CFD Modeling of an Oscillating Wave Surge Converter using the Overset Grid Method

Madeline Riddle

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Civil Engineering

University of Washington

2022

University of Washington

Abstract

CFD Modeling of an Oscillating Wave Surge Converter using the Overset Grid Method

Madeline Riddle

Chair of the Supervisory Committee:
Associate Professor Michael Motley
Department of Civil and Environmental Engineering

In recent decades, the push for reliable renewable energy sources has led researchers to explore an ever-expanding array of novel devices capable of harvesting such energy. Of the current available renewable energy sources, marine energy is one of the least-utilized categories with wave energy being the least common. However, given their high amount of energy extraction potential, wave energy converter (WEC) devices are gaining interest in the global energy market. One such device is a WEC known as the oscillating wave surge converter (OWSC). OWSCs are paddle-like devices which rotate about a fixed hinge when driven by wave surges (or wave motions). Most research on OWSC devices to date has primarily been based on experimental, scale model testing. Experimental tests on OWSC devices in wave tanks are expensive, time-consuming, and testing facilities are limited. To improve the quality of such tests, computational fluid dynamics (CFD) models may be used to assess and fine-tune a design before running a physical experiment.

Modeling an OWSC using available CFD software poses a number of challenges related to model geometry, mesh quality, and solution stability. The high degrees of rotation experienced by these devices make typical mesh-morphing methods untenable. Alternative methods for dealing with the large mesh deformation issues have been devised and implemented; however, many of these alternatives require modifications to the available CFD software. This work presents two CFD models which utilize an overset mesh approach to simulate an OWSC device and require no software modification. The overset mesh approach uses a background and overset (body-fitted) mesh where the overset mesh overlays the static background mesh and moves relative to it. This method avoids mesh deformation altogether and instead relies on interpolation between the two meshes. CFD models were created using the OpenFOAM and STAR-CCM+ software so that the numerical model results could be compared across platforms. These CFD models are also compared to a scale model wave tank test.

TABLE OF CONTENTS

1 INTRODUCTION 1
1.1 Motivation 1
1.2 Previous Work 4
1.3 Objectives 7

2 NUMERICAL MODELING METHODOLOGY 9
2.1 OpenFOAM Fluid Solver 9
2.2 OpenFOAM Rigid Body Motion 15
2.3 STARCCM+ 19

3 NUMERICAL MODEL 24
3.1 CFD Model Domain 24
3.2 Waves and Boundary Conditions 26
3.3 Mesh Generation 32

4 OVERSET MESH APPROACH 38
4.1 Previous Methodology 38
4.2 Overset Grid Methodology 42
4.2.1 OpenFOAM v2012 46
4.2.2 STAR-CCM+ 2021.3 49

5 RESULTS 52
5.1 OpenFOAM v2012 Results 52
5.2 OpenFOAM Overset Comparison to Results by Schmitt & Elsaesser 2015 63
5.3 STAR-CCM+ 2021.3 Results 71

6 CONCLUSIONS AND FUTURE WORK 80
6.1 Conclusions 80
6.2 Future Work 83

7 BIBLIOGRAPHY 85

A OpenFOAM Scripts: Background Mesh 92

B OpenFOAM Scripts: Overset Mesh **124**

List of Figures

2.1 STAR-CCM+ dialog window listing the fluid physics models used. 21

3.1 Wave Tank Geometry and Water Level (in mm). Figure courtesy of [5]. 25
3.2 A single hexahedral block with vertex labeling convention. Figure courtesy of [43]. 33
3.3 Mesh blocks defined in *blockMeshDict*. 34
3.4 Background mesh, generated via the ParaView [44] software. . . 36
3.5 3D wave tank model with overset mesh, generated in the ParaView [44] software. 37

4.1 Initial mesh (top) and deformed mesh (bottom) for a moving cylinder. Figures courtesy of [45]. 39
4.2 A generic background mesh. Figure courtesy of [47]. 43
4.3 Two overset meshes laid over a background mesh. Figure courtesy of [47]. 44
4.4 OWSC overset mesh generated in the ParaView [44] software. . . 45
4.5 OpenFOAM file structure for overset grid simulation. 48
4.6 OpenFOAM overset simulation set-up process. 49
4.7 Regions branch of STAR-CCM+ simulation tree. 50

