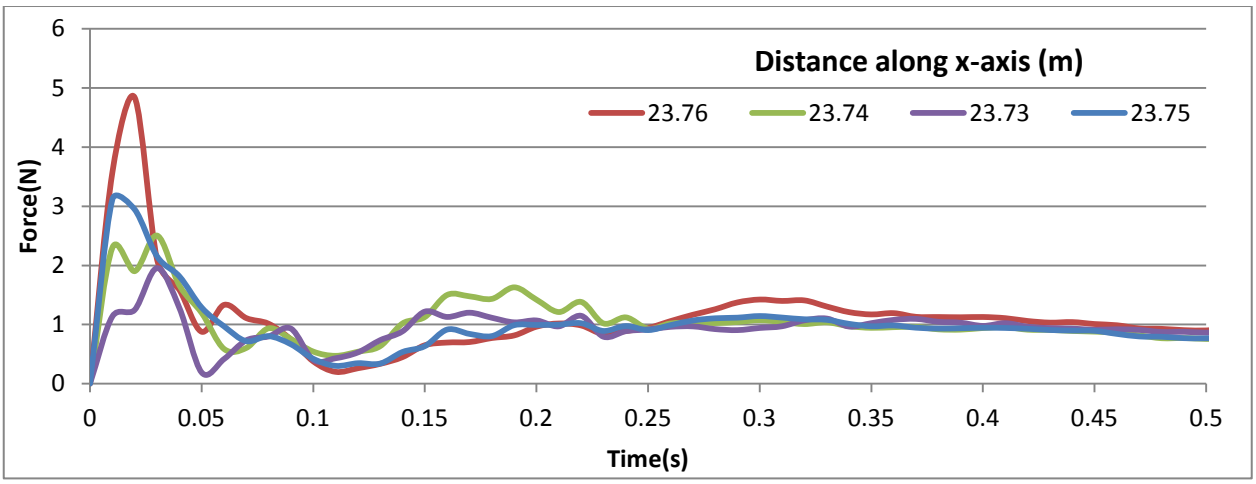

Figure 5-8 Variation of maximum force as the centre of the sampling area is moved away from the wall

An example of this process is provided here by taking samples from a test simulation of Ramsden's (1993) layout where the wall is located at x=23.77m from the origin. The sampling area's x-centre was moved by 0.01m away from the wall such that the sampling columns are located at 23.76m, 23.75m, 23.74m and 23.73m, as indicated in Figure 5-8 above. Force-time histories for the same set of data are shown to decrease as the sampling column moves further away from the wall. From Figure 5-8, it can be concluded that the actual maximum force values can be most accurately sampled at the distance 23.76m.

5.3 Solitary Wave Model

Goring (1978) proposed a mathematical model for the purpose of laboratory solitary wave generation. The surface profile $[\eta(x,t)]$ of a solitary wave can be described using the following equation:

$$\eta(x,t) = Hsech^2[\kappa(Ct - X_o)] \quad \textit{(5-6)}$$

$$C = \sqrt{g(h+H)} \quad \textit{(5-7)}$$

$$\kappa = \sqrt{\frac{3H}{4h^3}} \quad \textit{(5-8)}$$

where C is the wave celerity and X_o is the wave board displacement. Applying equation (5-6) for a piston-type wavemaker one can generate a new function relating the wave-board displacement to the properties of the wave as shown in equation (5-9).

$$X_o(t) = \frac{H}{\kappa h} tanh[\kappa(Ct - X_o)] \quad \textit{(5-9)}$$

Through an iterative process, such as Newton's rule, and solving equation (5-9), the position of the wave-board at specified time intervals can be predetermined. The stroke of the wave board is the distance that the board has to travel to generate the required wave. Its value can be estimated using equation (5-10). Also, the time required by the board to complete the stroke, or the stroke duration $[t_f]$, can be approximated using the relation in equation (5-11).

$$S = \sqrt{\frac{16Hh}{3}} \quad \textit{(5-10)}$$

$$t_f = \frac{2(3.80 + \frac{H}{h})}{\kappa C} \quad \textit{(5-11)}$$

Using the above sequence of equations to create an input file containing the time, board displacement and board velocity, the SPHysics model is able to further use this file to prescribe the motion of the piston-type wavemaker in the model. Results of this iteration

process for Ramsden (1993) and Hsiao and Lin's (2010) SPH model simulations will be discussed in Chapter 6.

5.4 Sensitivity Analysis

The parameters selected for each simulation were selected based on a parameter sensitivity analysis as well as recommendations from previous research by Didier and Neves (2010) and Gomez-Geistera *et al.* (2010a,b). In this study, the sensitivity analysis focuses mainly on how the smoothing length, beta-limiter, the speed of sound and particle spacing affect the force-time history and simulation time of a wave-structure impact. The analysis was conducted using the setup by Hsiao and Lin (2010) since it is considered the more complicated domain profile. The first parameter tested was the smoothing length [h] which would affect the kernel approximations of the model: results are displayed in Figure 5-9. There is a drop in the force values at around 1.2s which indicates instability in the simulation of the force values. The graph illustrates how decreasing the smoothing length increases the stability of the force calculations. Additionally, it was determined that the smoothing length did not significantly affect the time required to run the simulations.

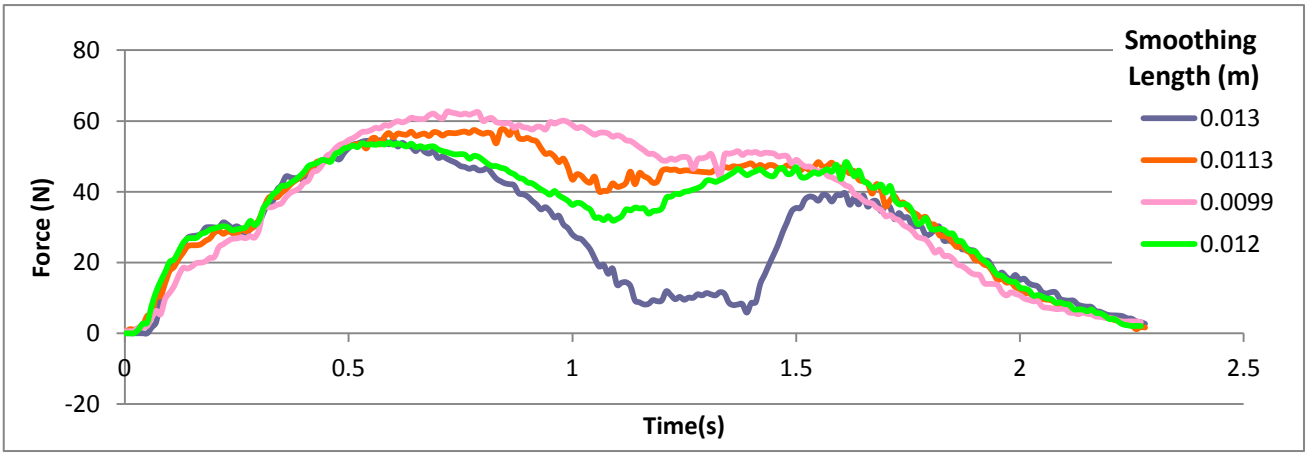

Figure 5-9 Comparison of force calculations using different smoothing length values

As already discussed, the speed of sound is artificially modelled to be ten times the maximum velocity of the wave in the simulation. In SPHysics, the speed of sound is varied through the use of a non-dimensional coefficient [ς_{sound}] ranging from 10 to 40. In this analysis, the speed of sound was set to vary between 15.41m/s and 28.01m/s. As shown in

Figure 5-10, as the speed of sound is increased, the maximum impact forces, occurring around 0.53s, increase as well. However, when comparing the lowest value of speed of sound to the highest, this is a very small change of only about 5%. After the initial impact, calculated forces are higher for lower values of speed of sound. It should be noted that the drop in the force profiles is reduced due to the selection of a more appropriate value of $[\varsigma_{sound}]$ and [h]. Increasing the value of $[\varsigma_{sound}]$ was shown to slightly increase the computational time.

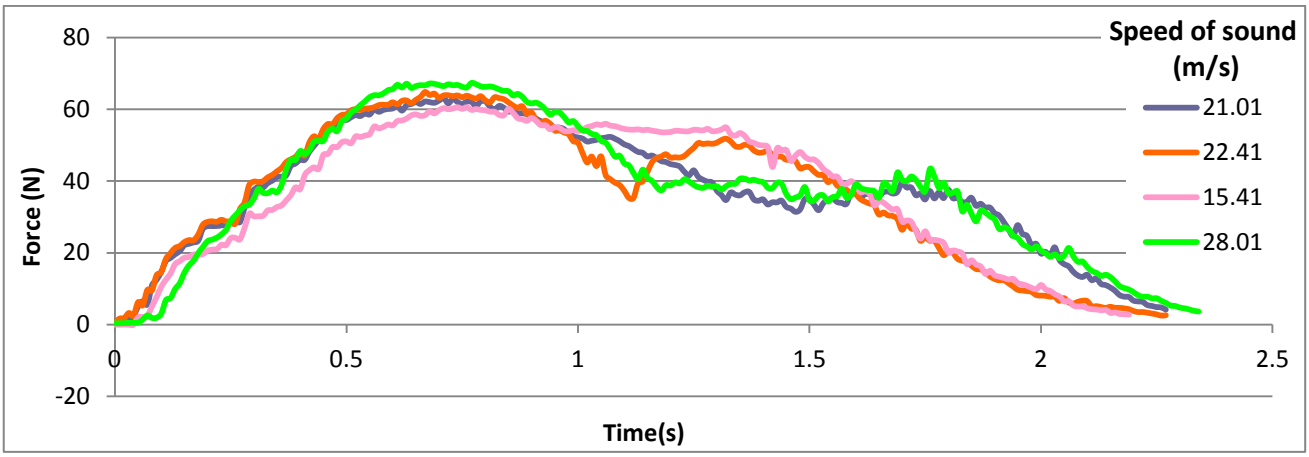

Figure 5-10 Comparison of force calculations using different speed of sound values

For the case of using a Riemann solver, with the purpose of minimizing pressure - and implicitly force – fluctuations, varying the β-limiter constant on the force-time history is shown in Figure 5-11. It can be seen that changing the β-limiter does not provide a clear correlation in terms of its influence on the calculation of the force-time history in this case; however, there are some variations of up to 15% which are caused by the slight perturbations observed in the water surface. This is due to the reasons discussed in section 3.3.8. Again, the drop in the force values at approximately 1.2s is attributed to a lack of stability in the computation. In terms of the computational time, the β-limiter was not observed to have any effect.

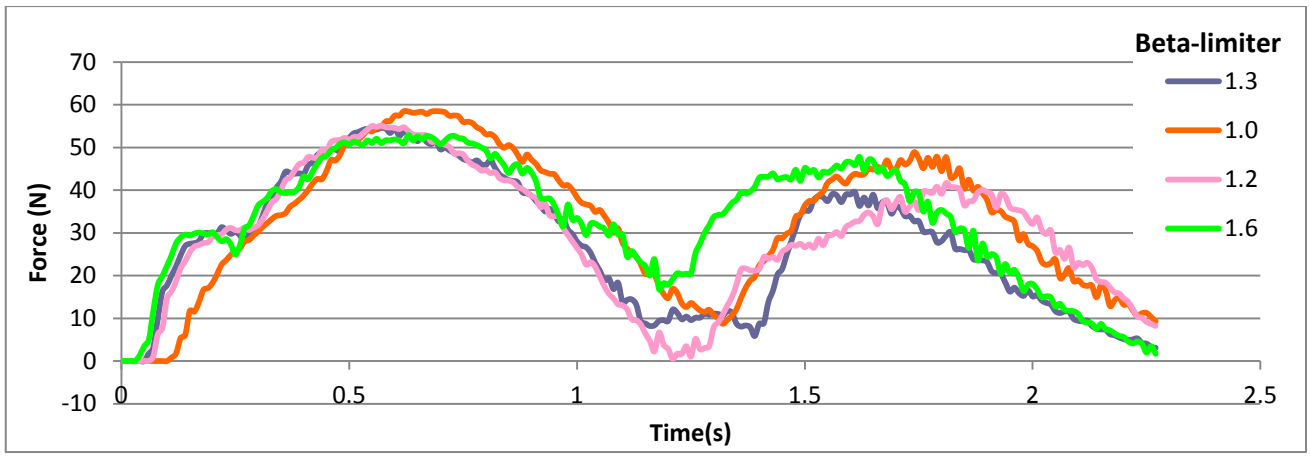

Figure 5-11 Comparison of force calculations using different Beta-limiter values

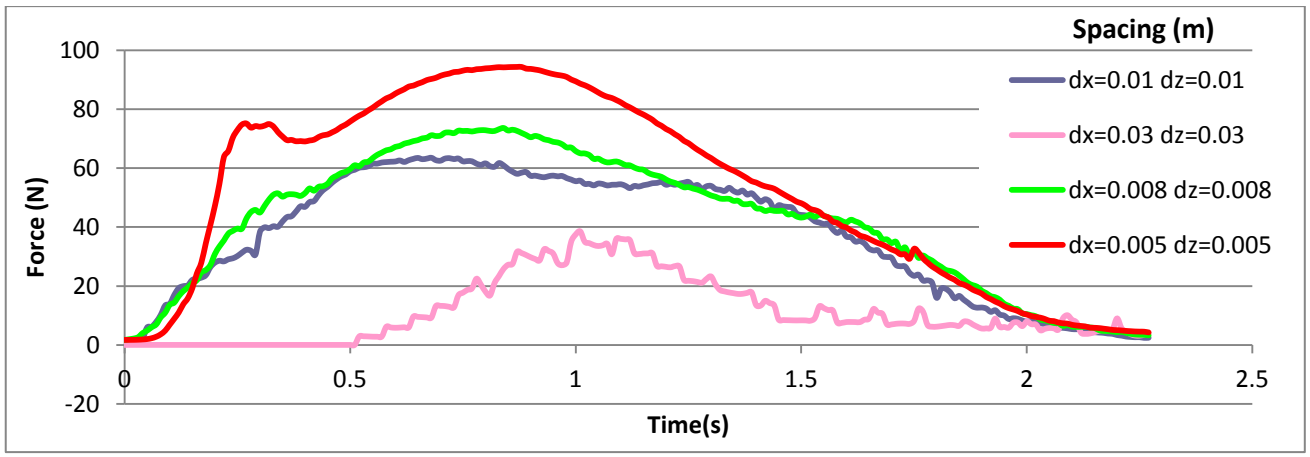

Figure 5-12 Comparison of force calculations using different varying spacing values

The inter-particle spacing is also varied and the results are shown in Figure 5-12. As the particle spacing is decreased, the instabilities in the estimation of the time-history of forces are eliminated. This is due to the higher resolution achieved such that the interpolation of particle properties in the calculation becomes accurate and physically realistic in term of the fluid behaviour. As indicated below in Table 5-1, the inter-particle spacing was observed to significantly affect the computational time. Additional simulation details for this sensitivity analysis can be found in Appendix B.

Table 5-1 Computational times for various inter-particle spacing

Particle Spacing [dx and dz, respectively] (m)	Computational Time (hours)
0.01, 0.01	4.67
0.008, 0.008	3.50
0.03, 0.03	0.17
0.005, 0.005	35.0

5.5 Parameter Selection

In this section, the most significant parameters used to model each physical experiment are discussed. In general, when using an SPH model, the main parameters to be specified are: the smoothing distance, the kernel equation, whether or not a density filter is to be used, the particle spacing, whether or not a Riemann solver and the corresponding values of constants is used, and the speed of sound and the time stepping method.

5.5.1 Ramsden (1993)

The majority of the parameters used were chosen with reference to recommendations of similar experiments and based on a trial and error method deduced from the sensitivity analysis conducted in section 5.4. For Ramsden's (1993) model simulation, the final parameter values that provided the best matching results, between the numerical and the expected physical results, are listed below in Table 5-2. In order to compare the values of the numerical models, two sets of parameters for the Ramsden (1993) model were compared; one with 'low' resolution due to a particle spacing [dx by dz] of 0.03m by 0.01m and one with 'high' resolution due to a particle spacing of 0.008m by 0.008m. Details of the low resolution parameters can be found in Appendix B.

Table 5-2 Numerical model parameters selected for of Ramsden's (1993) experiment

Parameter	Value
Particle Spacing [dx and dz] (m)	0.008, 0.008
Smoothing Distance [h] (m)	0.104

Smoothing Distance Coefficient	0.92
Kernel Type	Cubic
Density Filter	None
Riemann Solver	Non-conservative (Parshikov, 2000)
Riemann Solver`s Slope Limiter [Beta-limiter]	1.2
Reference Speed of Sound (m/s)	32.21
Coefficient of Speed of Sound $[\varsigma_{sound}]$	15
Time Stepping Method	Symplectic
Viscosity Treatment	Laminar + Sub-Particle Scale

5.5.2 Esteban *et al.* (2008)

Similar to what was discussed in section 5.5.1, the parameters for the Esteban *et al.* (2008) experiments are shown in Table 5-3 below. The fundamental difference comparing to the parameters used in Ramsden's model is that the domain is much shorter and hence a smaller number of particles was used. Additionally, there is no wavemaker and the hydraulic bore is produced using the dam-break method. The majority of the parameters are similar to those selected for the Ramsden (1993) numerical experiment.

Table 5-3 Numerical model parameters selected for the experiment of Esteban *et al.* (2008)

Parameter	Value
Particle Spacing [dx and dz] (m)	0.005, 0.005
Smoothing Distance [h] (m)	0.104
Smoothing Distance Coefficient	0.92
Kernel Type	Cubic
Density Filter	None
Riemann Solver	Non-conservative (Parshikov, 2000)
Riemann Solver`s Slope Limiter [Beta-limiter]	1.2
Reference Speed of Sound (m/s)	26.58
Coefficient of Speed of Sound $[\varsigma_{sound}]$	15
Time Stepping Method	Symplectic
Viscosity Treatment	Laminar + Sub-Particle Scale

5.5.3 Hsiao and Lin (2010)

For Hsiao and Lin's (2010) experiment, inter-particle spacing was chosen to be 0.005m since the required wave breaking was judged most physically and qualitatively well reproduced at this resolution. Additionally, the slope limiter for the Riemann Solver was chosen to be 1.3. A second SPH simulation was run involving a shorter domain in order to reduce the computational time. This was achieved by reducing the length of the flat section before the slope by 3.0m. A similar type of investigation was done by Hsiao and Lin (2010) when they used a shortened numerical model to compare to their physical experiments.

Table 5-4 Numerical model parameters selected for the experiment of Hsiao and Lin (2010)

Parameter	Value
Particle Spacing [dx and dz] (m)	0.005, 0.005
Smoothing Distance [h] (m)	0.0065
Smoothing Distance Coefficient	0.92
Kernel Type	Cubic
Density Filter	None
Riemann Solver	Non-conservative (Parshikov, 2000)
Riemann Solver`s Slope Limiter [Beta-limiter]	1.3
Reference Speed of Sound (m/s)	21.01
Coefficient of Speed of Sound $[\varsigma_{sound}]$	15
Time Stepping Method	Symplectic
Viscosity Treatment	Laminar + Sub-Particle Scale

5.6 Experimental versus Computational Time

Each experiment involved a different physical modeling time; hence, with varying parameters, each case generates different computational running times. Table 5-5 below summarizes details used for each of the three experimental setups and computational domain and run time.

Table 5-5 Computational domain and run-time parameters used for reproducing the physical experiments

Experiment	No. of Particles	No. of Boundary Particles	Data Collection Frequency (Hz)	Physical Model Time (s)	Computational Time (days)
Ramsden (1993) Low Res.	39398	999	100	18	1.92
Ramsden (1993) High Res.	181920	3229	100	15	15.3
Esteban *et al.* (2008)	148409	2790	20	10	11.0
Hsiao and Lin (2010)	194840	3424	100	15	13.9
Hsiao and Lin (2010) Short	138279	2819	100	12	8.13

The number of particles in each simulation is determined by the spacing size and the dimensions of the domain. According to Dalrymple and Rogers (2006), a balance must be obtained when selecting high particle resolutions versus lower ones since the artificial viscosity will become increasingly dissipative as the resolution increases. A dissipative viscosity means the surface of the fluid will become irregular and choppy. As indicated in Table 5-5, the simulation with the longest computational time, a value of 15.3 days, is Ramsden (1993) High Res. This is because the domain is the longest and the particle spacing was set at 0.008m. It is important to note that the shortened domain simulation conducted for Hsiao and Lin (2010) reduced the required computational time from 13.9 days to 8.13 days which is a significant improvement of just under 6 days. The effects of using a shorter domain will be presented in chapter 6 with the results of both numerical cases of Hsiao and Lin (2010).

6. Data Analysis

The results of the numerical simulations that have been conducted are discussed in this section. First, the preliminary numerical results of each experiment are given, followed by a data analysis that includes the values of the maximum forces, the force-time histories at critical locations on the structure as well as the wave propagation results.

6.1 Results

6.1.1 Ramsden (1993)

Wave Generation, Propagation, and Breaking

To confirm that waves were successfully modelled in the simulation, all three stages of the wave evolution - generation, propagation and breaking - must be carefully investigated and results of the numerical model must be compared to the data recorded in the physical experiments. In order to generate a wave in SPHysics 2D, a prescribed wavemaker motion file must be provided as input into the model. As previously mentioned, the mathematical model proposed by Goring (1978) was used to calculate the position of the piston from the given wave parameters in Ramsden's experiments (Ramsden 1993). By following the red solid line in Figure 6-1 below, the basic concept of generating a solitary wave is as follows. The paddle will have a slow start that eventually turns into a steep increase in velocity to a maximum followed by a steep decrease back to zero velocity of the wave paddle. The green dashed line describes the displacement of the wave paddle in the numerical domain. The displacement behaviour that corresponds to the velocity profile is a gradually increasing slope followed by an abrupt jump and finally a plateau at maximum displacement. This pattern is characteristic of any solitary wave profile. For Ramsden (1993), in order to generate an initial wave height ratio [H_o/h_o] of 0.288, the maximum velocity of the paddle is around 0.52m/s and the maximum displacement or total stroke is 0.53m.

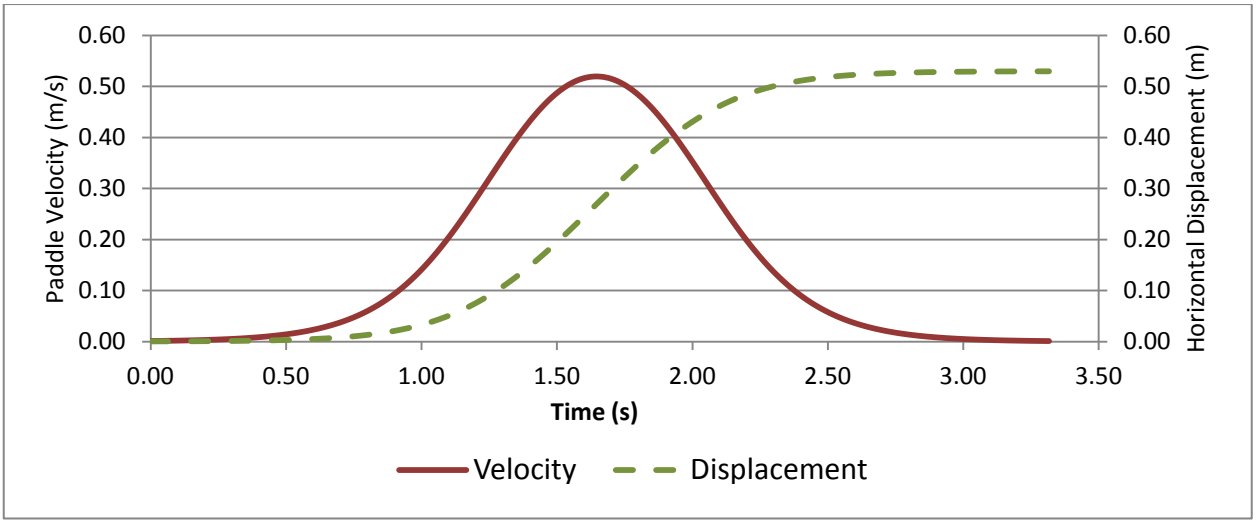

Figure 6-1 Wavemaker paddle velocity and displacement time-histories for the simulation of Ramsden's experiment (1993)

The numerical piston wave generator successfully generated a solitary wave with similar initial conditions as the one used in Ramsden's (1993) experiment. The initial x-position of the piston was set at -0.53m with respect to the origin of the axes. The initial wave height to water depth ratio [H_o/h_o] was 0.290, compared to Ramsden's value of 0.288 (Ramsden 1993). This is considered to be within the admissible range of error of less than 1 percent. A visual result of the generation of the solitary wave is shown in Figure 6-2. The solitary wave, generated by the piston, starts to form at around 2.00s into the simulation. At 3.50s, the wave is fully developed and it then propagates up the mild slope, towards the wall.

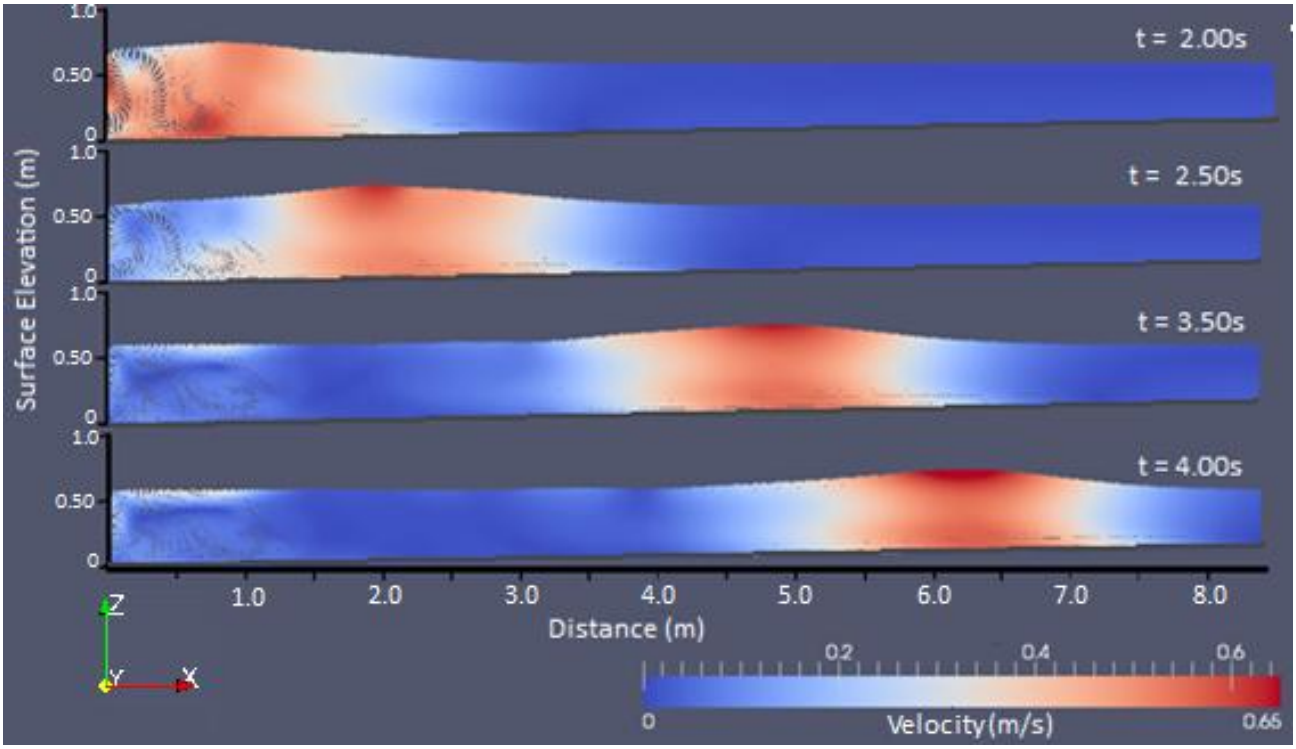

Figure 6-2 Solitary wave generation in SPH simulation

The next stage is the wave breaking on the slope. In the experiment performed by Ramsden (1993), the solitary wave breaks at a distance of 15.6m from the piston. In the numerical model, the wave breaks a bit earlier, at a distance of around 15.0 m. The velocity distribution and evolution of the wave breaking are shown in Figure 6-3. Just before the wave starts to break, at a time 8.0s into the simulation, the breaking wave height ratio $[H_b/h_b]$ is about 1.2. Further on, at time 8.4s, the wave is fully breaking in the form of a plunging breaker, as also observed in Ramsden's experiment. During the flip-through, the top of the plunger spills forward hence the wave is classified as a spiller-type plunging breaker with an Irribaren number of $\xi = 0.15$. In Ramsden's experiment, $[H_b/h_b]$ was found to be 1.21 which is almost identical to the value obtained with the numerical model.

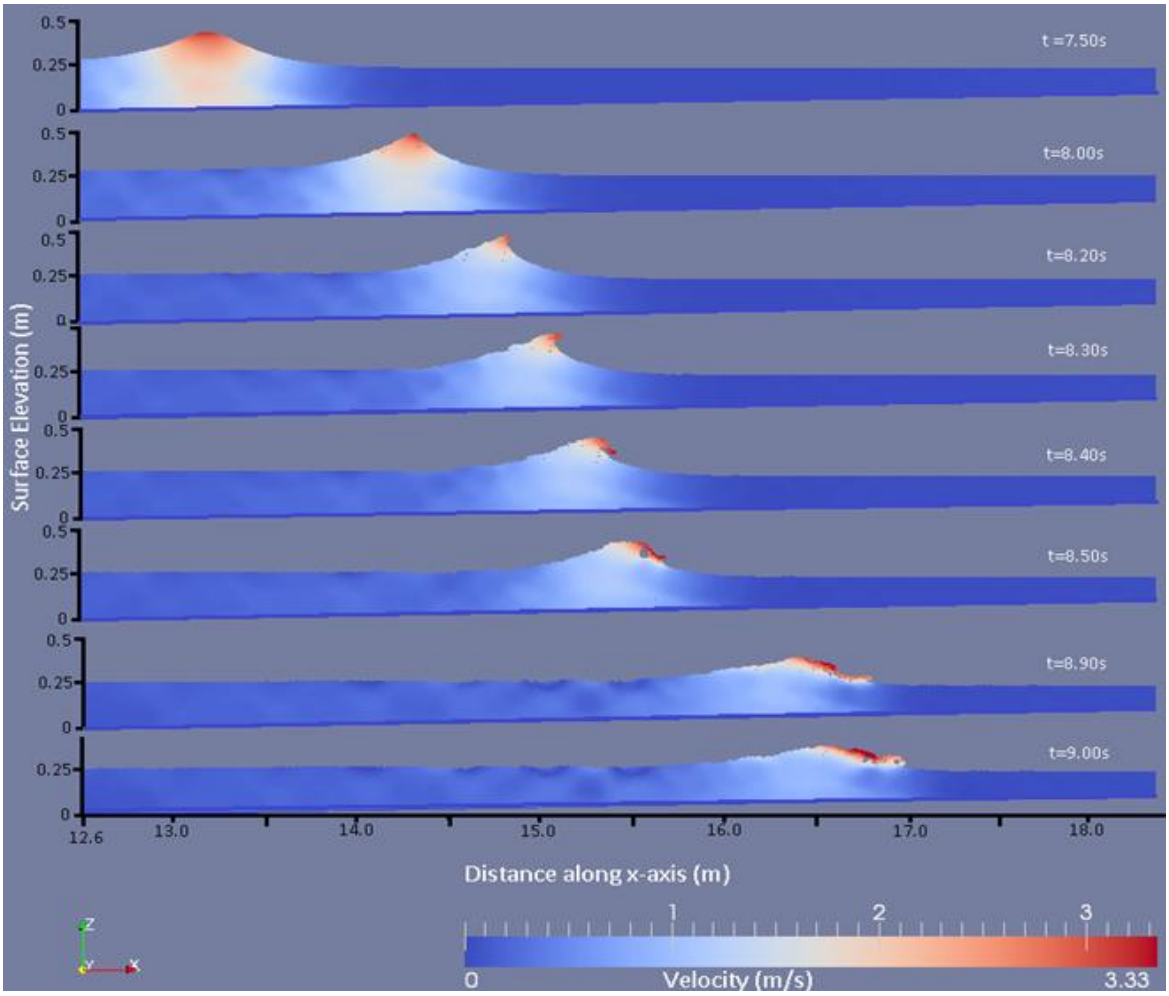

Figure 6-3 Wave propagation and breaking in SPH simulation of Ramsden's (1993) experiment

Comparison of Numerical and Experimental Results

In this section, the results of the numerical model and the experimental results are compared, first by visually/qualitatively comparing the results of the hydraulic bore (broken wave) propagating towards the wall just before and following impact. Secondly, the magnitudes (experimental and numerical) of the forces exerted on the wall due to the bore impact are compared. In Figures 6-4 and 6-5, panel (a) represents the hydraulic bore (broken wave) profile from the numerical model with low resolution, panel (b) represents that from the numerical model with high resolution and panel (c) represents the experimentally-measured bore profile of Ramsden (1993). The general shape of the bore as it approaches the wall in Figure 6-4(a) shows that the lower resolution simulation does not

contain enough particles to accurately model the physical bore profile shown in Figure 6-4(c). Also, the approaching depth is around 3.5cm, at a distance 0.4m from the wall, which is lower than the value of 5.0cm used by Ramsden (1993) in his experiment. There is much better agreement between the high resolution numerical simulation results and the ones of the physical experiment as shown in Figure 6-4(b). The surface elevation is around 5.0cm, at 0.4m from the wall, and the hydraulic bore starts to decrease in depth towards zero at around 9.0cm away from the wall. The overall geometry of the numerically-generated hydraulic bore in Figure 6-4(b), as it approaches the wall, is similar to that obtained in the experiment, as shown in Figure 6-4(c).

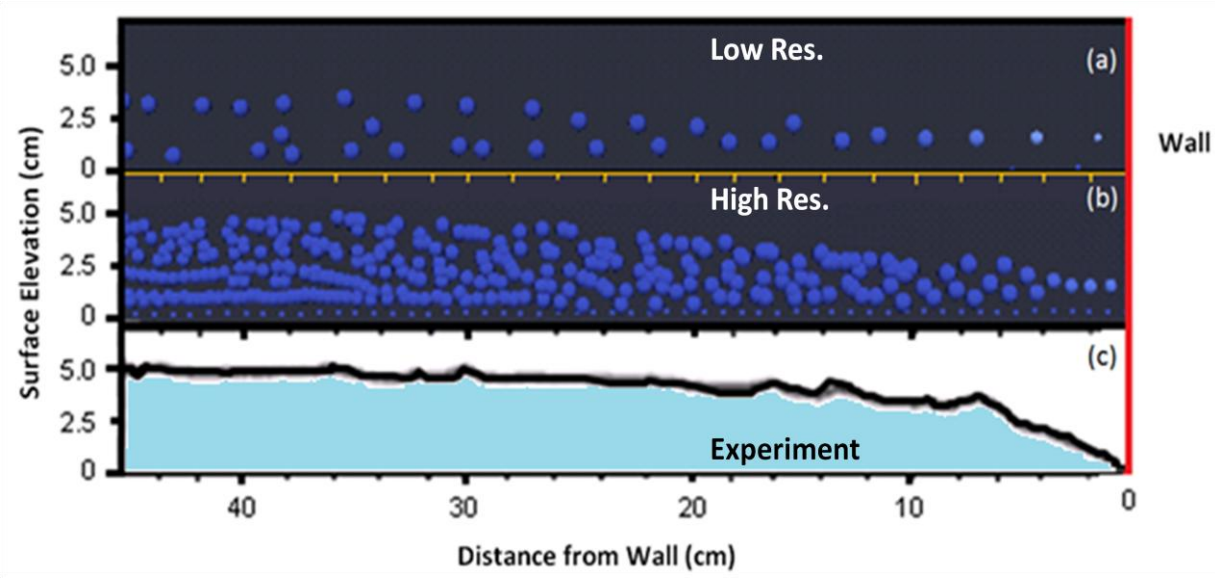

Figure 6-4 Comparison of bore profiles just before wall impact ((c) panel adapted from Ramsden's experiment, (1993)

Next, comparing Figure 6-5(a) to (b) and (c), the height of the bore at around 4.0m from the wall is approximately 5.0cm for all three experiments indicating good agreement. After the bore first impacts the wall, the initial runup height is approximately 6.0cm in Ramsden (1993) and 8.0cm in the low resolution SPH simulation results. This difference of 33 percent is not considered accurate and may be due to presence of shallow water near the wall. This can be explained by the fact that the numerical wave broke later down the flume in the low-resolution numerical model comparing to the wave breaking in Ramsden's experiment, such that the bore had less time to dissipate and hence impacted the wall with a larger

force. This also means that bore celerity was higher, causing particles to move further upward on the wall surface. Figure 6-5(b) shows that that the bore profile in the high resolution simulation accurately mimics the shape of the physical experiment starting with the surface elevation of approximately 5.0cm at a distance of 4.0m away from the wall, the dip located at around 3.0cm from the wall and finally the runup itself of approximately 8.0cm. These values are in agreement with the values obtained by Ramsden (1993) where the surface elevation was around 5.0cm and the runup was about 7.0cm.

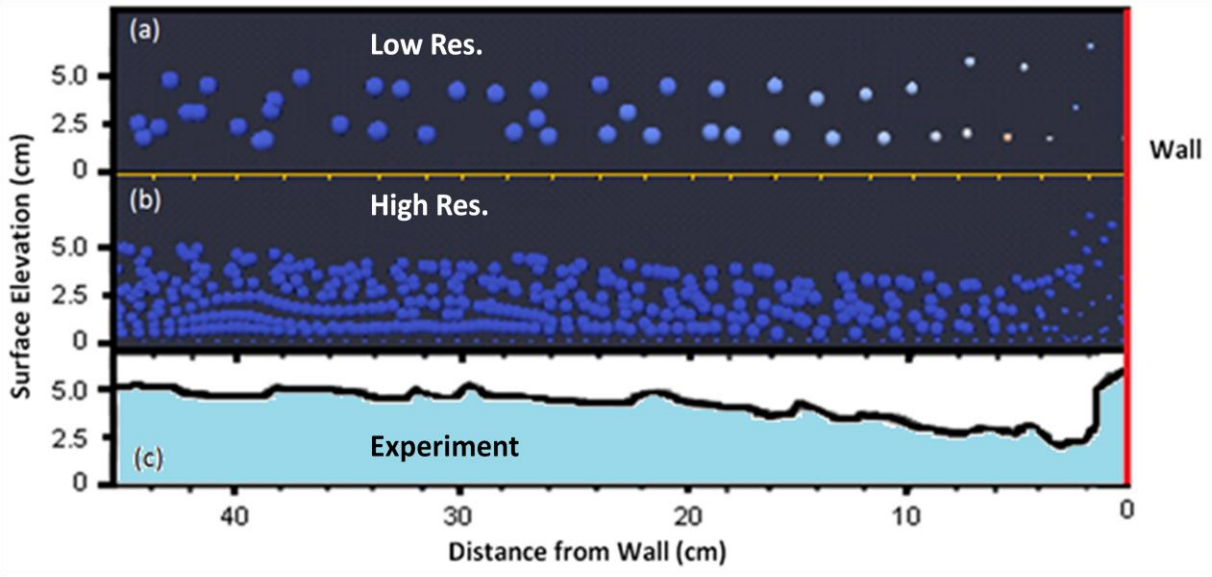

Figure 6-5 Comparison of initial runup profiles on wall ((c) panel adapted from Ramsden, 1993)

As shown in Figure 6-6, the red line represents the total force measurements recorded by Ramsden (1993) in the experiment while the blue and green lines represent the total averaged force calculated by the numerical model using low and high resolution, respectively. The force magnitude from the physical experiment shows a quick initial increase up to approximately 4.5N, followed by a smaller peak. As the wave pulls away and is reflected off the wall, a gradual decrease in the force occurs until the force reaches close to zero. As shown by the low resolution force profile, the force spikes up to a maximum value of approximately 12N followed by a gradual increase as the runup occurs on the wall. The force then reaches a second peak of approximately 4.8N and gradually decreases further to zero.

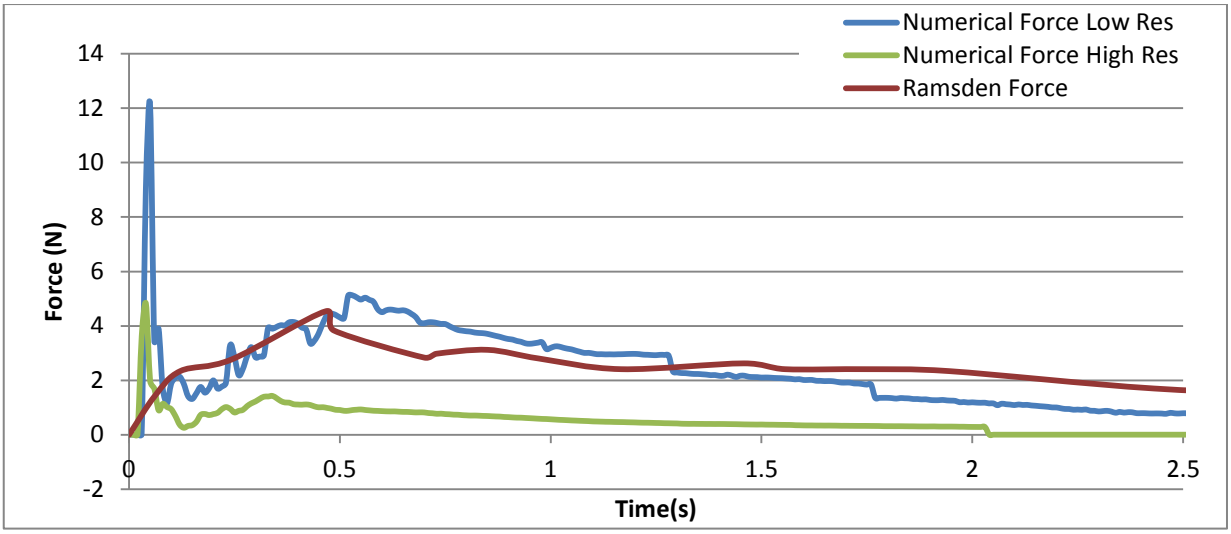

Figure 6-6 Comparison numerical results and experimental data for the total force on the wall for Ramsden (1993)

At closer inspection of Figure 6-6, shows that the peak force found in the low resolution numerical results was not recorded in the experimental results. As indicated in Ramsden's discussion (1993), the rise time for the impulse force was about 11 percent of the wall's natural frequency. This meant that the force transducers that Ramsden used in the experiment could not record this initial spike in the force (Ramsden 1993). At approximately the instant of 0.4s from initial impact and further on, forces from the low resolution model and the experiment are similar.

The magnitudes of the average force are different in the case of the high resolution numerical experiment. As shown in Figure 6-6, the average force in the high resolution experiment reaches a peak of 4.6N and then drops, forming a secondary peak of approximately 0.8N. Other than the inter-particle spacing, the difference in the parameters for the low and high resolution runs is the β-limiter value of 1.15 and 1.2 respectively. Changing these parameters caused a significant decrease in the magnitude of forces exerted on the wall. This is a counter-intuitive result that suggests that although the qualitative results of the numerical experiments are shown to be more accurate for the higher resolution (refer to Figures 6-4 and 6-5), the quantitative results of the impact forces

are not. Further analysis will be required in order to pinpoint the cause of this phenomenon.