5.1 OWSC rotation results from OpenFOAM. 53
5.2 One wavelength of OWSC rotation results from OpenFOAM. . . 54
5.3 OpenFOAM OWSC rotation for the following times (read left to right and top to bottom): 10.15, 10.4, 17.75, 18.0, 21.65, 21.9, 35.75, 35.95 (seconds). 55
5.4 Angular velocity of the OWSC from OpenFOAM. 56
5.5 OWSC angular velocity and rotation from OpenFOAM. 56
5.6 Angular acceleration of OWSC from OpenFOAM. 57
5.7 Close-up of OWSC angular acceleration from OpenFOAM. . . . 58
5.8 Force components on the OWSC from OpenFOAM. 58
5.9 Longitudinal (x-direction) force on the OWSC from OpenFOAM. 59
5.10 Close-up of longitudinal (x-direction) force on the OWSC from OpenFOAM. 60

5.11 Longitudinal force (x-direction) on the OWSC and rotation from OpenFOAM. 60
5.12 Moment components on the OWSC from OpenFOAM. 61
5.13 Close-up of moment about the axis of rotation on the OWSC for OpenFOAM. 62
5.14 Moment (y-direction) on the OWSC and rotation from OpenFOAM. 63
5.15 Free surface elevation changes from initial height near the OWSC. 64
5.16 Free surface elevation changes from initial height near the OWSC with phase-shifted OpenFOAM data. 65
5.17 Rotation comparison of OpenFOAM overset vs. Schmitt & Elsaesser AMI. 66
5.18 Tangential acceleration comparison of OpenFOAM overset vs. Schmitt & Elsaesser [5] accelerometer data. 68
5.19 Amplitude and behavior comparison of time-shifted OpenFOAM tangential acceleration data to Schmitt & Elsaesser [5] experiment. 69
5.20 Comparison of tangential acceleration and rotation of the OWSC in OpenFOAM. 69
5.21 Tangential acceleration results for Benites-Munoz CFD [38] and the OpenFOAM simulation. 71
5.22 Free surface elevation changes from initial height near the OWSC for STAR-CCM+ and OpenFOAM. 72
5.23 OWSC rotation comparison of OpenFOAM vs. STAR-CCM+. . 73
5.24 OWSC angular velocity comparison of OpenFOAM vs. STAR-CCM+. 74
5.25 OWSC angular acceleration comparison of OpenFOAM vs. STAR-CCM+. 74
5.26 Close-up of OWSC rotation comparison of OpenFOAM vs. STAR-CCM+. 75
5.27 Close-up of OWSC angular velocity comparison of OpenFOAM vs. STAR-CCM+. 76
5.28 Close-up of OWSC angular acceleration comparison of OpenFOAM vs. STAR-CCM+. 76
5.29 Longitudinal force on the OWSC comparison between STAR-CCM+ and OpenFOAM. 77
5.30 Moment on the OWSC comparison between STAR-CCM+ and OpenFOAM. 78
5.31 Close-up comparison of moment behavior between STAR-CCM+ and OpenFOAM. 79

List of Tables

3.1 OWSC device properties. Height measurements in reference to lowest floor. [5, 37] 25
3.2 OpenFOAM Boundary Conditions 29

6.1 Maximum and minimum OWSC rotation angles. 81

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Michael Motley, for his encouragement and guidance throughout this project, particularly in the majority online format of the last year and a half. Thank you for your support and for always giving me a nudge when I went too far down a rabbit hole. I would also like to thank Dr. Richard Wiebe and Dr. Morteza Derakhti for serving on my committee.

Thank you, in particular, to Andrew Winter, for lending his expertise on OpenFOAM, STAR-CCM+, and CFD modeling; for his continuing support throughout the entirety of this project; and for reviewing and suggesting revisions for this thesis. Thanks as well to Nicolette Lewis, for all her help with OpenFOAM and using and navigating the Hyak supercomputer system.