6.1.2 Esteban *et al.* (2008)

Bore Advancement and Impact

As previously stated, only the second case (T2) for Esteban *et al.* (2008) was considered where the SWL is 17cm and the dam-break wave broke on the caisson. A closer look at the dam-break wave approaching the caisson is shown in Figures 6-7 and 6-8 where the top panels show photos from the physical experiment of Esteban *et al.* (2008) while the bottom panels show results from the SPH numerical model. Initially, in panel (a), the dam-break wave in the numerical experiment has the same shape as the one recorded in the physical experiment. As it will be shown later, the incident wave gauge readings from the physical model and the ones calculated in the numerical model are similar. In panel (b), at 4.25s, there is a dip in the surface water level, located however closer to the wall in the numerical model results. Continuing with panel (c), the wave starts to run up the front of the caisson creating a turbulent splash up. The splash is not observed in the numerical model and the surface water level remains lower than the level recorded in the physical experiment. Finally, at around 4.95s, as shown in panel (d), as the wave is reflected, the bore flows back down the flume. One can notice significant turbulence in the water surface of the numerical model when compared to the wave peak that was recorded in the physical model.

As the wave approaches the caisson, the maximum wave velocities are observed to be near the surface and at the front of the wave crest. This is shown by in Figures 6-7 and 6-8 where blue represents the lowest velocity while red is the highest. When the wave is reflected off the caisson, the velocity dissipates due to the collision and the reflected wave maintains a maximum velocity of approximately 1.3m/s.

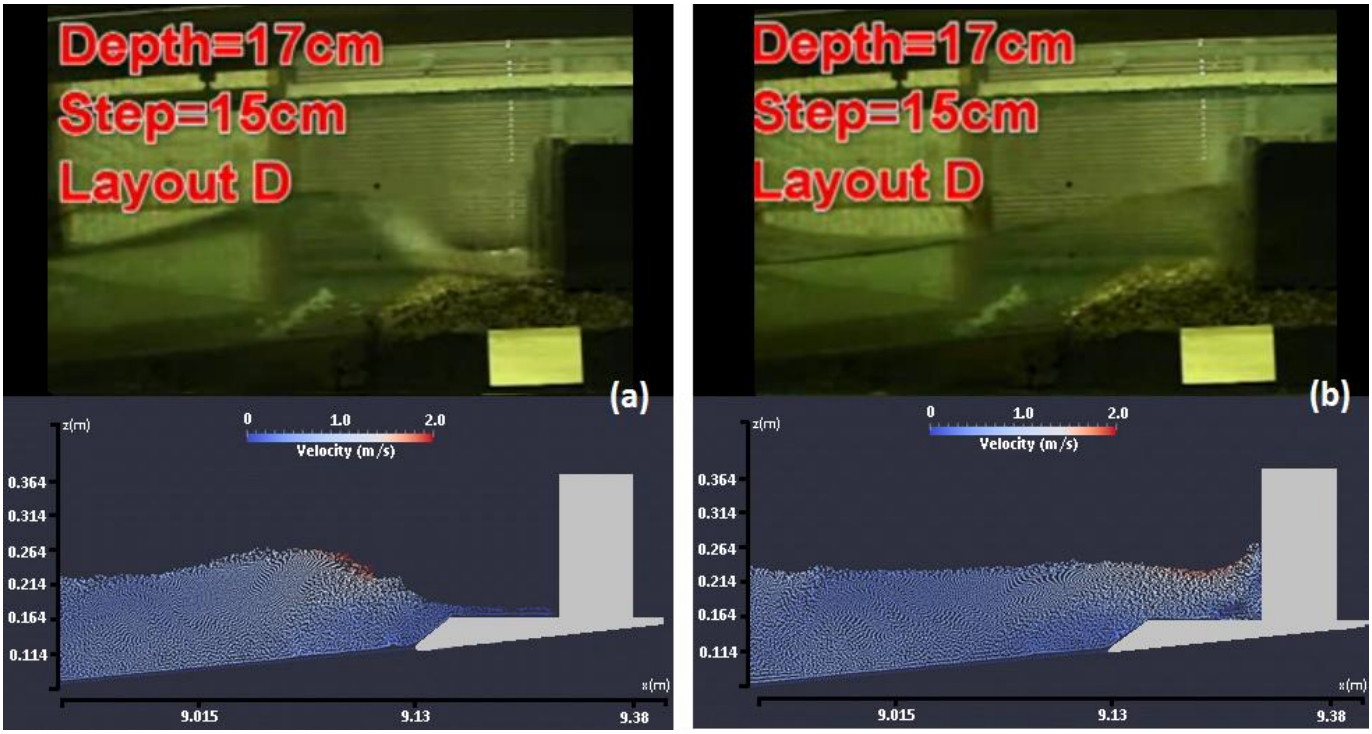

Figure 6-7 Approaching bore profiles from Esteban *et al.* (2008) (top) and numerical model (bottom) at simulation times (a) t = 4.05s (b) t = 4.25s

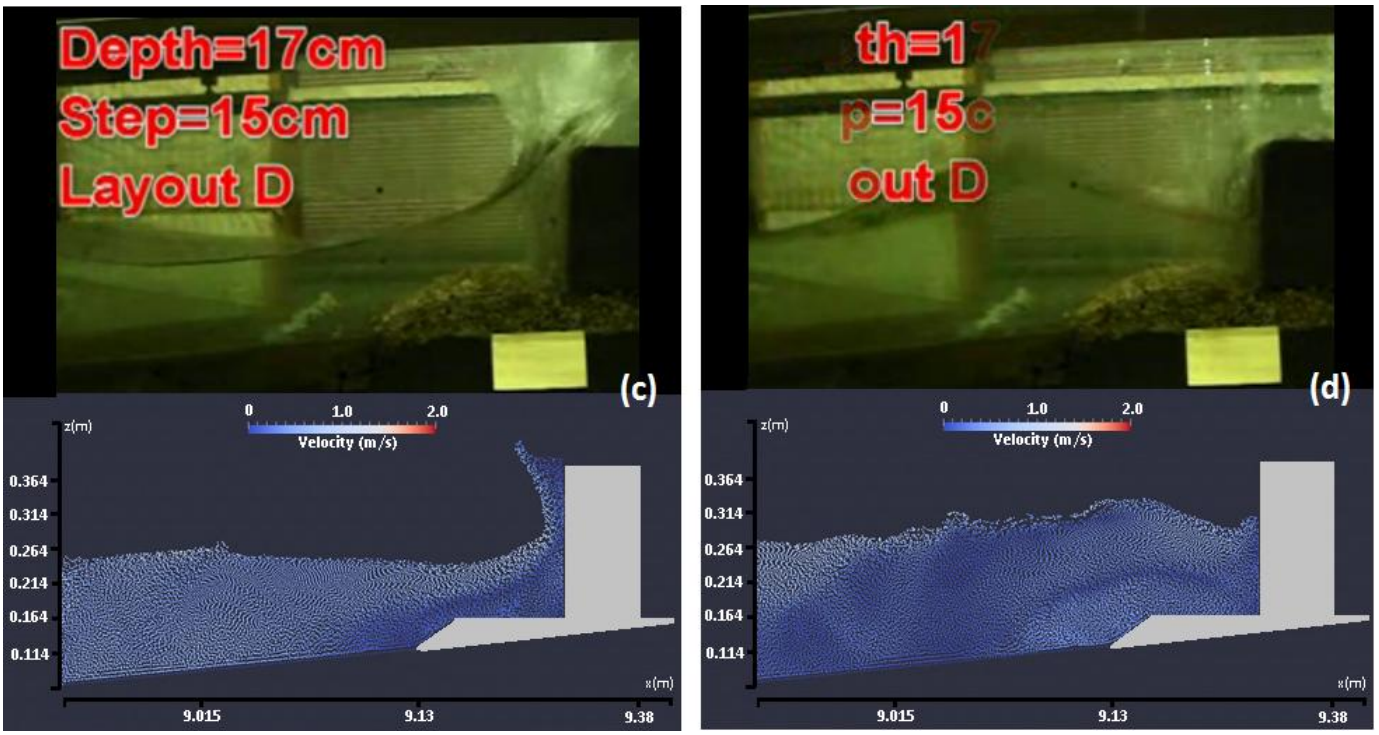

Figure 6-8 Impacting bore profiles from Esteban *et al.* (2008) (top) and numerical model (bottom) at simulation times (c) t = 4.5s (d) t = 4.95s

The mid-tank wave and incident wave height profiles can be seen in Figure 6-9 and 6-10 respectively, where the blue line represents the data from Esteban *et al.* (2008) and the red etched line represents the numerical results. In the case of the mid-tank wave height, there is a lag of 0.7s in the time at which the initial rise of wave height occurs between the numerical model and the results from the physical experiment. As the solitary wave passes, the wave height peaks at around 10cm in the numerical model compared to the 7.8cm recorded in the physical experiment results. The wave heights then gradually drop to the SWL after which the reflected wave train passes at a maximum of 10.2cm in the physical experiment results and at around 8.6cm in the numerical model. The numerical wave appears to have an increased wavelength comparing to the one observed in the physical experiment.

The fluctuations in the reflected wave train can be clearly seen in both the physical experiment results and the numerical model. These differences in the wave height may be due to the fact that in the physical model, the reservoir gate is lifted manually altering the way in which the water interacts with the existing still water in the channel to create this solitary/dam-break wave. In the numerical model, the release of the 'virtual' dam gate is instantaneous. Also, the fluctuations in the numerical solitary wave signal might be partly due to the kernel polynomial chosen or the particle viscosity effects. A sensitivity analysis for this experiment would be required in order to narrow down the possibilities.

The incident wave heights were taken by wave gauges placed at 0.25m from the caisson. The incident wave heights from the numerical model and the physical experiment appear to be in better agreement than the mid-tank wave heights. The wave reaches the gauge point at around 4.35s in the physical experiment, where it has a maximum height of approximately 20cm. The numerical wave also reaches a peak of 20cm but arrives at the gauge faster. The numerical incident wave shown in Figure 6-10, also indicates that its wavelength is longer than the wave from Esteban *et al.* (2008). The wave height then slightly fluctuates as the wave passes and gradually drops to zero.

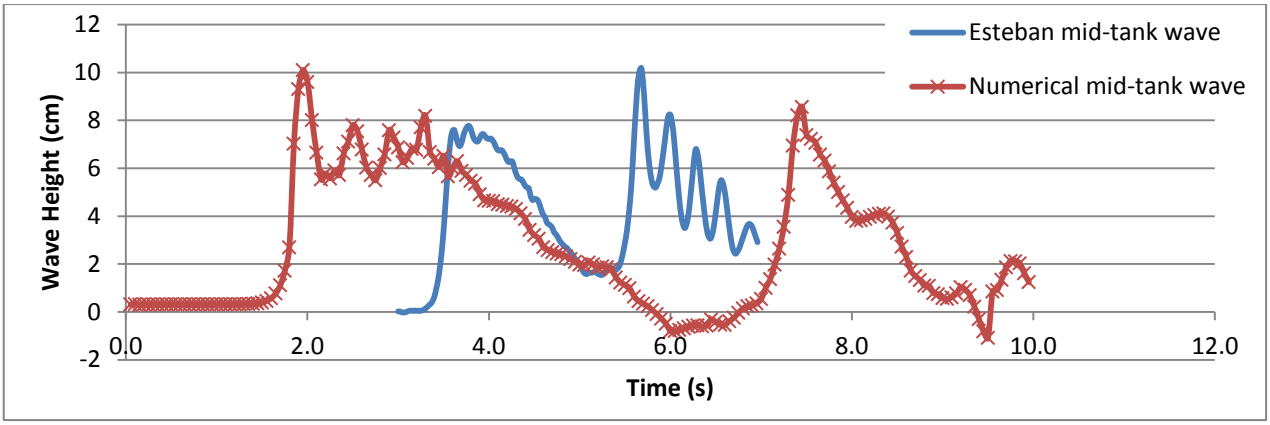

Figure 6-9 Comparison of mi-tank wave profile for numerical wave and Esteban *et al.* (2008) wave

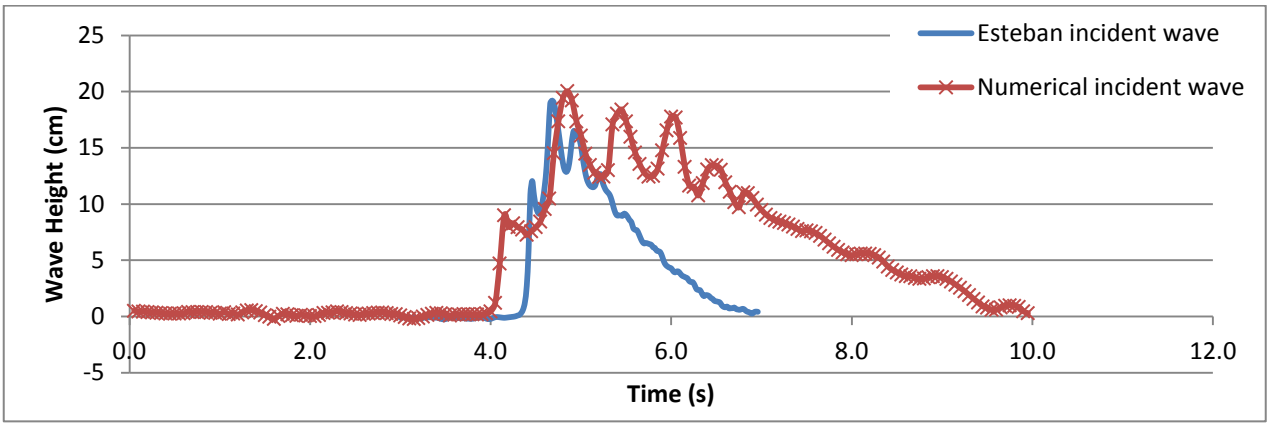

Figure 6-10 Comparison of incident wave profile for numerical wave and Esteban *et al.* (2008) wave

The impact pressure distribution on the caisson can be plotted with respect to the water depth at the front face of the caisson. Figure 6-11 shows how at various time steps, the impact pressure peaks at around the SWL at the caisson front face which is approximately 0.02m. This is considered typical in wave-structure interactions as shown previously by Bullock *et al.* (2007) and Goda (2000). Since Esteban *et al.* (2008) only provided the pressure distribution at the back base of the caisson, there is no direct way to compare the numerical model's pressure distribution on the face of the caisson to the physical one. In SPH, the only way to simulate and measure the pressure at the back base of the caisson would be to consider the caisson a moving object rather than a fixed boundary. This requires additional research in order to understand the theory behind how moving objects interact in the SPH domain with fluid particles and the fixed boundaries. Due to a lack of

additional time, this topic was not included in the scope of this research hence only the wave formation and propagation were focused on.

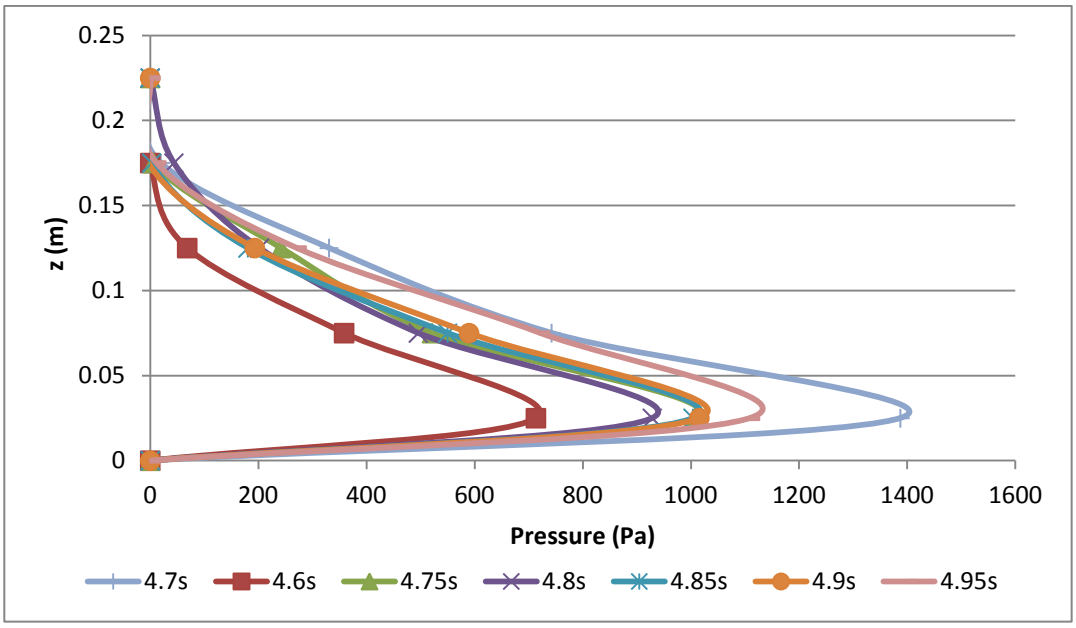

Figure 6-11 SPH model results of vertical impact pressure distribution on caisson front face for Esteban *et al.* (2008)

6.1.3 Hsiao and Lin (2010)

Bore Advancement Overtopping

In Hsiao and Lin (2010), the three different solitary wave cases that were considered are type 1, which is a turbulent bore, type 2, which is a wave directly breaking on the wall and type 3, which is the wave overtopping the wall directly. For this research, only wave type 1 was considered where the water depth is 0.2m, the wave height is 0.07m and the free-board height is 0.056m. Figure 6-12 shows the formation of the solitary wave for this case. The wave height ratio achieved by the model is 0.345 which is extremely close to the physical experiment result of 0.35. This means that the motion of the numerical wave-maker is accurate and the mathematical model by Goring (1978) can be used to effectively generate numerical solitary waves.

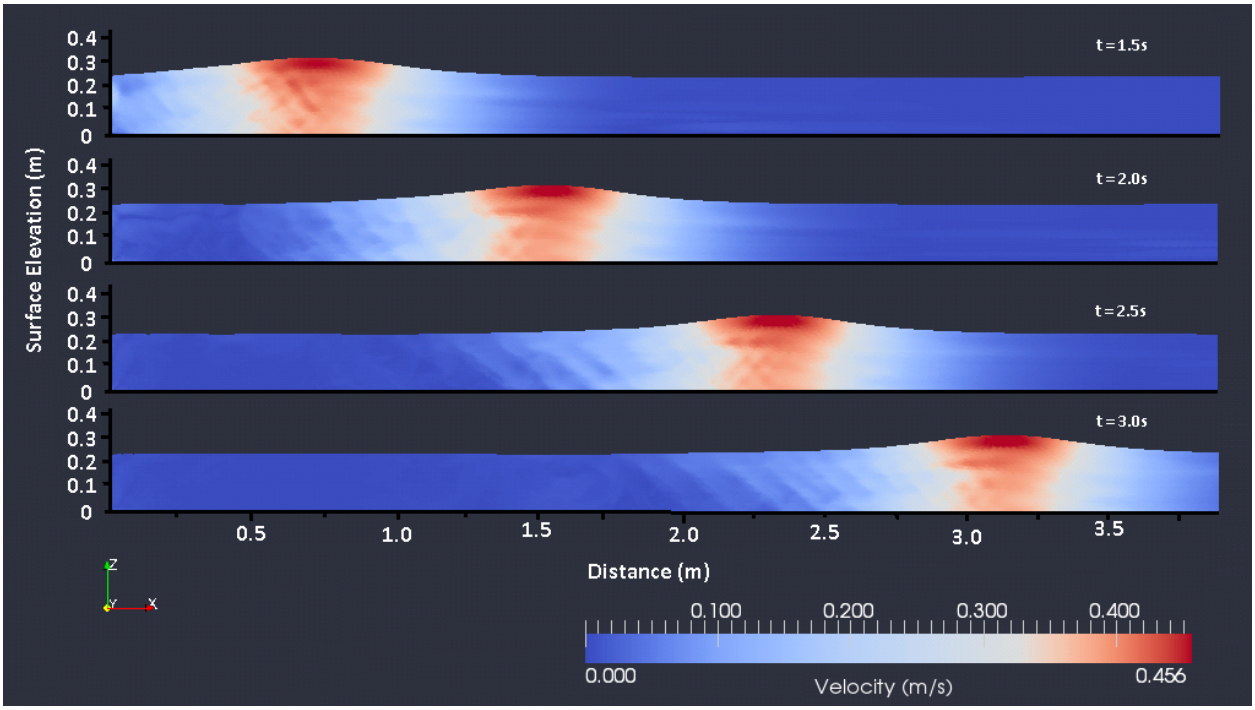

Figure 6-12 Wave formation in SPH model for Hsiao and Lin (2010)

It can be observed that at 2.0s, the wave has fully formed and is continues to propagate down the channel at 2.5s onwards. Also, Figure 6-12 shows the typical maximum velocity of the solitary wave at its centre. The wave board displacement and velocity distribution are shown in Figure 6-13 where the maximum velocity is around 0.42m/s and the wave board stroke is around 0.28m.

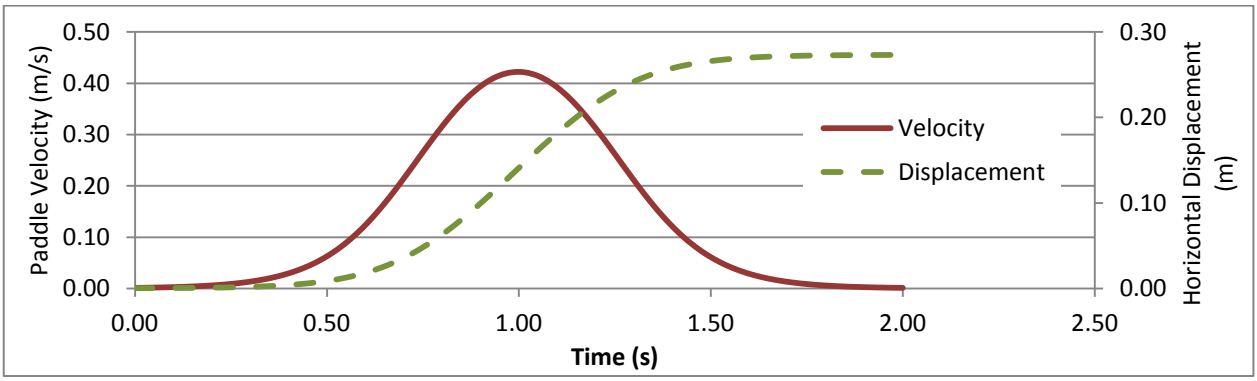

Figure 6-13 Wavemaker paddle velocity and displacement profiles for simulation of Hsiao and Lin (2010)

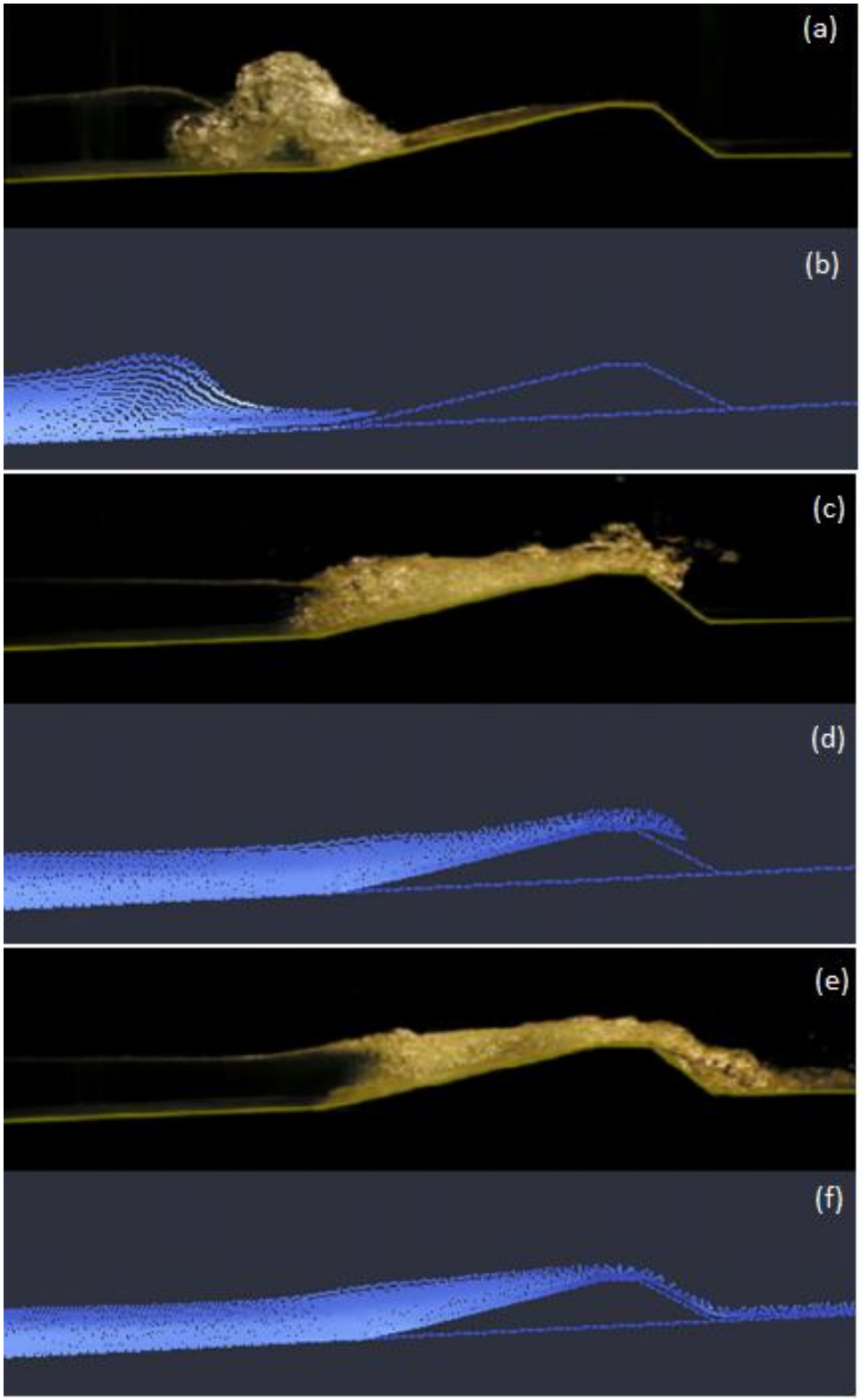

Figure 6-14 Comparison of stages of bore profile overtopping seawall for Hsiao and Lin (2010) and numerical model

The solitary wave then propagates down the channel and up the slope where it forms a bore-like wave that eventually impacts the seawall and overtops it. The impact and overtopping process is shown in Figure 6-14. Panels (a), (c) and (d) are from the physical experiment by Hsiao and Lin (2010) while panels (b), (d) and (f) are from the numerical model. When comparing panel (a) to (b), the approaching bore shows a high level of turbulence in (a) while the SPH model could not simulate this turbulence because the wave did not form a plunging breaker as early enough as in the physical experiment. The height of the bore, however, is very close to the height in the physical experiment. These differences may be due to the resolution of the model being too low for this case. Panels (c) and (d) show that the overtopping wave profiles are very close to one another except for the light turbulence at the surface of the water. Finally as the wave overtakes the entire seawall in (e) and (f), the profiles continue to match each other. Also, there is additional turbulence at the back of the seawall which can only be observed in the physical experiment.

The wave heights throughout the physical experiment were measured using wave gauges placed along the flume. The wave gauge locations can be seen in Figure 6-15 below.

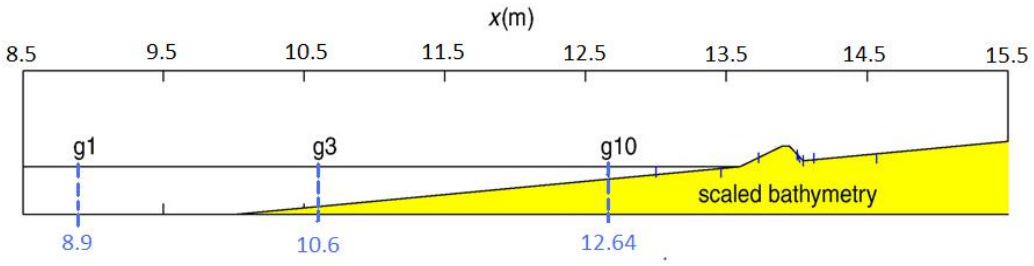

Figure 6-15 Locations of wave gauges g1, g3 and g10 (adapted from Hsiao and Lin (2010))

The wave heights obtained in the numerical model are compared to the physical experiment in Figures 6-16 to 6-18. The red line and blue line represent the physical experiment and the SPH model respectively. The SWL was set at 0.2m and the wave heights are measured from the SWL up to the top of the wave crest. Readings at gauge g1 show that the numerical model wave height starts to peak around 6.0s which is approximately at the same time as the physical experiment. The maximum wave height at gauge g1 is around

0.057m in the numerical model and 0.062m in the physical experiment which is a very small difference of 8 percent and is considered an accurate reading.

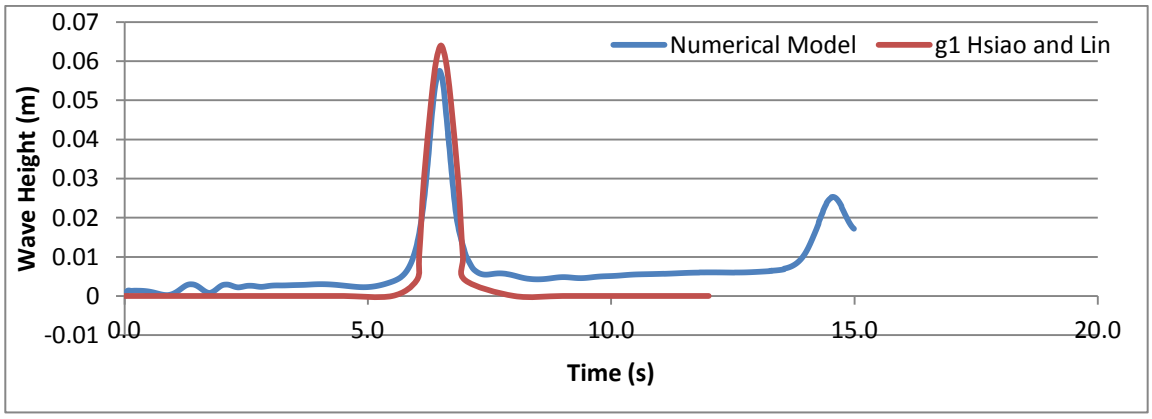

Figure 6-16 Wave gauge at x=8.9m, g1, readings from Hsiao and Lin (2010) and the numerical model

Closer to the beginning of the slope, at gauge g3, the wave height starts to peak around 7.0s to a maximum of 0.052m in the numerical model whereas it rises to 0.07m in the physical experiment. This is evidence that the wave should have started shoaling due to the oncoming change in the water level because of the slope. However, the simulated wave doesn't show signs of shoaling and maintains the average wave height. At around 12.5s, gauge g3 shows readings of an oscillatory reflected wave train. The maximum wave height for this reflected wave is around 0.03m in the numerical model while it is 0.026m in the physical experiment. Also, in the numerical model, the general water level seems to have swelled or increased by a value of 0.009m after the solitary wave passes.

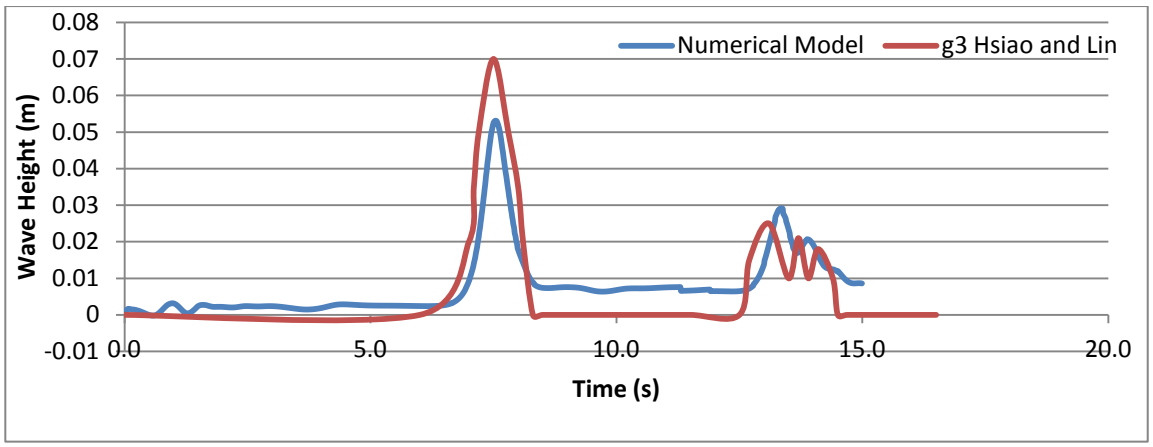

Figure 6-17 Wave gauge at x=10.6m, g3, readings from Hsiao and Lin (2010) and the numerical model

Lastly, at a distance of 12.64m, at gauge g10, the wave is close to impacting the seawall and the peak wave height starts to occur around 8.5s. The numerical model predicts a wave height of 0.06m while the physical results show a peak of 0.082m. This supports the fact that the numerical wave did not actually shoal to the correct height before breaking and impacting the seawall. The reflected wave train in the numerical model is shown to follow at the same time as the physical model at approximately 11.0s where the maximum wave height is around 0.03m.

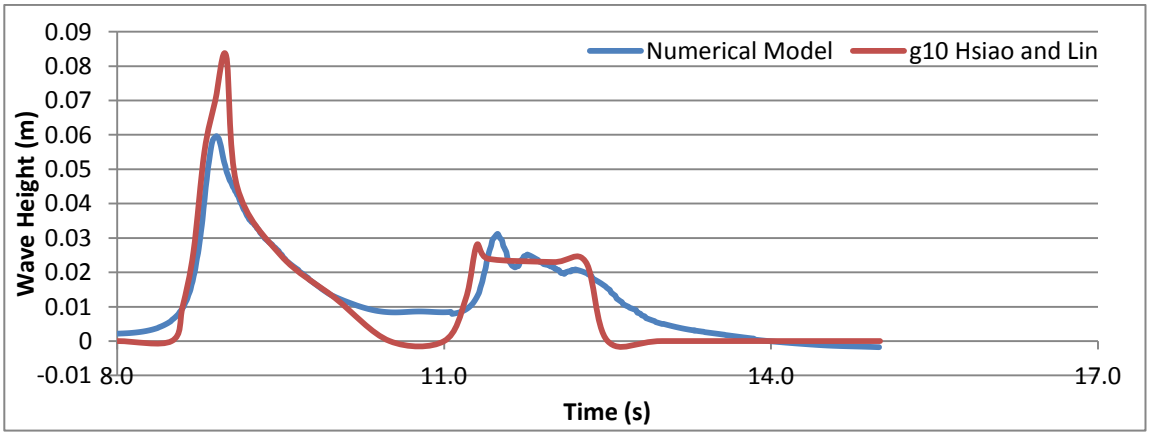

Figure 6-18 Wave gauge at x=12.64m, g10, readings from Hsiao and Lin (2010) and the numerical model

When finding the pressure on the sloped front of the seawall in the numerical model, the total pressure exerted by the wave impact on the wall is considered to act perpendicularly to the wall. Hence the force due to the impact pressure will have a horizontal and vertical

component. Typically, the pressure distribution on a wall for a dynamic load is in the shape of a triangle increasing downwards, from where the wave crest impacts the wall, to around or just above the SWL and then decreasing back to zero at the very base. This pressure distribution combined with the static pressure creates a sort of parabolic shaped pressure distribution on the wall. This is consistent with the pressure distribution proposed by Partenscky (1988). In the case of a sloped wall, the distribution shape is the same but slopes at the angle at which the wall slopes with the ground, as seen in the generalized schematic of the pressure distribution in Figure 6-19.

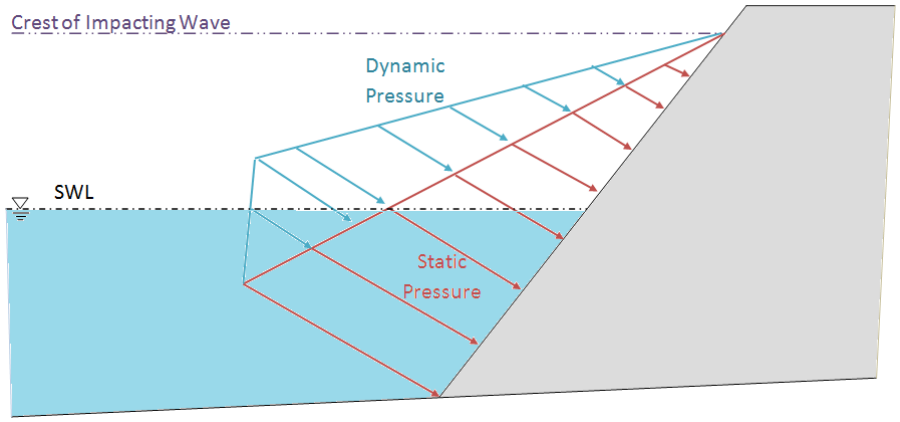

Figure 6-19 Generalized dynamic and static pressure distribution on sloping wall

As previously explained in section 5.3, the pressures on the front face of the seawall can be obtained in the SPH model and plotted versus the depth. In order to confirm the method of pressure calculation, a test case with the same layout as the Hsiao and Lin (2010) experiment was simulated using the SPH model but with simple static conditions and a SWL of 0.24m from the bottom of the domain. The static pressures at the locations of wave gauges 1 to 8 were calculated theoretically and compared to the numerical values obtained. Figure 6-20 shows a plot of these pressures versus water depth along the seaward face of the seawall where the blue and red lines represent the theoretical and numerical results respectively. Theoretically, the static pressure on an immersed sloping surface is found as the product of the water depth from the surface to the point of interest [d], the density [ρ] and the gravitational acceleration [g]. Figure 6-21 is a visualization of the pressure

distribution of the water particles in the SPHysics model. The pressures at the SWL are equal to zero (dark blue) and gradually increase as the water depth increases down to the bottom of the sloped region (dark red). In Figure 6-21, for the depth of 0.06m shown, the value of pressure acting on the lowest point is theoretically found to be 588.6Pa while in the numerical model the pressure values show to be close to 600Pa meaning that at first inspection, the pressure is being accurately modelled.

The pressures obtained by the numerical model closely match the theoretical results with an average error of 11 percent. Since the difference is considered low, the decision to use the pressure extraction code for the sloping model was made and it was used to obtain the pressures and forces simulated by the SPH model for the Hsiao and Lin (2010) case.

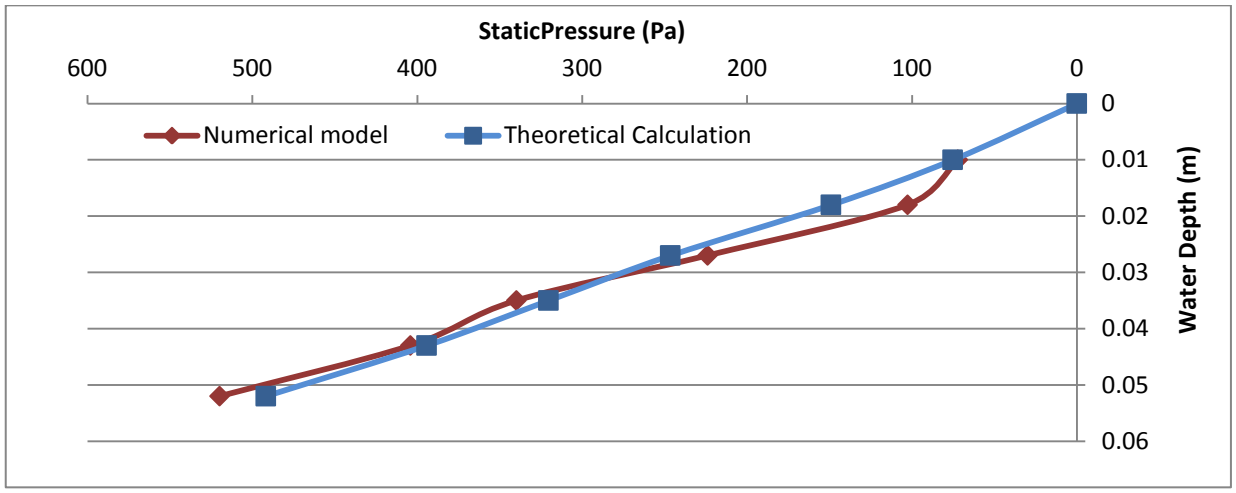

Figure 6-20 Numerical and theoretical calculation results of static pressure on seawall

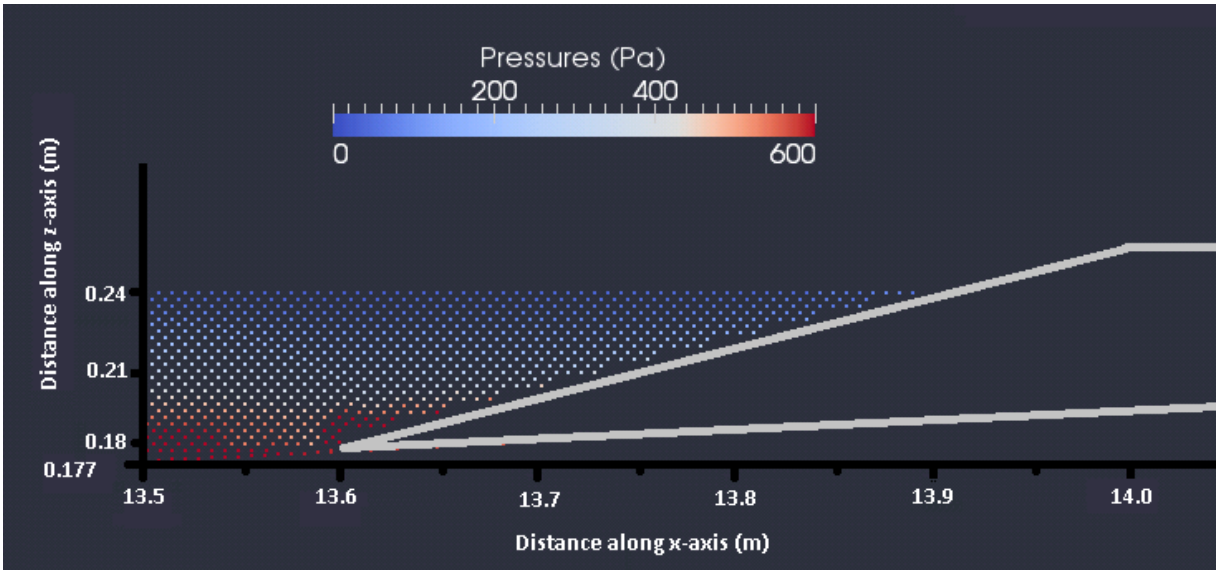

Figure 6-21 Static pressure on seawall in SPH model test (SWL of 0.24m)

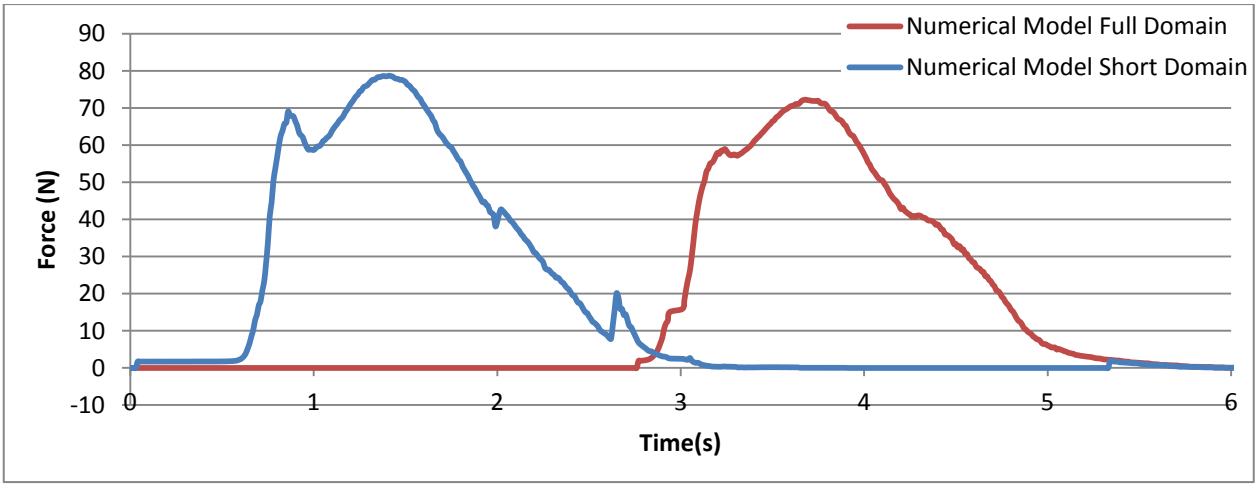

Figure 6-22 Comparison of numerical force-time histories of full and short domain lengths for Hsiao and Lon (2010)

As previously mentioned, for the SPH model of Hsiao and Lin (2010), two sets of simulations were completed: the full domain and a shortened domain. The purpose of this difference in layout is to show that, depending on the experiment, parts of the domain can be reduced such that there is little or no affect on the outcome in order to reduce computational time. Figure 6-22 shows the force time histories of the full and short domains as the red and blue lines respectively. In the full domain simulation, the wave reaches the seawall at around 2.3s after the shortened domain. This is as a direct result of shortening the domain and is

not considered an error. The maximum force is found to be around 80N in the shortened domain while the forces in the full domain reached a maximum of 73N. This slight difference may be caused by dissipative effects of the wave travelling a longer distance before reaching the seawall.