This work was funded by NAVFAC through the Pacific Marine Energy Center. The computational fluid dynamics modeling for this research was made possible by the Hyak supercomputer cluster developed in a partnership between the University of Washington eScience Institute and UW-IT. Thank you to both of these organizations, without which this work would not have been possible.

Chapter 1

INTRODUCTION

1.1 Motivation

In recent decades, the push for reliable renewable energy sources has led researchers to explore an ever-expanding array of novel devices capable of harvesting energy from sources such as wind, solar, and marine. Of these, wave energy is one of the least-utilized renewable energy sources, despite the large amount of energy extraction potential available around the globe [1, 2]. In the last forty years, many different wave energy converter (WEC) prototypes have been designed with varying levels of success [3]. However, most of these are still in the research and development phase and have not been implemented at full scale [2]. Wave energy has some significant benefits when compared to other renewable energy sources such as wind and solar. Firstly, in relation to other major renewable energy sources, wave energy has the highest energy density. This means that fewer wave devices would be needed to generate the same amount of energy as other renewable energy sources. Additionally, wave energy devices have been reported to have more reliable energy generation. Wave energy devices are capable of generating power up to 90 percent of the time, unlike wind and solar power devices, which generate power approximately 20-30 percent of the time [2].

In order to take advantage of these benefits, there are a number of technical challenges that must be dealt with to ensure proper performance and, thus, competitiveness in the global energy market [2]. For instance, wave energy is distributed unevenly throughout the globe such that regions with the highest wave energy potentials are located between latitudes of approximately 30 and 60 degrees in both hemispheres [3]. Thus, WECs located within these bands can perform better than those located elsewhere. Contending with extreme weather conditions presents another key challenge; rough sea states often result in extreme hydrodynamic loading of marine structures such as WECs, which may result in loading up to 100 times higher than that at normal operating conditions. Designing WECs to withstand such harsh conditions may result in increased WEC price to ensure robustness of the device [3]. More specifically, WECs require rigorous testing and evaluation procedures to ensure their expected performance is achieved as well as to convince investors and commercial developers to implement the devices at a large scale [3, 4]. However, such design expenses may be significantly reduced by conducting preliminary computational fluid dynamics (CFD) modeling of WECs. CFD models avoid the need to physically construct and test prototypes until the final design phase and allow engineers to explore a wider range of design variations and features where cost would otherwise be prohibitive. Additionally, the advent of affordable, high-performance computing (HPC) resources, such as cloud computing, has allowed for CFD to emerge as a crucial engineering design tool, making it the most viable approach for advancing WEC research.

Thus, the focus of this research is on simulating WEC devices using available computational fluid dynamics (CFD) software. In particular, near-shore WEC devices were focused on, owing to several key advantages they possess over offshore devices. Near-shore devices have the benefit of a more predictable wave direction due to the natural phenomena of refraction and reflection than offshore devices, where wave direction is difficult to predict due to the random nature of offshore sea states [2]. Near-shore devices are also easier to install and maintain since they are near land. Their proximity to land – and therefore the electrical grid – has the added benefit of requiring shorter transmission lines, resulting in smaller electricity losses [2]. On the other hand, the proximity of near-shore devices to coastlines results in the device-driving waves possessing less energy compared to those out at sea, and appropriate sites are harder to come by [3].

The near-shore device of interest to this study is called an oscillating wave surge converter (OWSC). This type of WEC is fixed to the seabed, consisting of a buoyant flap which is free to rotate about a hinge near its base [5]. OWSCs are different than the majority of wave devices in that they operate predominantly due to wave surge, producing horizontal oscillations, instead of heave, which produces vertical oscillations and is more common of offshore devices [6]. These structures must be capable of bearing stresses due to waves that pass over them at varying degrees of flap rotation [3].

Some notable experiments involving OSWCs include the Oyster prototypes (1 and 2) [6] and the wave tank experiment by Schmitt and Elsaesser 2015 [5] from the Queen's University Belfast. Experiments such as these are expensive and the available testing facilities are limited. In an effort to develop improved OWSC technology, numerical simulations can be conducted in conjunction with experimental testing to streamline the design and testing.