The numerical model pressure-time series for the full domain simulation, at all the wave gauge locations, is shown in Figure 6-23. The location of pressure gauge P1 is at the SWL at the seawall and the maximum pressure of 430Pa occurs around this point as shown by the dark blue line.

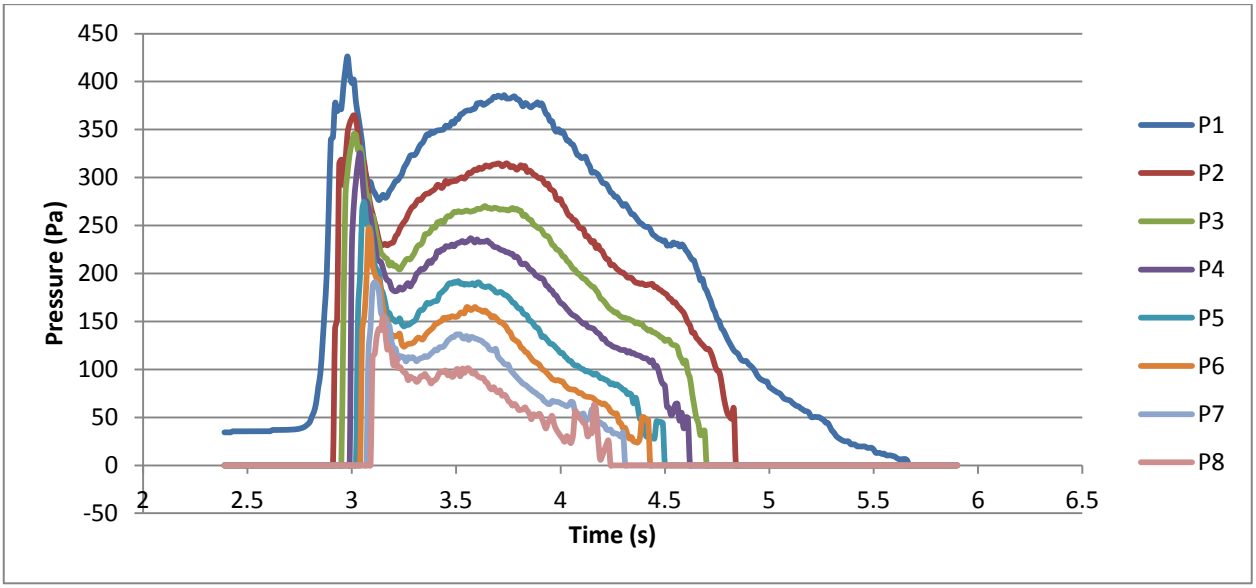

Figure 6-23 Impact pressure-time series at each wave gauge position from numerical model for Hsiao and Lin (2010)

The pressures obtained by the numerical model are now compared to the experiments by Hsiao and Lin (2010) in Figure 6-24. As shown, for all the pressure gauge locations, the numerical model results correctly depict the typical shape of the pressure-time history in the case of a non-vertical wall where the initial impact pressure is a sharp peak and drop followed by a secondary gradual increase and finally a decrease to zero. This type of pressure distribution is seen in research by Bullock *et al.* (2001).

The maximum pressures acting on the seawall are under-predicted by the numerical model by approximately 30% which is considered a large error especially if the numerical model is to be used in the design of a real breakwater. One of the main reasons that can be seen to cause such a drop in pressure magnitudes is that the wave did not reach the correct shoaling height and hence did not impact the seawall with the same amount of force as the physical experiment. Also, the choice of computational parameters would have affected the wave shoaling process. Hence, an in-depth sensitivity analysis for this case would be required to pinpoint the cause.

The force-time series exerted on the seawall is shown in Figure 6-25 where the solid line represents the physical experiment and the etched lines represent the numerical model. The forces on the seawall were obtained by integrating the pressure distributed across the front face of the seawall. Both horizontal and vertical forces are shown in the diagram. The maximum horizontal and vertical forces in the numerical model are found to be 70N per meter width of wall and 19N per meter width of wall respectively. In the physical experiment, however, the maximum horizontal and vertical forces were found to be 148N/m and 40N/m respectively. This discrepancy is an expected error carried forward since the pressures in the numerical model were found to be lower than in the physical experiment and the forces are integrated over the surface of the seawall from the pressure values.

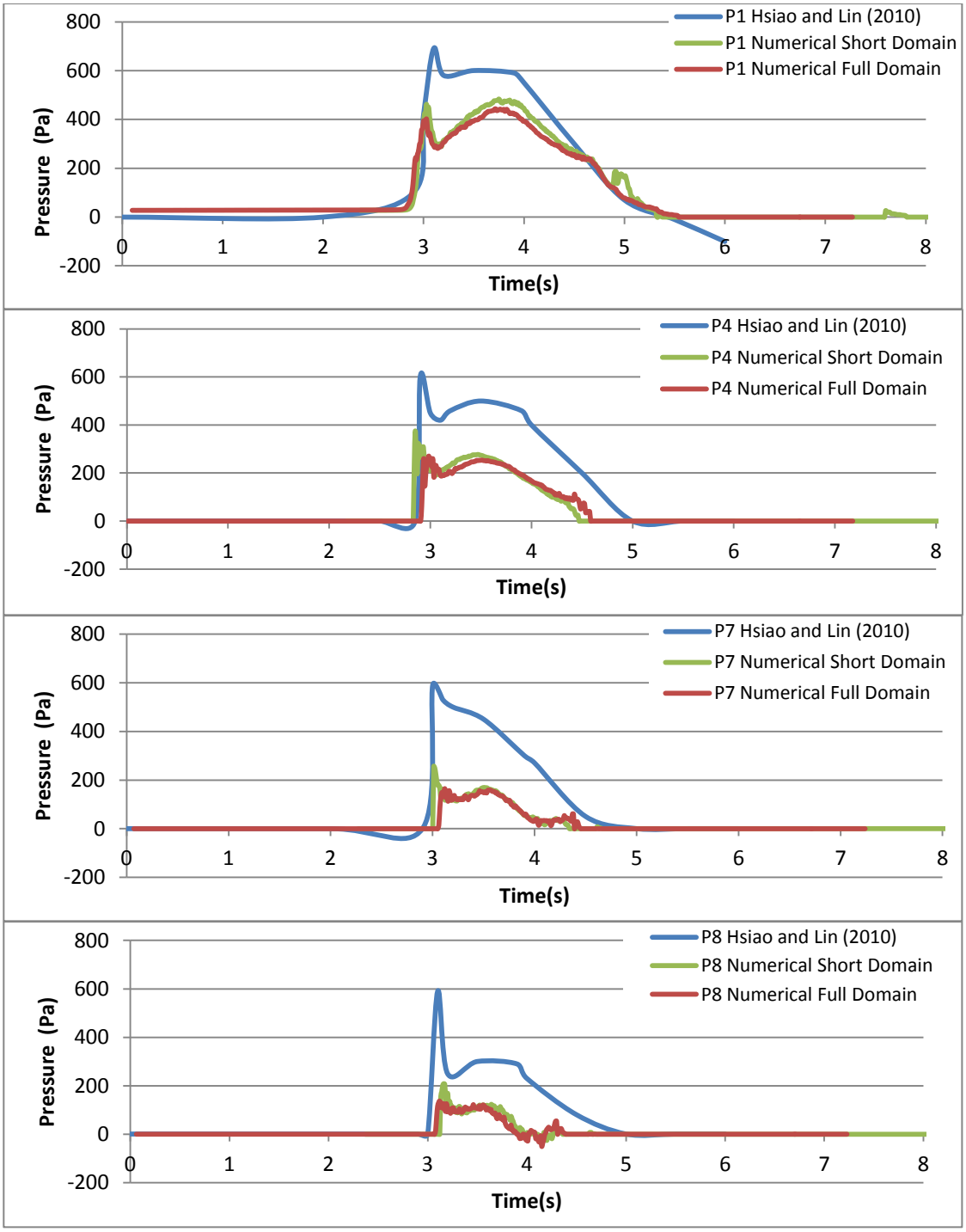

Figure 6-24 Pressure-time histories at pressure gauges p1, p4, p7 and p10 for the numerical model and Hsiao and Lin (2010)

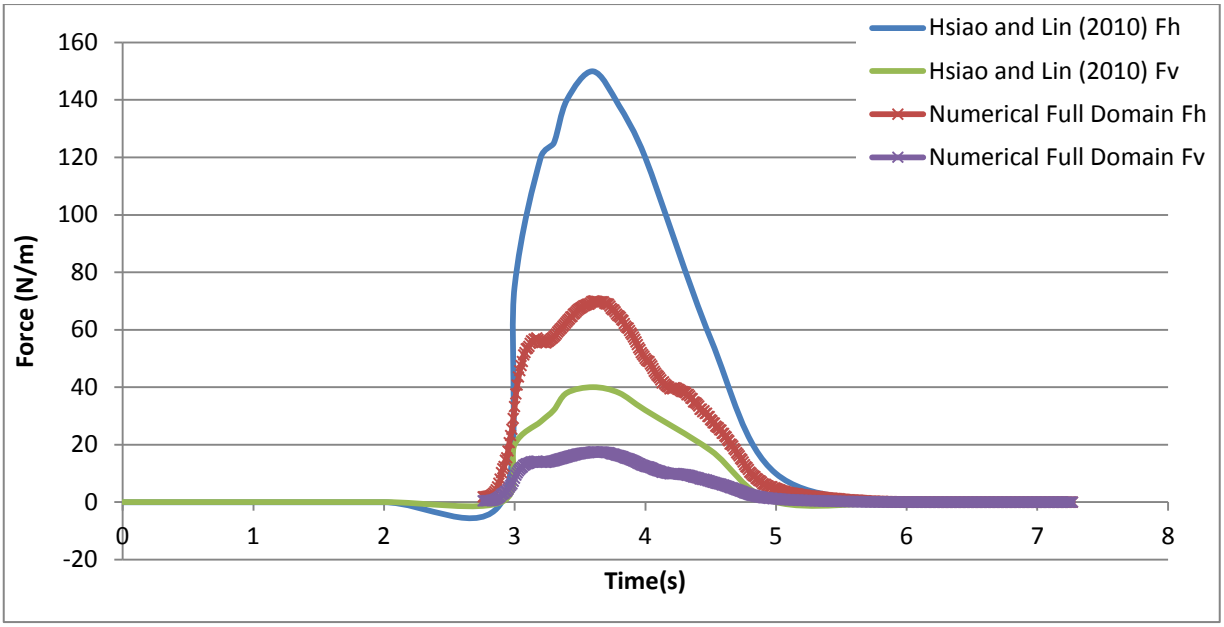

Figure 6-25 Horizontal and vertical force-time histories for both the numerical model and Hsiao and Lin (2010)

Wave Overtopping Rates

When designing seawalls, it is not sufficient to simply ensure the wall has the bearing capacity to withstand the impact loads, but to also estimate the possible wave overtopping rate in the case of an extreme wave event. Typically, the wave overtopping of a seawall is calculated using empirical formulae such as those used in the United Kingdom (UK) proposed by Owen (1980). The basic formula consists of the following relation between the overtopping rate Q and the dimensionless rate Q^*, as well as, the free-board height R_c:

$$Q^* = \frac{Q}{\sqrt{T_m H_s g}} \qquad \textit{(6-1)}$$

$$R^* = \frac{R_c}{T_m \sqrt{H_s g}} \qquad \textit{(6-2)}$$

where, T_m is the wave period at the toe of the wall and $H_s$is the significant wave height at the toe of the wall. According to TAW (2002), the overtopping rate can be calculated using

an empirical formula of dimensionless constants. For the case of a simple sloping seawall, as in Hsiao and Lin (2010), the overtopping rate can be found using:

$$Q^* = \frac{Q}{\sqrt{H_s^3 g}} = \frac{0.067}{\sqrt{tan\alpha}} \gamma_b \xi_m exp\left[-4.75 \frac{R_c}{\xi_m H_s \gamma_b \gamma_f \gamma_\beta \gamma_v}\right] \quad \textbf{(6-3)}$$

where, α is the slope of the front face of the structure, $\gamma_b$is the berm constant, γ_f is the friction constant, γ_β is the wave angle constant, γ_v is the vertical wall constant and ξ_m is the surf similarity parameter found using the following formula relating H_s to L_m, the deepwater wavelength:

$$\xi_m = \frac{tan\alpha}{\sqrt{H_s/L_m}} \quad \textbf{(6-4)}$$

In order to apply equation 6-3 to the Hsiao and Lin (2010) experiment, the dimensionless constants [$\gamma_b, \gamma_f, \gamma_\beta, \gamma_v$] were all taken as 1 since there is no berm, the seawall is smooth, the waves are perpendicular and there is no wall on top of the seawall. The surf similarity number or Irribarren number [ξ_m] was found to be 1.36 and is considered a plunging/spilling type breaker. Finally, by substituting the appropriate values into equation 6-3, the dimensionless overtopping rate was found to be 0.008 which equals an overtopping rate of 0.00042m^3/s/m. In order to estimate the overtopping rate in the SPH model, a counter was placed in the output extraction code in order to keep track of how many particles pass over the top of the seawall. For the simulation with the full domain, the SPH model dimensionless overtopping rate [Q^*] was found to be 0.00042 and the overtopping rate [Q] is 0.000022m^3/s/m. The numerical result is approximately one order of magnitude less than the design prediction which is a significant difference especially when considering the safety of the seawall in the design. It is assumed that the design equation is meant to over-predict the value of [Q]. There is no way to compare the overtopping rates of the numerical model to the physical model since these results were not provided by Hsiao and Lin (2010). Possible discrepancies in the overtopping rate may be caused by the lower wave heights seen in the numerical experiment and the lower amount of energy in the wave due to the smaller pressures exerted at impact. This will cause fewer

particles to reach over the top of the seawall. Also, the value of the inter-particle spacing will change the number of particles in the domain thus effecting how many particles can overtop the seawall.

6.2 Weakly Compressible SPH Assumption

After each numerical simulation was completed, the assumption that this model of SPH is for a weakly compressible fluid can be manually confirmed. This serves as a verification of the equations used in the model. In order to ensure this assumption is valid, Monaghan (1994) proposed that the variation of the densities calculated by the numerical model should not exceed 1% of the reference density value [$\rho_o = 1000 kg/m^3$]. Below is Table 6-1, showing the density variation for each experiment at the time of structure impact. All the density variations are well below 1% which means the assumption is valid and no unexpected pressure variations would have been observed.

Table 6-1 Density Variation for each numerical simulation

Experiment	Time of Impact (s)	Maximum Density (kg/m^3)	Percent Density Variation (%)
Ramsden (1993) Low Res.	13.5	1001	0.1
Ramsden (1993) High Res.	13.0	1003	0.3
Esteban *et al.* (2008)	4.2	1003	0.3
Hsiao and Lin (2010)	9.47	1004	0.4
Hsiao and Lin (2010) Short	8.0	1004	0.4

6.3 Discussion

The results obtained by the numerical simulations were compared qualitatively and quantitatively to the results of the physical experiments. In general, the SPH model used in this study was able to produce satisfactory results; however, with some limitations to be discussed.

The wave formation, propagation and breaking were successfully modelled by the high resolution simulation of the experiment by Ramsden (1993). Contrary to expectations, the impact forces on the vertical wall were accurately modelled by the low resolution simulation as opposed to the higher one. It is not clear as to why this occurred; however, it may be due to some combinational effect of the Riemann solver's beta-limiter and the extremely small inter-particle spacing [dx=dz=0.008].

Considering the experiment by Esteban et al. (2008), the wave generated by the numerical model started off slightly faster and higher than the physical wave around mid-tank but later the wave from both models synchronised just before impacting the caisson. This indicates that the wave had to have dissipated some of its energy in order to slow down and lose height. In the numerical model, as the wave impacted the caisson, there was no splash up observed as in the physical experiment. This is due to the assumption that this SPH model is weakly compressible and the pressure is calculated directly from the density of the particles.

For the experiments by Hsiao and Lin (2010), the SPH model was efficiently able to simulate the wave generation, propagation and breaking. The shape of the wave bore as it approaches the seawall is in good agreement with the physical experiment; however, it was observed that there was no turbulence in the bore front. As in the case of Esteban (2008), this is due to the weakly compressible assumption of the SPH model used. Also, the model is considered one-phase and so cannot simulate the air entrainment that would occur as the bore propagates up a slope. The times at which the wave heights were recorded by both the virtual (numerical) and the physical wave gauge were found to be approximately the same with a slight difference in the wave height values. The numerical model under-predicted the wave height values at the wave gauge locations which subsequently affected the values of the impact pressures.

The SPH model was also able to successfully model a hydrostatic case that was then used to calibrate the pressure extraction code for a sloping seawall. The shapes of the pressure- and force-time histories were found to mimic the results of Hsiao and Lin's (2010) experimental

results as well as other similar research experiments. The values of the forces, however, were under-estimated by the SPH model which leads to the need for a case-specific sensitivity study.

Overtopping rates were not provided in the results by Hsiao and Lin (2010); however, the overtopping rate was calculated for the numerical model results in order to provide an example of how this can be done. There aren't many precedent cases of modelling of overtopping rates using SPH. In this study, published design equations were used to approximate the overtopping rate of a wave on a seawall floor the case of Hsiao and Lin (2010). This result was then compared to the numerical result and was found to be an order of magnitude larger. This proves that for this case of seawall overtopping, the design equation provides a safe estimate.

7. Conclusions

The increased number of coastal disasters has raised the issue of improving coastal structures protection design in disaster-prone areas. A large amount of research has been conducted in the past three decades to physically and numerically model wave-structure interactions. However, not much advancement has occurred since Goda's (1985) and other similar equations had been developed. The use of a numerical model rather than a costly physical model provides many advantages such as a lower cost and the lack of scale limitations in some instances.

The present research focused on investigating the accuracy and robustness of a two-dimensional weakly compressible SPH-based numerical model used to simulate various scenarios derived from physical experiments. The intent was to add to the variety of existing studies and to provide a novel set of research data and results showing the impacts of rapidly-propagating hydraulic bores on structure and to analyze the wave-structure interactions. The following conclusions can be made from this research:

- The numerical model used in this research was able to successfully simulate all wave-processes studied. It was observed that, in some instances, wave heights were under-estimated hence leading to differences in the pressures exerted by the wave on the wall at impact between the experimental and numerical work.
- The analyses presented in Chapters 5 and 6 demonstrate the sensitivity of the investigated SPH model to some of the parameters. Depending on the analyzed case, in order to achieve more accurate results, the influence of each parameter had to be investigated and the test results analyzed before proceeding to the next parameter in question.
- The quantitative results of the simulations seem to be most affected by two parameters: the inter-particle spacing and the deployment of the Riemann solver.

- The drawback of using a Riemann solver is that at lower particle resolutions, the numerical simulation experiences increased viscosity effects that negatively affect the pressures exerted by the water particles at impact.
- The sensitivity analysis performed in Chapter 5 demonstrated that an increased inter-particle spacing, or higher particle resolution, allowed for a more stable and physically sound estimation of the force-time history.

8. Recommendations for Future Work

The analysis performed in the present research has allowed for the realization of the shortcomings of the model. As a result, the following recommendations for future work are provided.

It would be of great advantage if the simulation times were greatly diminished through the use of the GPU versions of the SPH model as discussed in section 3.5. Due to the time and scope of this research project, it was not possible to perfect the use of DualSPHysics or GPUSPH in order to simulate the above physical experiments. Since the source codes for these models have only been released within the last year, there is a large potential for new case examples to be modelled and research to be conducted on wave-structure interactions.

The comparison of the SPHysics model alongside other Lagrangian and non-grid-based numerical models is rare and should be considered if the model is to be improved and made more popular.

Since the focus of this research is wave-processes and subsequent structure interactions, the phenomenon of air entrainment in waves is critical especially when considering impact force-time histories. This can be achieved by using a multi-phase SPH model to consider the aeration. It will probably require additional computational time and other considerations that may be well worth improving the current results of impact forces.

Additional research regarding the comparison of compressible SPH models to weakly compressible models would provide more insight to which is better for the wave-structure interaction scenario.

9. References

American Society of Civil Engineers (2010). Minimum Design Loads for Buildings and Other Structures. ASCE/SEI 7-10.

Azarmsa, S., Yasuda, T. and Mutsuda, H. (1996a). Breaker-Type Effect on Impact Pressure Exerted on a Vertical Wall. J. Coastal Eng. in Japan. Vol. 39, No. 1, JSCE, pp. 39-57.]

Azarmsa, S., Yasuda, T. and Mutsuda, H. (1996b). Cause and Characteristics of Impact Pressure Exerted by Spilling and Plunging Breakers on a Vertical Wall. Coastal Engineering Conference Proceeding Paper, ASCE, 2442-2455.

Bergman, H. and Oumeraci, H. (1998). Wave Pressure Distribution on Permeable Vertical Walls. Coastal Engineering, 2042-2055.

Blackmore, P. and Hewson, P. (1984). Experiments on Full-Scale Wave Impact Pressures. Coastal Engineering Journal, 8, 331-346.

Bonet, J. and Lok, T.-S. L. (1999). Variational and Momentum Preservation Aspects of Smoothed Particle Hydrodynamics Formulations. Computational Methods Applied Mechanics Engineering, 180, 97-115.

Boussinesq, J. (1872). Theorie des ondes et des remous qui se propagent le long d'un canal rectangulaire horizontal, en communiquant au liquide contenu dans ce canal des vitesses sensiblement pareil les de la surface au fond. J. Math. Pures Appl., 17, 55-108.

Bullock, G., Obhrai, C., Muller, G., Wolters, G., Peregrine, H. and Bredmose, H. (2001). Field and Laboratory Measurement of Wave Impacts. Coastal Structures 2004, ASCE, 343-355.

Bullock, G., Obhrai, C., Peregrine, D. and Bredmose, H. (2007). Violent Breaking Wave Impacts. Part I: Results from Large-scale Regular Wave Tests on Vertical and Sloping Walls. Coastal Engineering, 602-617.

Capone, T., Panizzo, A. and Dalrymple, R. (2007). Accuracy and Stability of Numerical Schemes in SPH. *Secon SPHERIC Workshop. Madrid.*

Chan, E.-S. (1994). Mechanics of Deep Water Plunging-wave Impacts on Vertical Structures. Coastal Engineering, 22, 115-133.

Chanson, H. (2006). Tsunami Surges on Dry Coastal Plains: Application of Dam Break Wave Equations. Coastal Engineering Journal, Vol. 48, No. 4, 355-370.

Chen, G., Kharif, C., Zaleski, S. and Li, J., (1999). Two-dimensional Navier-Stokes Simulation of Breaking Waves. *Physics of Fluids,* 11: 121-133.

CNN Wire Staff (2011). "Japan marks 1 month since deadly quake". Available at http://www.cnn.com/2011/WORLD/asiapcf/04/11/japan.disaster.month/index.html.

Colagrossi, A. and Landrini, M. (2003). Numerical Simulation of Interfacial Flows by Smoothed Particle Hydrodynamics. J. Computational Physics, 191, 448-475.

Crespo, A., Gomez-Gesteira, M., Dalrymple, R. A. (2008). 3D Simulation of Large Waves Mitigation with a Dike. Journal of Hydraulic Research, Vol. 45, No. 5, 631–642.

Crespo, A., Dominguez, J., Gómez-Gesteira, M., Barreiro, A., Rogers, B. (2011a), User Guide for the DualSPHysics Code v1.0, http://www.dual.sphysics.org.

Crespo, A., Dominguez, J., Barreiro, A., Gómez-Gesteira, M. and Rogers, B. (2011b). GPUs, a New Tool of Acceleration in CFD: Efficiency and Reliability on Smoothed Particle Hydrodynamics Methods. PLoS ONE. doi:10.1371/journal.pone.0020685.

Cuomo, G., Allsop, W., Bruce, T. and Pearson, J. (2010).Breaking Wave Loads at Vertical Seawalls and Breakwaters. Coastal Engineering, 57, 424–439.

Dalrymple, R.A. and Rogers, B.D. (2006). Numerical Modeling of Water Waves with the SPH method. Coastal Engineering, 53, 141 -147.

Didier, E. and Neves, M. G. (2010). A Lagrangian Smoothed Particle Hydrodynamics Method for Modelling Waves – Coastal Structure Interaction. Proceedings of the Fifth European Conference on Computational Fluid Dynamics.

Dilts, G. (1999). Moving-Least–Squares-Particle Hydrodynamics – I. Consistency and Stability. Int. J. Numer. Meth. Engng, 44, 1115-1155.

FEMA (Federal Emergency Management Agency) (2011). Coastal Construction Manual, Fourth Edition. FEMA P-55, Washington, USA.

FEMA (Federal Emergency Management Agency) (2008). Guidelines for Design of Structures for Vertical Evacuation from Tsunamis. FEMA P-646, Washington, USA.

Franco, L., de Gerloni, M. and van der Meer, J.W. (1994). Wave Overtopping on Vertical and Composite Breakwaters. Coastal Engineering, 1030-1045.

Gerrits, J., Veldman, A. (2003). Dynamics of Liquid-filled Spacecraft. Journal of Engineering Mathematics, 45, 21-38.

Gingold, R.A. and Monaghan, J.J. (1977). Smoothed Particle Hydrodynamics: Theory and Application to Non-spherical Stars. Monthly Notices of the Royal Astronomical Society, 375-389.

Gomez-Gesteira, M. and Dalrymple, R. (2004). Using a Three-dimensional Smoothed Particle Hydrodynamics Method for Wave Impact on a Tall Structure. Journal of Waterway, Port, Coastal and Ocean Engineering, Vol. 130, No. 2, 63-69.

Gómez-Gesteira, M., Rogers, B.D., Dalrymple, R.A., Crespo, A.J.C. and Narayanaswamy, M. (2010a). User Guide for the SPHysics Code v2.0, http://www.sphysics.org.

Gómez-Gesteira, M., Rogers, B.D., Dalrymple, R.A., Crespo, A.J.C. (2010b). State of the Art of Classical SPH for Free-surfaceFlows. Journal of Hydraulic Research, 48, 6 -27.

Gómez-Gesteira, M., Rogers, B.D., Crespo, A.J.C., Dalrymple, R.A., Narayanaswamy, M. and Dominguez, J.M. (2012a). SPHysics - development of a free-surface fluid solver- Part 1: Theory and Formulations. Computers & Geosciences, 48, 289–299.

Gómez-Gesteira, M., Crespo, A.J.C., Rogers, B.D., Dalrymple, R.A., Dominguez, J.M. and Barreiro, A. (2012b). SPHysics - development of a free-surface fluid solver- Part 2: Efficiency and test cases. Computers & Geosciences, 48, 300–307.

Goring, D. G. (1978). Tsunamis – The Propagation of Long Waves Onto a Shelf. Doctoral Dissertation, Report No. KH-R-38, Keck Laboratory of Hydraulics and Water Resources, California Institute of Technology, Pasadena, California.

Hasselmann, D., Dunckel, M., Ewing, J. (1980) Directional Wave Spectra Observed during JONSWAP 1973. J. Phys. Oceanogr., 10, 1264–1280.

Hattori, M., Arami, A. and Yui, T. (1994). Wave Impact Pressure on Vertical Walls under Breaking Waves of Various Types. Coastal Engineering, 22, 79-114.

Hérault, A., Bilotta, G., Dalrymple, R., Rustico, E., Del Negro, C. (2010). GPUSPH (Version 2.0) [Software] Available from www.ce.jhu.edu/dalrymple/GPUSPH.

Hérault, A., Bilotta, G., Dalrymple, R. (2010). SPH on GPU using CUDA. Journal of Hydraulic Research, 48 Extra Issue: 74-79.

Hofland, B., Kaminski, M.L and Wolters, G. (2010). Large-scale Impacts on a Vertical Wall. Coastal Engineering, No. 32.

Hsiao, S.-C. and Lin, T.-C. (2010). Tsunami-like Solitary Waves Impinging and Overtopping an Impermeable Seawall: Experiment and RANS Modeling. Coastal Engineering, 57, 1–18.

Kirkgoz, M. S. (1990). An Experimental Investigation of a Vertical Wall Response to Breaking Wall Impact. Ocean Engineering, Vol. 17, No 4, 379-391.

Kirkgoz, M.S. (1995). Breaking Wave Impact on Vertical and Sloping Coastal Structures. Ocean Engineering, Vol. 22, No. 1, 35-48.

Kirkgoz, M.S. and Mamak, M. (2004). Impulse Modelling of Wave Impact Pressures on Vertical Wall. Ocean Engineering, 31, 343‑352.

Kortenhaus, A., Miller, C. and Oumeraci, H. (1996). Design of Vertical Walls Against Storm Surge. Coastal Engineering, 1403-1416.

Lee, E., Moulinec, C., Xuc, R., Violeau, D., Laurence, D., Stansby, P. (2008). Comparisons of Weakly Compressible and Truly Incompressible Algorithms for the SPH Mesh-free Particle Method. Journal of Computational Physics, 227, 8417–8436.

Li, S. and Liu, W. (1996). Moving Least Square Reproducing Kernel Method Part II: Fourier Analysis. Journal of Fluid Mechanics, Vol. 139, 15, 159‑193.

Lin, P. and Liu, P. (1998). A Numerical Study of Breaking Waves in the Surf Zone. Journal of Fluid Mechanics. 359, 239-264.

Liu, G.-R., and Liu, M. B. (2003). Smoothed Particle Hydrodynamics: a meshfree particle method. World Scientific Publishing Co.

Liu, P. and Al-Banaa, K. (2004). Solitary Wave Runup and Force on a Vertical Barrier. Journal of Fluid Mechanics, vol. 505, 225–233.

Lo, E. and Shao, S. (2002). Simulation of Near-shore Solitary Wave Mechanics by an Incompressible SPH Method. Applied Ocean Research, 24, 275–286.

McConnell, K., Kortenhaus, A. (1997). Analysis of Pressure Measurements from Hydraulic Model Tests and Prototype Measurements. Proc. 1st Overall Project Workshop, Las Palmas, Annex C3, PROVERBS project, Leichtweiss Institut, Technical University of Braunschweig, Braunschweig, Germany, MAST III.

Minikin, R. (1963). Winds, Waves and Maritime Structures, 2nd edition. Charles Griffin, London, UK.

Mokrani, C., Abadie, S., Grilli, S. and Zibouche, K. (2010). Numerical Simulation of the Impact of a Plunging Breaker on a Vertical Structure and Subsequent Overtopping Event using a Navier-

Monaghan, J.J. (1994). Simulating Free Surface Flows with SPH. J. of Comp. Phy., 110, 399-406.

Monaghan, J.J. and Kos, A. (1999). Solitary waves on a Cretan beach. J. of Waterway, Port, Coastal and Ocean Engineering. 125, 145-154.

Morris, J., Fox, P. and Shu, Y. (1997). Modeling Lower Reynolds Number Incompressible Flows using SPH. Journal of Computational Physics. 136, 214-226.

NOAA National Geophysical Data Center (2012). “Tsunami Data and Information”. Available at http://www.ngdc.noaa.gov/hazard/tsu.shtml.

Okada, T., Sugano, T., Ishikawa, T., Ohgi, T., Takai, S. and Hamabe, C. (2006). Structural Design Method of Buildings for Tsunami Resistance. Building Technology Research Institute, Building Center of Japan.

Okamura, M. (1993). Impulsive Pressure due to Wave Impact on an Inclined Plane Wall. Fluid Dynamics Research, 12, 215-228.

Oumeraci, H., Klammer, P. and Partenscky, H.-W. (1993). Classification of Breaking Wave Impact Loads on Vertical Structures. J. of Waterway, Port, Coastal and Ocean Engineering, 381-397.

Owen, M. (1980). Design of Seawalls Allowing for Overtopping. HR Wallingford, Wallingford, Report EX924.

Owen, M. (1982). Overtopping of Sea Defences. Proceedings of the Conference on Hydraulic Modelling of Civil Engineering Structures, Coventry. BHRA, Bedford, 469—480.

Parshikov, A. (1999). Application of a solution of the Riemann Problem to the SPH Method. Computational Mathematics and Mathematical Physics, 39, 1173.

Partenscky, H. (1988). Dynamic Forces due to Waves Breaking at Vertical Coastal Structures. Coastal Engineering, 2504-2518.

Peregrine, D. and Topliss, M. (1994). The Pressure Field due to Steep Water Waves Incident on a Vertical Wall. Coastal Engineering, 1496-1510.

Peregrine, D. H. (2003). Water-wave Impact on Walls. Annual Review of Fluid Mechanics, 35, 23–43.

Ramsden, J. (1993). Tsunamis: Forces on a Vertical Wall Caused by Long Waves, Bores and Surges on a Dry Bed. Thesis, California Institute of Technology, Pasadena, California.

Ramsden, J. (1996). Forces on a Veritcal Wall due to Long Waves, Bores and Dry-Bed Surges. J. of Waterway, Port, Coastal and Ocean Engineering, 134-141.

Ramsden, J. and Raichlen, F. (1990). Forces on Veritcal Wall Caused by Incident Bores. J. of Waterway, Port, Coastal and Ocean Engineering, 116, 592-613.

Randles, P. and Libersky, L. (1996). Smoothed Particle Hydrodynamics: Some Recent Improvements and Applications. Computational Methods Applied Mechanical Engineering, 139, 375-408.

Rogers, B., Dalrymple, R. and Stansby, P. (2010). Simulation of Caisson Breakwater Movement Using 2-D SPH. Journal of Hydraulic Research, Vol. 48, Extra Issue, 135–141.

Schmidt, R., Oumeraci, H. and Partenscky, H.-W. (1992). Impact Loads Induced by Plunging Breakers on Vertical Structures. Coastal Engineering, 1545-1588.

Shao, S. (2005). SPH Simulation of Solitary Wave Interaction with a Curtain-type Breakwater. Journal of Hydraulic Research, Vol. 43, No. 4, 366-375.

Silvester, T. and Cleary, P. (2006). Wave-structure Interaction using Smoothed Particle Hydrodynamics. Proceedings of Fifth international Conference on CFD, Melbourne, Australia.

Stokes VOF model. Proceedings of the Twentieth (2010) International Offshore and Polar Engineering Conference.

Tait, P. (1888). Report on Some of the Physical Properties of Fresh Water and Sea Water. Physical Chemistry, 2.

Tanizawa, K. and Yue, K. (1992). Numerical Computation of Plunging Wave Impact Loads on a Vertical Wall. Part 2: The Air Pocket. Proceeding of 7th Annual Workshop on Water Waves and Floating Bodies.

TAW (2002). Technical Report – Wave Runup and Wave Overtopping at Dikes. Technical Advisory Committee for flood defence in the Netherlands, Delft.

Toro, E. (2001). Shock Capturing Methods for Free Surface Shallow Flows. John Wiley & Sons.

Viccionne, G. and Carratelli, E. (2008). Simulation of Wave Impact Pressure on Vertical Structures with the SPH Method. Proceedings of the 3rd ERCOFTAC SPHERIC Workshop on SPH Applications.

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

Verlet, L. (1967). Computer Experiments on Classical Fluids I: Thermodynamical Properties of Lennard-Jones Molecules. Physics Review. 159, 98-103.

Wemmenhove, R., Loots, G. and Veldman, A. (2006). Numerical Simulation of Hydrodynamic Wave Loading by a Compressible Two-phase Model. Proceedings of European Conference on Computational Fluid Dynamics (2006), the Netherlands.

Wu, T. (2004). A Numerical Study of Breaking Waves and Turbulence effects. Ph.D. Thesis, Cornell University.

Yamamoto, Y., Takanashi, H., Hettiarachchi, S., Samarawickrama, S. (2006). Verification of the Destruction Mechanism of Structures in Sri Lanka and Thailand due to the Indian Ocean Tsunami. Coastal Eng. J. 48 (2), 117–145.

Xiao, H. and Huang, W. (2008). Numerical Modeling of Wave Runup and Forces on an Idealized Beachfront House. Ocean Engineering, 35, 106–116

Appendices

A. Numerical Model Output Sampling Codes

A.1 Extraction Code for Vertical Wall

The direct output of the SPHysics code consists of ASCII text files, each with columns of the particles' properties per time step. In the case of a two dimensional simulation, each text file will contain seven columns: the x and z position of the particle, the velocity along the x and z axes and the density, pressure and mass of that particle. Each row represents one particle in the simulation. Depending on the number of particles, there can be thousands, if not millions, of lines of data in each output file. An efficient way to read this data is to use an extraction code that will 'pull out' the required columns, such as the pressure, in order to plot a time history. The ASCII files are labelled as 'PART' files and the corresponding time history of a simulation is stored in an ASCII file called 'DT'. As discussed in section 5-2, a numerical procedure was implemented from research and code developed throughout Philippe St-Germain's master's thesis for a three-dimensional case. This code is written in C++ programming language using Microsoft Visual Studio. The source of the extraction program has three main components: the main program called 'data_analysis.cpp' and two header files called 'particle.h' and 'sampling_volume.h'. The complete set of code is shown in sections A.1.2, A.1.3 and A.1.4. The green text in the code represents comments describing what he purpose of the lines of code is.

A.1.1 Description of Input File

The extraction code accepts an input file that contains a column of details required to complete the extraction. A sample of this input file can be seen in Figure A.1 below. The red labels indicate what each entry represents. The first three lines indicate the number of PART files produced by the simulation as well as the total number of particles and boundary particles. The next tree lines represent the number of columns of sampling areas to be analysed in iterations, the direction along which the analyses will be performed (CX in this case since it is a 2D model) and the distance between the iterations. The three lines

following represent the initial location of the sampling area centre and the minimum and maximum limits on the z-axis. The last three lines represent the x- and z-dimension of the sampling area chosen and the property to be analysed. To specify which property is to be analysed the following input should be used for the extraction file: 'PRESSURE' for the pressure, 'VELOCITY' for the velocity and 'Z_POS' for the virtual wave gauge.

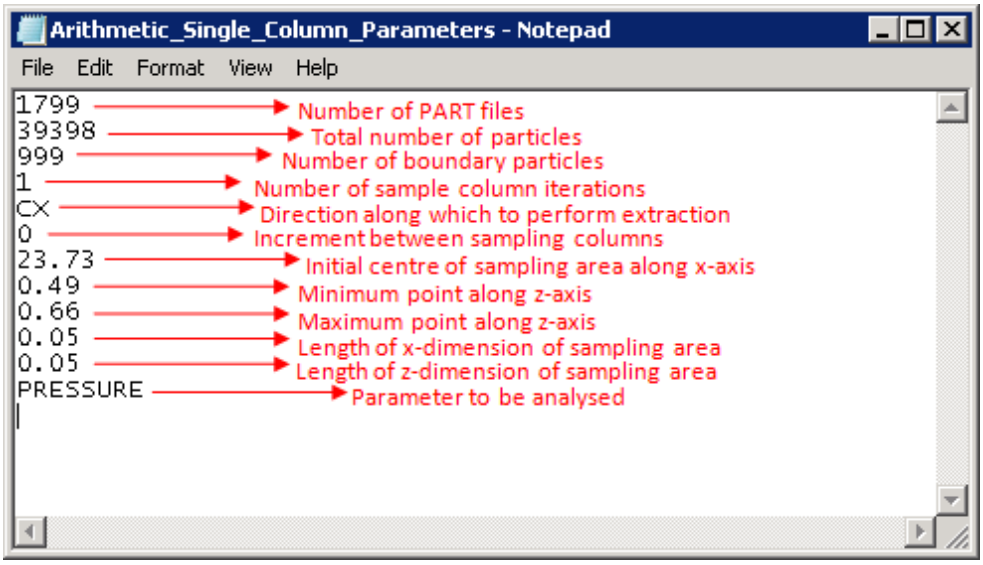

Figure A-0-1 Sample input file for extraction code

A.1.2 Header File 'particle.h'

```
#ifndef PARTICLE_H
#define PARTICLE_H

namespace post_processing {

	class Particle {
		public:
			//Attributes associated to a water particle
			double x_pos;
			double z_pos;
			double x_vel;
			double z_vel;
			double density;
			double pressure;
			double mass;
	};//End of class Particle

}

#endif
```

A.1.3 Header File 'sampling_volume.h'

```
#ifndef SAMPLING_VOLUME_H
#define SAMPLING_VOLUME_H

#include "particle.h"
#include <math.h>
#include <iostream>
#include <vector>

namespace post_processing {

	class SamplingVolume {

		public:
			//Array of particles inside the sampling volume
			std::vector <Particle> particle_sample;

			//Center coordinates of sampling volume
			double center_x;
			double center_z;

			double x_dim;
			double z_dim;

			//Function for the approximation of the average x-
velocity
			double avg_x_vel(){
				double sum=0;
				for(int i=0; i<particle_sample.size();i++){
					sum += particle_sample[i].x_vel;
				}
				return sum/particle_sample.size();
			}//End of function avg_x_vel

			//Function for the approximation of the average z-
velocity
			double avg_z_vel(){
				double sum=0;
				for(int i=0; i<particle_sample.size();i++){
					sum += particle_sample[i].z_vel;
				}
				return sum/particle_sample.size();
			}//End of function avg_z_vel

			//Function for the approximation of the average density
			double avg_density(){
				double sum=0;
				for(int i=0; i<particle_sample.size();i++){
					sum += particle_sample[i].density;
				}
				return sum/particle_sample.size();
			}//End of function avg_density

			//Function for the approximation of the average pressure
			double avg_pressure(){
```

```
                        double sum=0;
                        for(int i=0; i<particle_sample.size();i++){
                              sum += particle_sample[i].pressure;
                        }
                        return sum/particle_sample.size();
                  }//End of function avg_pressure

                  //Function for the approximation of the average mass
                  double avg_mass(){
                        double sum=0;
                        for(int i=0; i<particle_sample.size();i++){
                              sum += particle_sample[i].mass;
                        }
                        return sum/particle_sample.size();
                  }//End of function avg_mass

                  //Function   for   the   approximation   of   the   maximum
elevation
                  void max_elevation(double &elevation, double &x){

                        //Intialising elevation
                        elevation=0;

                        //Looping over all water particles inside the
                        //sampling volume. If a higher elevation is
                        //found, the elev variable is updated.
                        for(int i=0; i<particle_sample.size();i++){
                              if (particle_sample[i].z_pos>elevation){
                                    elevation = particle_sample[i].z_pos;
                                    x=particle_sample[i].x_pos;
                              }
                        }
                  }//End of function max_elevation

                  int particle_count(){
                        return particle_sample.size();
                  }

      };//End of class SamplingVolume

}

#endif
```