However, the dynamic behavior of OSWC devices presents an obstacle for some CFD simulation approaches since these devices experience very large rotations. There are a variety of CFD approaches that are capable of dealing with large values of rigid body rotation; however, the default approach of most CFD packages for body motion is mesh-morphing, for which large rotations can be problematic. Large rotations of the OWSC in turn create large mesh deformations in the mesh-morphing approach; without any kind of mesh deformation management method, such as re-meshing, these deformations will cause poor mesh quality, which may result in increased computational errors, numerical instabilities, and ultimately simulation divergence if the mesh quality is poor enough. Thus, alternate CFD approaches must be used. Three such alternate approaches are detailed herein: the re-meshing approach, the arbitrary mesh interface (AMI) approach, and the overset mesh approach. The overset mesh approach is the main focus of this study.

1.2 Previous Work

Simulating an OWSC device is not straightforward, which is why most research on these devices has primarily been based on experimental, scale model testing to date [5]. There are a few significant reasons for this. The idea of combining the computational power of computers with CFD has been around since at least the 1960s; however, the resources and software were not widely available [7]. One of the programs used in this research, OpenFOAM, was originally developed in 1989 under the name "FOAM" and was not released as the open source "OpenFOAM" until December 2004 [8]. Additionally, the default functionality of available CFD software is not capable of accurately modeling an OWSC. In typical operating conditions, OWSC devices rotate up to approximately 40 degrees in either direction and up to 80 degrees in extreme conditions. At these large values of rotation, typical mesh distortion methods are not sufficient, resulting in highly distorted cells and simulation failure [5]. There are a number of options to avoid this issue that include: re-meshing, arbitrary mesh interfaces (AMI), and overset meshes. Each method has its own set of advantages and challenges.

A re-meshing scheme involves letting the OWSC device rotate until some pre-selected time or until the mesh deforms some unacceptable amount. It is computationally expensive and involves either checking the mesh quality often or re-meshing often after an arbitrary amount of time. Both approaches are time consuming, though the former is a bit more efficient. Such a simulation was conducted by Winter and Motley (2020) using OpenFOAM v7 based on the experimental set-up of [5]. This method was chosen because it had the most straightforward implementation in OpenFOAM v7 using the solvers available [1].

An AMI method requires a computational domain with two mesh regions: a cylindrical moving mesh which surrounds the OWSC and rotates about the bottom hinge, and a static mesh for the remaining tank geometry. The two domains are coupled using two cylindrical AMI patches [5]. Due to the proximity of the seabed to the OWSC, this method requires the use of a cylindrical mesh that extends outside of the computational domain (past the tank floor). The AMI mesh must rotate about the center of rotation (i.e. the hinge) and encompass the entire OWSC. Due to geometry, the radius of this cylinder will have to be larger than the distance from the hinge to the seabed. Thus, an additional non-physical fluid region must be added to the simulation with an assigned dissipation parameter. Such a simulation was conducted by Schmitt and Elsaesser using OpenFOAM with a custom body motion solver of their own making. However, this type of functionality is not included in the standard OpenFOAM packages and must be implemented directly by the user.

The overset mesh method, like the AMI method, requires a computational domain with two mesh regions: an overset (or body-fitted) mesh, which surrounds and moves with a body, and a static background mesh. The fluid solution is interpolated between the two meshes at the fringe cells – cells near the edge of the overset grid. Unlike the AMI method, the overset mesh need not be a cylinder and need not rotate its center about the bottom hinge. It is capable of creating an overset mesh which does not exceed the domain and so does not require any kind of non-physical fluid region or dissipation parameter. The overset compatible solver, *overInterDyMFoam*, in the OpenFOAM v2012 package was recently made available.

Schmitt and Elsaesser conducted CFD research on an OWSC device in 2015 in which they performed an experimental wave tank test as well as a corresponding OpenFOAM simulation using the AMI method. They found good agreement between their numerical model and their experimental model [5].