A.1.4 Main Program File 'data_analysis.cpp'

```
#ifndef DATA_ANALYSIS
#define DATA_ANALYSIS

#include "particle.h"
#include "sampling_volume.h"

#include <math.h>
#include <string.h>
#include <vector>
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include <cstdlib>

namespace post_processing {

	using namespace std;

	void sampleParticle(int np,int nb ,char* filename,
			    vector <SamplingVolume> &sampling_volumes,
			    double x_min, double x_max){

		//Array containing the inital sample of particle
		vector <Particle> sampled_particles;

		//Current particle being sampled
		Particle currentParticle;

		//Single line of the PART file
		string line;
		char *charLine;
		//Position of the current particle
		double x_position;

		//Opening the PART file for reading
		ifstream infile;
		infile.open(filename);

		//Looping over all lines of the PART file
		int nn=0;
		while (nn<np)
		{
			//Making sure that the end of the PART file
			//is not reached
			if (!infile.eof()){

			  //Reading a line
			  getline(infile, line);
			  //Adding a dummy token at the end of the string
			  //in order to avoid an error with the atof function
			  //when an empty line is read
			   line =line + " 1";
			   //If not an empty line
```

```
                    if (!(line==" 1")){
                       //Only sampling water particles and not
                       //boundary partilces
                       if (nn>nb){
                          //Decomposing the line into an array of
                          //characters
                          charLine = new char[line.length()];
                          for(unsigned int i=0; i< line.length(); i++){
                             charLine[i] = line[i];
                          }

                          //Determination of the fluid particle position

                          x_position = atof(strtok (charLine," "));

                          //Checking if the particle is within the
                          //outer limit of the sample area defined
                          if((x_min<x_position)&&(x_position<x_max)){

                          currentParticle.x_pos = x_position;
                          currentParticle.z_pos = atof(strtok (NULL," "));
                          currentParticle.x_vel = atof(strtok (NULL," "));
                          currentParticle.z_vel = atof(strtok (NULL," "));
                          currentParticle.density = atof(strtok (NULL," "));
                          currentParticle.pressure   =   atof(strtok   (NULL,"
"));
                          currentParticle.mass = atof(strtok (NULL," "));

                          sampled_particles.push_back(currentParticle);
                          }

                          //Deleting pointer
                          delete[] charLine;
                          charLine=0;
                          }//End of if water particle
                       }//end of if not empty line
                    }//end of if not end of file
                    nn++;
                }//End of loop over all particles

                //Closing the PART file
                infile.close();

                //Defining the focus region limits
                double vol_x_min;
                double vol_x_max;
                double vol_z_min;
                double vol_z_max;

                //Looping over all the "vertical alignments" of
                //sampling volumes
                for (int i=0 ; i < sampling_volumes.size(); i++){

                      //Calculating the values of the limits of focus region
                      vol_x_min = sampling_volumes[i].center_x -
                                  sampling_volumes[i].x_dim/2;
```

```
                  vol_x_max = sampling_volumes[i].center_x +
                              sampling_volumes[i].x_dim/2;
                  vol_z_min = sampling_volumes[i].center_z -
                              sampling_volumes[i].z_dim/2;
                  vol_z_max = sampling_volumes[i].center_z +
                              sampling_volumes[i].z_dim/2;

                  //Looping over sampled particles
                  for (int j=0 ; j < sampled_particles.size(); j++){
                        currentParticle = sampled_particles[j];

                  //Checking if the fluid particle is inside
                  //the current sampling volume

                        if (((vol_x_min<currentParticle.x_pos)&&
                              (currentParticle.x_pos<vol_x_max))&&
                              ((vol_z_min<currentParticle.z_pos)&&
                              (currentParticle.z_pos<vol_z_max))) {



      sampling_volumes[i].particle_sample.push_back
                                    (currentParticle);
                        }
                  }//End of looping over sampled particles
            }//End of looping over vertical alignments
      }//End of sampleParticle function

      // Function to calculate the vertical pressure distribution
      void printVerticalPressureDistribution(vector <SamplingVolume>
                                    &sampling_volumes,               vector
<vector<string>>
                              &pressure_series, int part_index,
                                    int num_part, string fileLabel) {

            double pressure;
            double elevation;
            int particleCount;

            string pressureString;
            string elevationString;
            string particleCountString;
            string outputLine;

            for (int i=0; i< sampling_volumes.size(); i++){

                  pressure = sampling_volumes[i].avg_pressure();
                  elevation = sampling_volumes[i].center_z;
                  particleCount=sampling_volumes[i].particle_count();

                  //Converting the maximum elevation into a string
                  ostringstream ee;
                  ee << elevation;
                  elevationString = ee.str();

                  //Converting the pressure into a string
                  ostringstream pp;
```

```
                  pp << pressure;
                  pressureString = pp.str();

                  ostringstream cc;
                  cc << particleCount;
                  particleCountString = cc.str();

                  if (part_index == 1){
                        vector <string> temp;
                        temp.push_back(pressureString);
                        pressure_series.push_back(temp);
                  }

                  else{
                        pressure_series[i].push_back(pressureString);
                  }

            }

            //If all the PART files have been processed,
            //the final output file is written
            if (part_index==num_part){

                  //Creating the output file containg the time-history
                  char* result_filename;
                  result_filename=0;
                  string result_filenameTemp;

                  result_filenameTemp = "Arithmetic_Single_Column
                                          _Pressure_series_"
                                          + fileLabel + ".txt";
                  result_filename                    =                    new
char[result_filenameTemp.length()+1];
                  strcpy(result_filename,result_filenameTemp.c_str());
                  ofstream outfile(result_filename);

                  //Writing the entire time-history to the output file
                  for (int i=0; i<num_part;i++){
                        outputLine ="";
                        for (int j =0; j<pressure_series.size();j++){
                              outputLine += (pressure_series[j])[i] + " ";
                        }
                        outputLine += "\n";
                        outfile << outputLine;
                  }

                  //Closing the output file
                  outfile.close();

                  //Deleting pointer
                  delete [] result_filename;
                  result_filename=0;
            }// End of looping over all PART files
      }// End of printVerticalPressureDistribution function


      // Function to calculate the velocity
```

```
      void printVelocityDistribution(vector <SamplingVolume>
                                          &sampling_volumes,
                                    vector<vector<string>>
                                    &velocity_series,int part_index,
                                          int  num_part,  string  fileLabel)
{

            double x_vel;
            double elevation;
            int particleCount;

            string x_velString;
            string elevationString;
            string particleCountString;
            string outputLine;

            for (int i=0; i< sampling_volumes.size(); i++){

                  x_vel = sampling_volumes[i].avg_x_vel();
                  elevation = sampling_volumes[i].center_z;
                  particleCount=sampling_volumes[i].particle_count();

                  ostringstream ee;
                  ee << elevation;
                  elevationString = ee.str();

                  ostringstream pp;
                  pp << x_vel;
                  x_velString = pp.str();

                  ostringstream cc;
                  cc << particleCount;
                  particleCountString = cc.str();

                  if (part_index == 1){
                        vector <string> temp;
                        temp.push_back(x_velString);
                        velocity_series.push_back(temp);
                  }
                  else{
                        velocity_series[i].push_back(x_velString);
                  }

            }

            if (part_index==num_part){

                  char* result_filename;
                  result_filename=0;
                  string result_filenameTemp;

                  result_filenameTemp = "Arithmetic_Single_Column
                                        _Velocity_series_" + fileLabel
                                        + ".txt";
                  result_filename                  =                  new
char[result_filenameTemp.length()+1];
                  strcpy(result_filename,result_filenameTemp.c_str());
```

```
            ofstream outfile(result_filename);

            for (int i=0; i<num_part;i++){
                  outputLine ="";
                  for (int j =0; j<velocity_series.size();j++){
                        outputLine += (velocity_series[j])[i] + " ";
                  }
                  outputLine += "\n";
                  outfile << outputLine;
            }

            outfile.close();

            delete [] result_filename;
            result_filename=0;
      }

}// End of printVelocityDistribution function

// Function to calculate the Maximum Elevation "virtual wave gauge"
void printMaxZSeries(vector <SamplingVolume> &sampling_volumes,
                           vector<vector<vector<string>>>
                                 &elevation_series, int part_index,
                           int num_part, string fileLabel) {

      double elevation;
      double x_pos;
      int particleCount;

      string elevationString;
      string x_pos_String;
      string particleCountString;
      string outputLine;

      for (int i=0; i< sampling_volumes.size(); i++){

            sampling_volumes[i].max_elevation(elevation,x_pos);
            particleCount=sampling_volumes[i].particle_count();

            ostringstream ee;
            ee << elevation;
            elevationString = ee.str();

            ostringstream xx;
            xx << x_pos;
            x_pos_String = xx.str();

            ostringstream cc;
            cc << particleCount;
            particleCountString = cc.str();

            if (part_index == 1){
                  vector <string> temp;
                  vector < vector < string > > temp2;
                  temp.push_back(elevationString);
                  temp.push_back(x_pos_String);
```

```
                        temp.push_back(particleCountString);
                        temp2.push_back(temp);
                        elevation_series.push_back(temp2);
                  }
                  else{
                        vector <string> temp;
                        temp.push_back(elevationString);
                        temp.push_back(x_pos_String);
                        temp.push_back(particleCountString);
                        elevation_series[i].push_back(temp);

                  }

            }

            if (part_index==num_part ){

                  char* result_filename;
                  result_filename=0;
                  string result_filenameTemp;

                  result_filenameTemp = "Arithmetic_Single_Column
                                        _Max_elevation_series_"
                                        + fileLabel + ".txt";
                  result_filename                  =                  new
char[result_filenameTemp.length()+1];
                  strcpy(result_filename,result_filenameTemp.c_str());

                  ofstream outfile(result_filename);

                  for (int i=0; i<num_part;i++){
                        outputLine ="";
                        for (int j =0; j<elevation_series.size();j++){
                              for (int k =0; k<4;k++){
                                    outputLine +=
                                          ((elevation_series[j])[i])[k]   +
" ";
                              }
                        }
                        outputLine += "\n";
                        outfile << outputLine;
                  }

                  outfile.close();

                  delete [] result_filename;
                  result_filename=0;
            }

      }// End of printMaxZSeries function

}


      using namespace post_processing;

      // Main program
```

```
	int main( int argc, char** argv ) {

	char* in_filename;
	string in_filenameTemp;

	string indexString;
	vector < vector < string > > pressure_series;
	vector < vector < string > > velocity_series;
	vector < vector < vector < string > > > elevation_series;

	/*****************************************************************
**
	Input parameters
	*****************************************************************
**/

	//File name for reading in parameter inputs
	ifstream infile;
	infile.open("Arithmetic_Single_Column_Parameters.txt");

	int num_part; //Number of PART files to be analysed
	int np; //Number of particles in the simulation
	int nb; //Number of boundary particles in the simulation
	double ini_center_x; //Initial x-coordinate at sampling center
	double center_x;
	double z_min; //Minimum z-coordinate of sampling column
	double z_max; //Maximum z-coordinate of sampling column
	double x_dim; //Width of sampling area along x-axis
	double z_dim; //Width of sampling area along z-axis
	int numIterations; //Number of columns to evaluate
	double  increment;  //Incremental  distance  along  x-axis  between
columns
	string variableCoordinate; // Has to be equal to CX since in 2D
	string fileLabel;

	//Property to analyze for (PRESSURE, VELOCITY, Z_POS)
	string propertyToAnalyze;

	//Reading in the input
	infile >> num_part;
	infile >> np;
	infile >> nb;
	infile >> numIterations;
	infile >> variableCoordinate;
	infile >> increment;
	infile >> ini_center_x;
	infile >> z_min;
	infile >> z_max;
	infile >> x_dim;
	infile >> z_dim;
	infile >> propertyToAnalyze;

	//Closing the input file
	infile.close();

	// Output the read parameters to the screen
	cout << num_part <<" Numberof PART files to analyze" << "\n";
```

```
	cout << np<<" Total number of boundary particles" << "\n";
	cout << nb<<" Number of boundary particles" << "\n";
	cout  <<  numIterations<<"  Number  of  analysis  to  perform  with
different                                   location  along  the  specified
direction"  <<                                    "\n";
	cout << variableCoordinate<<" Direction in which the analysises are
                                        performed (CX or CY)"  << "\n";
	cout << increment <<" Distance increment between analysises" <<
                                  "\n";
	cout << ini_center_x <<" Initial center of sampling volumes in the x
                                    direction" << "\n";
	cout << z_min << " Minimum elevation of lowest sampling volumes" <<
                                  "\n";
	cout << z_max << " Maximum elevation of highest sampling volumes" <<
                                  "\n";
	cout << x_dim <<" X dimension of sampling volumes" << "\n";
	cout << z_dim <<" Z dimension of sampling volumes" << "\n";
	cout  <<  propertyToAnalyze  <<"  Property  to  analyze  in  sampling
volumes                                         (PRESSURE,   VELOCITY   or
Z_POS = Max                                                 Elevation)"  <<
"\n";

//*********************************************************************

	for (int j = 0;  j < numIterations; j++){

		//Checking how many columns are to be evaluated and obtain
			//center of column along x-axis accordingly
		if (variableCoordinate == "CX"){

			center_x = ini_center_x + j * increment;
			ostringstream cx;
			cx << center_x;
			fileLabel = "Cx=" + cx.str();
		}

		//Looping over PART files
		for (int i=1; i<= num_part; i++){

			vector <SamplingVolume> sampling_volumes;

			for (double z=z_min; z<z_max; z=z+z_dim){

				SamplingVolume samplingVolume;

				samplingVolume.center_x =center_x;
				samplingVolume.center_z = z+z_dim/2;
				samplingVolume.x_dim = x_dim;
				samplingVolume.z_dim = z_dim;

				sampling_volumes.push_back(samplingVolume);

			}


	//*****************************************************
			//Generation of the string corresponding to the current
```

```
                //PART file to be processed

      //*******************************************************
                ostringstream ii;
                ii << i;
                indexString = ii.str();

                if (i <10){
                      in_filenameTemp = "PART_000" +indexString;
                      in_filename  =  new  char[in_filenameTemp.length()  +
1];
                      strcpy(in_filename,in_filenameTemp.c_str());

                }
                else if (i <100){

                      in_filenameTemp = "PART_00" + indexString;
                      in_filename  =  new  char[in_filenameTemp.length()  +
1];
                      strcpy(in_filename,in_filenameTemp.c_str());

                }
                else if (i <1000){
                      in_filenameTemp = "PART_0" +indexString;
                      in_filename  =  new  char[in_filenameTemp.length()  +
1];
                      strcpy(in_filename,in_filenameTemp.c_str());

                }
                else{
                      in_filenameTemp = "PART_" +indexString;
                      in_filename  =  new  char[in_filenameTemp.length()  +
1];
                      strcpy(in_filename,in_filenameTemp.c_str());

                }

                //Calling the function that will search the current PART
                //file and construct the list of particle located inside
                //the sampling volume
                sampleParticle(np,nb, in_filename, sampling_volumes,
                              center_x - x_dim/2, center_x +
                                    x_dim/2);

                //Checking which property was analysed and

                //calling the function that will write the results to
                //the output file
                if (propertyToAnalyze == "PRESSURE" ){

      printVerticalPressureDistribution(sampling_volumes,
            pressure_series, i, num_part,fileLabel);
                }

                if (propertyToAnalyze == "X_VELOCITY" ){
                      printVelocityDistribution(sampling_volumes,
                            velocity_series, i, num_part, fileLabel);
```

```
                  }

                  if (propertyToAnalyze == "Z_POS" ){
                        printMaxZSeries(sampling_volumes,
elevation_series,                                    i, num_part, fileLabel);
                  }

                  //Progress feedback on screen
                  cout << "Analysing file " << in_filenameTemp << " for "
<<                          fileLabel<< "\n";

                  //Clearing the sampled particles
                  sampling_volumes.clear();

                  //Deleting pointer
                  delete [] in_filename;
                  in_filename=0;
            }

            //Emptying the vectors
            pressure_series.clear();
            velocity_series.clear();
            elevation_series.clear();

      }//End of looping over PART files
      return 0;
}//End of main program

#endif
```

A.2 Extraction Code for Sloping Wall

As described in section 5-2, for the case of a sloping seawall, the extraction code needs to be modified slightly in order to account for the angle the front face of the seawall makes with the horizontal. This was achieved using the axis transformation equations to rotate the sampling areas. This version of the extraction code source also has three main components that can be found in sections A.2.2, A.2.3 and A.2.4.

A.2.1 Description of Input File

The input file required for the sloping wall is the same as that described in Appendix A1 except for a few changes that can be seen in Figure A-2 below.

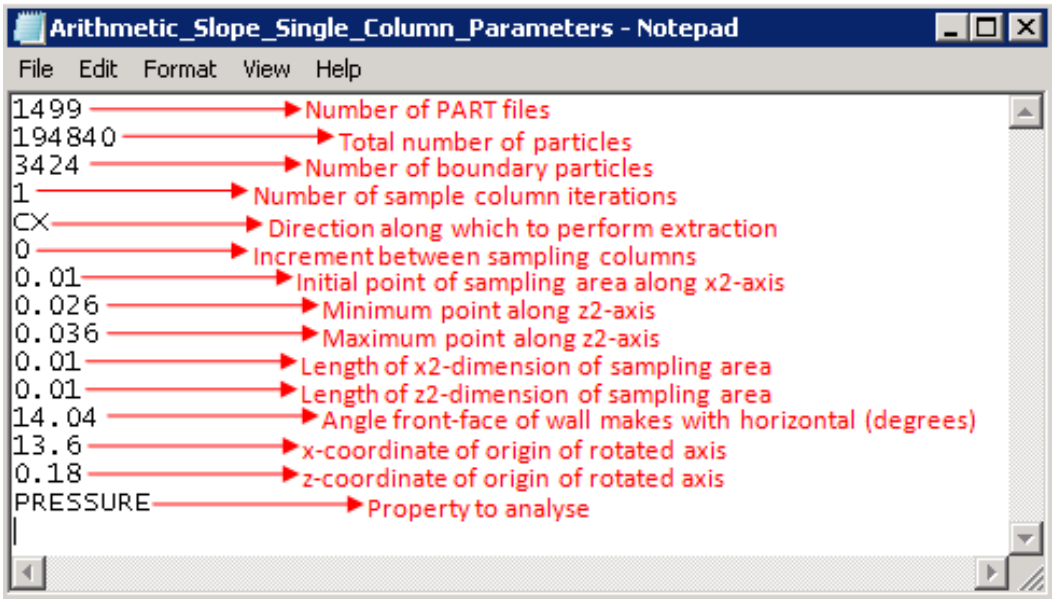

Figure A-0-2 Sample input file for extraction code on sloping structure

In addition to the twelve previous input parameters, there are three more: the angle that the structure makes with the horizontal in degrees and the x- and z-coordinate of the location of the origin [x_2=0, z_2=0] of the secondary rotated axis. The dimensions of the sampling areas are now defined with respect to the rotated axes rather than the simulation axes. Moreover, a particle counter at each PART file was included in order to provide an estimate of the overtopping rate.