Additionally, in 2005, a commercial OWSC system called Oyster was developed by the Aquamarine Power Ltd after a study by the Queen's University Belfast concluded that "a flap hinged to the sea bed at its lower edge with the top edge penetrating the water surface was the most promising form of this type of [OWSC] device" [6]. The conclusion was based on meeting five desired design criteria: 1) wide bandwidth response for good power capture over the entire working frequency range, 2) decoupling from incident waves for dealing with extreme conditions, 3) high structural efficiency, 4) minimal redundancy, and 5) easily replaceable sub-assemblies for maintenance purposes [6]. The progressive decoupling comes naturally to this type of OWSC device; as the flap rotates farther from the vertical position, the projected frontal area reduces, putting less and less force on the system. The proximity of the device to the seabed allows for tightly spaced units and flexible power transmission connections to help maximize energy extraction [6]. The deployment of Oyster 1 (2009) was the first instance of fixing a mobile machine of such size to the seabed in shallow coastal waters. The results from its first year of operation showed that it is possible to convert ocean wave power into electrical energy delivered to a national grid. However, OWSCs cannot be optimized on hydrodynamics alone. Other factors such as structure, power conversion and transmission, installation, and maintenance must be considered [6].

In this regard, the research herein hopes to better understand the effects of structure on OWSC devices and how such factors can be effectively modeled in the future. The work of Winter and Motley 2020 illustrated that, while no CFD code modifications were necessary, the default mesh-morphing method with re-meshing approach required time-consuming mesh quality monitoring and re-meshing operations. The work of Schmitt and Elsaesser illustrated that the AMI method, while effective for large rotations, is especially challenging for OWSCs due to their proximity to the seabed and requires modifications to existing CFD code. Meanwhile, the overset mesh method can handle large rotations without any re-meshing or mesh quality monitoring like the AMI method without requiring any CFD code modifications or non-physical fluid regions.

1.3 Objectives

As stated previously, most research on OWSC devices has primarily been based on experimental, scale model testing to date. Experimental tests on OWSC devices in wave tanks are expensive, time-consuming, and testing facilities are limited. Thus, it is advantageous to ensure that an experiment will provide useful results before beginning a physical test. Numerical simulations on OWSC devices can provide a preliminary screening of a wave tank experiment before investing the time and resources in a scale model or full-scale test. They can be used for fine-tuning of a design and prevent the necessity for quite as many physical tests if not all adjustments need be made in a lab. This can speed the design process and conserve resources. However, such numerical models of OWSC devices are neither numerous nor very straightforward, and there are a variety of meshing methods for dealing with OWSC devices depending on the available software and computational power. It is therefore of interest to investigate the differences between the various mesh methods that can be used to model an OWSC device and their results.

This research focuses primarily on the use of the overset grid method when modeling the OWSC device detailed in [5]. Two different overset grid models were created using two different programs: OpenFOAM v2012 and STAR-CCM+ 2021.3. The results of the two programs are compared to each other and to the experimental results of [5] as well as to the re-meshing method used in [1].

Chapter 2

NUMERICAL MODELING METHODOLOGY

The work presented in this thesis was performed using two computational fluid dynamics software packages: OpenFOAM and STAR-CCM+. This chapter provides an overview of both packages' fluid and rigid body motion solvers used for modeling a simplified OWSC device rotating about a single axis in two-phase (air, water) flow. This overview includes brief discussions of the equations and the numerical methods used by the solvers. Detailed discussions of the theory can be found in many places, including the OpenFOAM User Guide [9], the STAR-CCM+ User Guide [10] and many fluid mechanics textbooks (e.g. Munson et. al., 2013 [11]).

2.1 OpenFOAM Fluid Solver

Open Source Field Operation and Manipulation (OpenFOAM) is a C++ library with a large selection of executable applications. The version of OpenFOAM used for this research was OpenFOAM v2012, which was released in 2020 and supports overset grid capabilities. OpenFOAM applications fall into two broad categories: solvers and utilities. The solvers are designed to solve specific types of physics, and the utilities are designed for data manipulation and processing purposes. Some examples of OpenFOAM solvers include *simpleFoam* – a steady-state, single-phase solver for incompressible, turbulent flows – and *interFoam* – a transient, multiphase solver for two incompressible fluids which uses a volume of fluid method to capture the interface between them [12]. A wave tank is a multiphase problem with two phases: air and water. Thus, the solver used in this study was chosen from OpenFOAM's selection of multiphase solvers. The multiphase solver compatible with the overset mesh method is called *overInterDyMFoam* and is the solver used herein.