A.2.2 Header File 'particle.h'

```
#ifndef PARTICLE_H
#define PARTICLE_H

namespace post_processing {

	class Particle {
		public:
			//Attributes associated to a water particle
			double x_pos;
			double z_pos;
			double x_pos_2;
			double z_pos_2;
			double x_vel;
			double z_vel;
			double density;
			double pressure;
			double mass;
	};//End of class Particle

}

#endif
```

A.2.3 Header File 'sampling_volume.h'

```
#ifndef SAMPLING_VOLUME_H
#define SAMPLING_VOLUME_H

#include "particle.h"

#include <vector>

namespace post_processing {

	class SamplingVolume {

		public:
			//Array of particles inside the sampling volume
			std::vector <Particle> particle_sample;

			//Center coordinates of sampling volume
			double center_x;
			double center_z;

			double x2_dim;
			double z2_dim;

			//Function for the approximation of the average x-
velocity
			double avg_x_vel(){
				double sum=0;
				for(int i=0; i<particle_sample.size();i++){
					sum += particle_sample[i].x_vel;
				}
				return sum/particle_sample.size();
			}//End of function avg_x_vel

			//Function for the approximation of the average z-
velocity
			double avg_z_vel(){
				double sum=0;
				for(int i=0; i<particle_sample.size();i++){
					sum += particle_sample[i].z_vel;
				}
				return sum/particle_sample.size();
			}//End of function avg_z_vel

			//Function for the approximation of the average density
			double avg_density(){
				double sum=0;
				for(int i=0; i<particle_sample.size();i++){
					sum += particle_sample[i].density;
				}
				return sum/particle_sample.size();
			}//End of function avg_density

			//Function for the approximation of the average pressure
			double avg_pressure(){
				double sum=0;
```

```
                        for(int i=0; i<particle_sample.size();i++){
                              sum += particle_sample[i].pressure;
                        }
                        return sum/particle_sample.size();
                  }//End of function avg_pressure

                  //Function for the approximation of the average mass
                  double avg_mass(){
                        double sum=0;
                        for(int i=0; i<particle_sample.size();i++){
                              sum += particle_sample[i].mass;
                        }
                        return sum/particle_sample.size();
                  }//End of function avg_mass

                  //Function   for   the   approximation   of   the   maximum
elevation
                  void max_elevation(double &elevation, double &x){

                        //Intialising elevation
                        elevation=0;

                        //Looping over all water particles inside the
                        //sampling volume. If a higher elevation is
                        //found, the elev variable is updated.
                        for(int i=0; i<particle_sample.size();i++){
                              if (particle_sample[i].z_pos>elevation){
                                    elevation = particle_sample[i].z_pos;
                                    x=particle_sample[i].x_pos;
                              }
                        }
                  }//End of function max_elevation

                  int particle_count(){
                        return particle_sample.size();
                  }

      };//End of class SamplingVolume
}

#endif
```

A.2.4 Main Program File 'data_analysis.cpp'

```
#ifndef DATA_ANALYSIS
#define DATA_ANALYSIS

#include "particle.h"
#include "sampling_volume.h"

#include <math.h>
#include <string.h>
#include <vector>
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include <cstdlib>

const double PI = 3.14159265359;

namespace post_processing {

      using namespace std;

      void sampleParticle(int np,int nb ,char* filename,
                          vector<SamplingVolume> &sampling_volumes,
                          double    alpha,    double    origin_x2,    double
origin_z2,                                  double    z2_min,    double    z2_max,
double*                                                 num_over_par){

            //Array containing the inital sample of particle
            vector <Particle> sampled_particles;

            //Array containing the sampled particles in the focus region
            vector <Particle> sampled_particles_region;

            //Current particle being sampled
            Particle currentParticle;

            //Single line of the PART file
            string line;
            char *charLine;

            //Position of the current particle
            double x_position;

            //Opening the PART file for reading
            ifstream infile;
            infile.open(filename);

            //Looping over all lines of the PART file
            int nn=0;
            while (nn<np)
            {
               //Making sure that the end of the PART file
               //is not reached
               if (!infile.eof()){
```

```
                //Reading a line
                getline(infile, line);

                //Adding a dummy token at the end of the string
                //in order to avoid an error with the atof function
                //when an empty line is read
                line =line + " 1";
                //If not an empty line
                if (!(line==" 1")){

                   //Only sampling water particles and not
                   //boundary partilces
                   if (nn>nb){
                      //Decomposing the line into an array of
                      //characters
                      charLine = new char[line.length()];
                      for (unsigned int i=0; i< line.length(); i++){
                                        charLine[i] = line[i];
                   }

                   //Defining limits of focused region that
                         //contains sampling areas
                   double region_min_z;
                   double region_max_z;
                   double region_min_x;
                   double region_max_x;
                   double region_vert;
                   double region_horz;

                   //Vertical and horizontal limits of focus
                         //region (adding 50% to make region larger):
                               region_vert =

z2_max*sin(alpha*PI/180)*(1+0.50);
                   region_horz =
                                     z2_max*cos(alpha*PI/180)*(1+0.50);
                   region_min_x = origin_x2 - region_horz;
                   region_max_x = origin_x2 + region_horz;
                   region_min_z = origin_z2;
                   region_max_z = origin_z2 + region_vert;

                   //Extract particles in focused region
                   //and convert coordinates to imaginary axis
                   x_position = atof(strtok (charLine," "));

                   //To count the number of particles that overtop
                         //in the current PART
                   if (x_position > 13.9){
                            *(num_over_par) += 1;
                   }

                   if ((region_min_x < x_position)&&
                       (x_position<region_max_x)){

                      currentParticle.x_pos = x_position;
                      currentParticle.z_pos = atof(strtok(NULL," "));
```

```
                        currentParticle.x_vel = atof(strtok(NULL," "));
                        currentParticle.z_vel = atof(strtok(NULL," "));
                        currentParticle.density = atof(strtok(NULL," "));
                        currentParticle.pressure = atof(strtok(NULL," "));
                        currentParticle.mass = atof(strtok(NULL," "));

                        //Converting x and z to x2 and z2:
                        currentParticle.x_pos_2 = -(currentParticle.x_pos
                                                -          origin_x2)          *
sin(alpha*PI/180)
                                                + (currentParticle.z_pos
                                                -                   origin_z2)*
cos(alpha*PI/180);

                        currentParticle.z_pos_2 =  (currentParticle.x_pos
                                                -          origin_x2)          *
cos(alpha*PI/180)
                                                + (currentParticle.z_pos
                                                -          origin_z2)          *
sin(alpha*PI/180);


      sampled_particles_region.push_back(currentParticle);
                    }

                    //Deleting pointer
                    delete[] charLine;
                    charLine=0;
                   }//End of if water particle
                  }//End of if not empty line
                 }//End of if not end of file
                 nn++;
           }//End of loop over all particles

           //Closing the PART file
           infile.close();

           //Defining the focus region limits
           double vol_x2_min;
           double vol_x2_max;
           double vol_z2_min;
           double vol_z2_max;

           //Looping over all the "vertical alignments" of
           //sampling volumes
           for (int i=0 ; i < sampling_volumes.size(); i++){

                 //Calculating the values of the limits of focus region
                 vol_x_min = sampling_volumes[i].center_x -
                             sampling_volumes[i].x_dim/2;
                 vol_x_max = sampling_volumes[i].center_x +
                             sampling_volumes[i].x_dim/2;
                 vol_z_min = sampling_volumes[i].center_z -
                             sampling_volumes[i].z_dim/2;
                 vol_z_max = sampling_volumes[i].center_z +
                             sampling_volumes[i].z_dim/2;
```

```
                  //Looping over sampled particles
                  for (int j=0 ; j < sampled_particles.size(); j++){
                        currentParticle = sampled_particles[j];

                        //Checking if the fluid particle is inside
                  //the current sampling volume

                        if (((vol_x_min<currentParticle.x_pos)&&
                              (currentParticle.x_pos<vol_x_max))&&
                              ((vol_z_min<currentParticle.z_pos)&&
                              (currentParticle.z_pos<vol_z_max))) {



      sampling_volumes[i].particle_sample.push_back
                                    (currentParticle);
                        }
                  }//End of looping over sampled particles
            }//End of looping over vertical alignments
      }//End of sampleParticle function

      // Function to calculate the sloped pressure distribution
      void printSlopedPressureDistribution(vector <SamplingVolume>
                                    &sampling_volumes,
vector<vector<string>>
                              &pressure_series,int     part_index,     int
num_part,
                              string           fileLabel,           double
overtop_par_array[],
                              int curr_num_part){

            double pressure;
            double elevation;
            int particleCount;

            string pressureString;
            string elevationString;
            string particleCountString;
            string outputLine;
            string outputLine2;

            for (int i=0; i< sampling_volumes.size(); i++){

                  pressure = sampling_volumes[i].avg_pressure();
                  elevation = sampling_volumes[i].center_z;
                  particleCount=sampling_volumes[i].particle_count();

                  //Converting the maximum elevation into a string
                  ostringstream ee;
                  ee << elevation;
                  elevationString = ee.str();

                  //Converting the pressure into a string
                  ostringstream pp;
                  pp << pressure;
                  pressureString = pp.str();
```

```
                ostringstream cc;
                cc << particleCount;
                particleCountString = cc.str();

                if (part_index == 1){
                      vector <string> temp;
                      temp.push_back(pressureString);
                      pressure_series.push_back(temp);
                }
                else{
                      pressure_series[i].push_back(pressureString);
                }

          }

          //If all the PART files have been processed,
          //the final output file is written
          if (part_index==num_part){

                //Creating the output file containg the time-history
                char* result_filename;
                result_filename=0;
                string result_filenameTemp;

                result_filenameTemp = "Arithmetic_Slope_Single_Column_
                                       Pressure_series_"
                                       + fileLabel + ".txt";
                result_filename                    =                    new
char[result_filenameTemp.length()+1];
                strcpy(result_filename,result_filenameTemp.c_str());
                ofstream outfile(result_filename);

                //Writing the entire time-history to the output file
                for (int i=0; i<num_part;i++){
                      outputLine ="";
                      for (int j =0; j<pressure_series.size();j++){
                            outputLine += (pressure_series[j])[i] + " ";
                      }
                      outputLine += "\n";
                      outfile << outputLine;
                }

                //Closing the output file
                outfile.close();

                //Output file to list all the overtopping number
                      //of particles per PART file
                ofstream outfile2("Number_Overtopped_Particles
                                       _per_PART.txt");

                //Convert overtop_par_array contents into a string
                //to fill the output text file
                ostringstream overtop;

                for (int n=0; n<num_part;n++){

                      outfile2 <<  overtop_par_array[n] << endl;
```

```
                }

                //Closing output file 2
                outfile2.close();

                //Deleting pointer
                delete [] result_filename;
                result_filename=0;

          }// End of looping over all PART files
    }// End of printSlopedPressureDistribution function

    // Function to calculate the velocity
    void printVelocityDistribution(vector <SamplingVolume>
                                        &sampling_volumes,
                                   vector<vector<string>>
                                   &velocity_series,int part_index,
                                        int  num_part,  string  fileLabel)
{

          double x_vel;
          double elevation;
          int particleCount;

          string x_velString;
          string elevationString;
          string particleCountString;
          string outputLine;

          for (int i=0; i< sampling_volumes.size(); i++){

                x_vel = sampling_volumes[i].avg_x_vel();
                elevation = sampling_volumes[i].center_z;
                particleCount=sampling_volumes[i].particle_count();

                ostringstream ee;
                ee << elevation;
                elevationString = ee.str();

                ostringstream pp;
                pp << x_vel;
                x_velString = pp.str();

                ostringstream cc;
                cc << particleCount;
                particleCountString = cc.str();

                if (part_index == 1){
                      vector <string> temp;
                      temp.push_back(x_velString);
                      velocity_series.push_back(temp);
                }
                else{
                      velocity_series[i].push_back(x_velString);
                }

          }
```

```
            if (part_index==num_part){

                  char* result_filename;
                  result_filename=0;
                  string result_filenameTemp;

                  result_filenameTemp = "Arithmetic_Single_Column
                                         _Velocity_series_" + fileLabel
                                         + ".txt";
                  result_filename                    =                    new
char[result_filenameTemp.length()+1];
                  strcpy(result_filename,result_filenameTemp.c_str());

                  ofstream outfile(result_filename);

                  for (int i=0; i<num_part;i++){
                        outputLine ="";
                        for (int j =0; j<velocity_series.size();j++){
                              outputLine += (velocity_series[j])[i] + " ";
                        }
                        outputLine += "\n";
                        outfile << outputLine;
                  }

                  outfile.close();

                  delete [] result_filename;
                  result_filename=0;
            }

      }// End of printVelocityDistribution function

      // Function to calculate the Maximum Elevation "virtual wave gauge"
      void printMaxZSeries(vector <SamplingVolume> &sampling_volumes,
                           vector<vector<vector<string>>>
                                 &elevation_series, int part_index,
                           int num_part, string fileLabel) {

            double elevation;
            double x_pos;
            int particleCount;

            string elevationString;
            string x_pos_String;
            string particleCountString;
            string outputLine;

            for (int i=0; i< sampling_volumes.size(); i++){

                  sampling_volumes[i].max_elevation(elevation,x_pos);
                  particleCount=sampling_volumes[i].particle_count();

                  ostringstream ee;
                  ee << elevation;
                  elevationString = ee.str();
```

```
                ostringstream xx;
                xx << x_pos;
                x_pos_String = xx.str();

                ostringstream cc;
                cc << particleCount;
                particleCountString = cc.str();

                if (part_index == 1){
                      vector <string> temp;
                      vector < vector < string > > temp2;
                      temp.push_back(elevationString);
                      temp.push_back(x_pos_String);
                      temp.push_back(particleCountString);
                      temp2.push_back(temp);
                      elevation_series.push_back(temp2);
                }
                else{
                      vector <string> temp;
                      temp.push_back(elevationString);
                      temp.push_back(x_pos_String);
                      temp.push_back(particleCountString);
                      elevation_series[i].push_back(temp);

                }

          }

          if (part_index==num_part ){

                char* result_filename;
                result_filename=0;
                string result_filenameTemp;

                result_filenameTemp = "Arithmetic_Single_Column
                                      _Max_elevation_series_"
                                      + fileLabel + ".txt";
                result_filename                =                  new
char[result_filenameTemp.length()+1];
                strcpy(result_filename,result_filenameTemp.c_str());

                ofstream outfile(result_filename);

                for (int i=0; i<num_part;i++){
                      outputLine ="";
                      for (int j =0; j<elevation_series.size();j++){
                            for (int k =0; k<4;k++){
                                  outputLine +=
                                        ((elevation_series[j])[i])[k]   +
" ";
                            }
                      }
                      outputLine += "\n";
                      outfile << outputLine;
                }

                outfile.close();
```

```
                    delete [] result_filename;
                    result_filename=0;
                }

        }// End of printMaxZSeries function
}

using namespace post_processing;

// Main program
int main( int argc, char** argv ) {

char* in_filename;
string in_filenameTemp;

string indexString;
vector < vector < string > > pressure_series;
vector < vector < string > > velocity_series;
vector < vector < vector < string > > > elevation_series;

/*********************************************************************
Input parameters
*********************************************************************/

//File name for reading in parameter inputs
ifstream infile;
infile.open("Arithmetic_Slope_Single_Column_Parameters.txt");

int num_part; //Number of PART files to be analysed
int np; //Number of particles in the simulation
int nb; //Number of boundary particles in the simulation
double ini_center_x2; //Initial x2-coordinate at sampling centre
double center_x;
double z2_min; //Lower limit of sampling column along z2-axis
double z2_max; //Upper limit of sampling column along z2-axis
double x2_dim; //Dimension of sampling area on rotated access along x2
double z2_dim; //Dimension of sampling area on rotated access along z2
double alpha; //Angle the rotated axis makes with the horizontal
double origin_x2; //Rotated access origin x-coordinate
double origin_z2; //Rotated access origin z-coordinate
double num_over_par = 0 ; //To count the number of overtopping particles
int numIterations; //Number of columns to evaluate
double increment; //Incremental distance along x-axis between columns
string variableCoordinate; // Has to be equal to CX since in 2D
string fileLabel;

//Property to analyze for (PRESSURE, VELOCITY, Z_POS)
string propertyToAnalyze;

infile >> num_part;
infile >> np;
infile >> nb;
infile >> numIterations;
infile >> variableCoordinate;
infile >> increment;
infile >> ini_center_x2;
```

```
infile >> z2_min;
infile >> z2_max;
infile >> x2_dim;
infile >> z2_dim;
infile >> alpha;
infile >> origin_x2;
infile >> origin_z2;
infile >> propertyToAnalyze;

//Closing the input file
infile.close();


// Output the read parameters to the screen
cout << num_part <<"    Numberof PART files to analyze" << "\n";
cout << np<<"    Total number of boundary particles" << "\n";
cout << nb<<"    Number of boundary particles" << "\n";
cout << numIterations<<"     Number of analyses to perform with different
                              location along the specified direction"   <<
"\n";
cout << variableCoordinate<<"    Direction in which the analyses are
                                      performed (CX or CY)"  << "\n";
cout << increment <<"    Distance increment between analyses" << "\n";
cout << ini_center_x2 <<"     Initial center of sampling volumes in the x2
                                 direction on imaginary axis" << "\n";
cout << z2_min << "   Minimum elevation of lowest sampling volumes on
                              rotated axis" << "\n";
cout << z2_max << "   Maximum elevation of highest sampling volumes on
                              rotated axis" << "\n";
cout << x2_dim << "    X dimension of sampling volumes on rotated axis" <<
                              "\n";
cout << z2_dim << "    Z dimension of sampling volumes on rotated axis" <<
                              "\n";
cout << alpha << "     Slope angle of sampling area with horizontal" <<
"\n";
cout << origin_x2 << "    X coordinate of origin of rotated axis" << "\n";
cout << origin_z2 << "    Z coordinate of origin of rotated axis" << "\n";
cout << propertyToAnalyze <<"    Property to analyze in sampling volumes
                                 (PRESSURE,  VELOCITY  or  Z_POS  =  Max
Elevation)"                                    << "\n";

//*********************************************************************

      for (int j = 0;  j < numIterations; j++){

            //Checking how many columns are to be evaluated and obtain
                  //center of column along x-axis accordingly
            if (variableCoordinate == "CX"){

                  center_x = ini_center_x2 + j * increment;

                  ostringstream cx;
                  cx << center_x;

                  fileLabel = "Cx=" + cx.str();

            }
```

```
	//Looping over PART files
	for (int i=1; i<= num_part; i++){

		vector <SamplingVolume> sampling_volumes;

		for (double z=z2_min; z<z2_max; z = z+z2_dim){

			SamplingVolume samplingVolume;

			samplingVolume.center_x =center_x;
			samplingVolume.center_z = z + z2_dim/2;
			samplingVolume.x2_dim = x2_dim;
			samplingVolume.z2_dim = z2_dim;

			sampling_volumes.push_back(samplingVolume);

		}


	//*****************************************************
			//Generation of the string corresponding to the current
			//PART file to be processed

	//*****************************************************

			ostringstream ii;
			ii << i;
			indexString = ii.str();

			if (i <10){
				in_filenameTemp = "PART_000" +indexString;
				in_filename  =  new  char[in_filenameTemp.length()  +
1];
				strcpy(in_filename,in_filenameTemp.c_str());
			}
			else if (i <100){

				in_filenameTemp = "PART_00" + indexString;
				in_filename  =  new  char[in_filenameTemp.length()  +
1];
				strcpy(in_filename,in_filenameTemp.c_str());
			}
			else if (i <1000){
				in_filenameTemp = "PART_0" +indexString;
				in_filename  =  new  char[in_filenameTemp.length()  +
1];
				strcpy(in_filename,in_filenameTemp.c_str());
			}
			else{
				in_filenameTemp = "PART_" +indexString;
				in_filename  =  new  char[in_filenameTemp.length()  +
1];
				strcpy(in_filename,in_filenameTemp.c_str());
			}

			//Calling the function that will search the current PART
```

```
                    //file and construct the list of particle located inside
                    //the sampling volume
                    sampleParticle(np,nb, in_filename, sampling_volumes,
                                        alpha,  origin_x2,  origin_z2,  z2_min,
                                              z2_max,&num_over_par);

                    //Array to contain the number of particles

                    //overtopping each PART
                    int curr_num_part = i;
                    double overtop_par_array[2000];
                    overtop_par_array[i]= num_over_par;
                    //Re-initialize counter before loop goes to next PART
                    num_over_par = 0;

                    //Checking which property was analysed and

                    //calling the function that will write the results to
                    //the output file
                    if (propertyToAnalyze == "PRESSURE" ){
                          printSlopedPressureDistribution(sampling_volumes,
                                pressure_series, i, num_part,fileLabel,
                                      overtop_par_array, curr_num_part);
                    }

                    if (propertyToAnalyze == "X_VELOCITY" ){
                          printVelocityDistribution(sampling_volumes,
                                velocity_series, i, num_part, fileLabel);
                    }

                    if (propertyToAnalyze == "Z_POS" ){
                          printMaxZSeries(sampling_volumes,
elevation_series,                                   i, num_part, fileLabel);
                    }

                    //Progress feedback
                    cout << "Analysing file " << in_filenameTemp << " for "
<<                                  fileLabel<< "\n";

                    //Clearing the sampled particles
                    sampling_volumes.clear();

                    //Deleting pointer
                    delete [] in_filename;
                    in_filename=0;
              }

              //Emptying the vectors
              pressure_series.clear();
              velocity_series.clear();
              elevation_series.clear();

        }//End of looping over PART files
        return 0;
}//End of main program

#endif
```

B. Analysis Parameter Details

B.1 Simulation Parameters for Sensitivity Analysis

Simulation Name	ζ_{smooth}	ζ_{sound}	β Limiter	dx	dz	T_{max} (s)	h (m)	C_o (m/s)	Comp. Time (hours)	[np]	[nb]
SEN Beta1	0.92	15	1.3	0.01	0.01	7	0.0130	21.011	5.50	18893	1013
SEN Beta2	0.92	15	1	0.01	0.01	7	0.0130	21.011	5.40	18893	1013
SEN Beta3	0.92	15	1.2	0.01	0.01	7	0.0130	21.011	5.50	18893	1013
SEN Beta4	0.92	15	1.6	0.01	0.01	7	0.0130	21.011	5.83	18893	1013
SEN Smooth1	0.92	15	1.3	0.01	0.01	7	0.0130	21.011	7.83	18893	1013
SEN Smooth2	0.8	15	1.3	0.01	0.01	7	0.0113	21.011	5.17	18893	1013
SEN Smooth3	0.7	15	1.3	0.01	0.01	7	0.0099	21.011	4.33	18893	1013
SEN Smooth4	0.85	15	1.3	0.01	0.01	7	0.0120	21.011	5.00	18893	1013
SEN Sound1	0.72	15	1.3	0.01	0.01	7	0.0102	21.011	4.67	18893	1013
SEN Sound2	0.72	16	1.3	0.01	0.01	7	0.0102	22.411	5.00	18893	1013
SEN Sound3	0.72	11	1.3	0.01	0.01	7	0.0102	15.408	3.75	18893	1013
SEN Sound4	0.72	20	1.3	0.01	0.01	7	0.0102	28.014	6.00	18893	1013
SEN Spacing1	0.72	16	1.25	0.01	0.01	7	0.0102	22.411	4.67	18893	1013
SEN Spacing2	0.72	16	1.25	0.05	0.05	7	0.0509	22.411	0.05	921	208
SEN Spacing3	0.72	16	1.25	0.008	0.008	7	0.0081	22.411	3.50	29214	1266
SEN Spacing4	0.72	16	1.25	0.03	0.03	7	0.0305	22.411	0.17	2260	342
SEN Spacing5	0.72	16	1.25	0.005	0.005	7	0.0051	22.411	35.00	73567	2019

B.2 Additional Simulation Parameters

Table B-1 Parameter values for low resolution numerical model of Ramsden (1993)

Parameter Name	Parameter Value
Particle Spacing [dx and dz] (m)	0.03, 0.01
Smoothing Distance [h] (m)	0.029
Smoothing Distance Coefficient	0.92
Kernel Type	Cubic
Density Filter	None
Riemann Solver	Non-conservative (Parshikov 2000)
Riemann Solver`s Slope Limiter [Beta-limiter]	1.15
Reference Speed of Sound (m/s)	32.21
'Coefficient' of Speed of sound	15
Time Stepping Method	Symplectic
Viscosity Treatment	Laminar + Sub-Particle Scale