The fundamental underlying numerical method for all subsequent discussion of OpenFOAM and STAR-CCM+ is based on the finite volume method. In this method, a flow domain is divided up into cells of finite volume, often referred to as the control volume. The flow governing equations are discretized in space and time to form a system of linear algebraic equations that can be solved for every control volume in the domain grid. The flow field values are solved at the centroids of these finite volume cells, while flow field values at every other point in the flow domain are solved using user-specified interpolation, discretization, and matrix solution schemes [10, 13]. The finite volume method is an attractive option for CFD problems since the numerical flux between neighboring discretized cells is conserved (i.e. locally conservative) and it may be used for arbitrary mesh geometries [14]. The finite volume method is described in detail in [10] and [14].

The *overInterDyMFoam* multiphase fluid solver is defined as a "Solver for two incompressible, isothermal immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach, with optional mesh motion and mesh topology changes including adaptive re-meshing." by the OpenFOAM User Guide [12]. The VOF approach works by tracking the volume fraction of water (via a parameter called *alpha*) in each mesh cell over the course of the simulation. A cell containing only water would have an alpha value of 1.0 and a cell containing only air would have an alpha value of 0. A cell containing both water and air would have some fractional value that would then apply to the entire cell. For this reason, it is important to have a sufficiently fine mesh near the air-water interface to ensure the VOF method appropriately captures the free surface, minimizing interface smearing and maintaining and a sharp interface. In addition to mesh-refinement near the interface, OpenFOAM employs a built-in interface capturing scheme to help maintain sharp free surface interfaces [15, 16].

The goal of CFD solvers, such as *overInterDyMFoam*, is to solve a set of partial differential equations for flow field properties over a domain of interest. To this end, pressure and velocity are predicted by the Navier-Stokes equations and the continuity equation solved together for the given boundary conditions and geometry. The Navier-Stokes equations are derived via a momentum balance and represent the conservation of momentum. They are valid everywhere in the flow field of the fluid continuum [11]. The continuity equation expresses the conservation of mass. These derivations can be found in most standard fluid dynamics textbooks (e.g. Munson et al.'s *Fundamentals of Fluid Mechanics* [11]). For the case of incompressible Newtonian fluids, the Navier-Stokes equation can be expressed by Equation 2.1 and the continuity equation can be expressed by Equation 2.2 [10].

$$\frac{\partial(\rho\mathbf{u})}{\partial t} + \nabla \cdot (\rho\mathbf{u} \otimes \mathbf{u}) = -\nabla \cdot (p\mathbf{I}) + \nabla \cdot \mathbf{T} + \mathbf{f_b} \tag{2.1}$$

$$\nabla \cdot (\mathbf{u}) = 0 \tag{2.2}$$

In Equation 2.1, the left side of the equation represents the inertial forces, $\mathbf{f_b}$ represents the external forces (gravity, g), and the remainder represents the pressure and viscous forces. While Equations 2.1 and 2.2 can be used directly to solve a limited set of very simple problems analytically, more complex problems, such as modeling an OWSC, require numerical simulations where these partial differential equations are approximated with discretized algebraic equations and solved cell by cell. A CFD simulation numerically solves these equations for the relevant flow field values at discrete points specified by the user-defined mesh, and flow field values at non-grid point locations are determined via specified interpolation schemes [11].

It should be noted that the *overInterDyMFoam* solver assumes the governing equations have been divided by the density of the fluid; thus, all post-processed pressure values must be multiplied by the density to obtain the real physical results.

overInterDyMFoam uses the PIMPLE algorithm to iteratively solve equations for pressure and velocity. The pressure-implicit method for pressure-linked equations (PIMPLE) algorithm is a combination of the pressure-implicit split-operator (PISO) and semi-implicit method for pressure-linked equations (SIMPLE) algorithms. Both the PISO and SIMPLE algorithms are iterative solvers that involve evaluating initial conditions and then performing corrections. The PISO algorithm is used for transient cases and requires more than one correction while the SIMPLE algorithm is used for steady-state cases and only makes one correction. The PIMPLE algorithm can be adjusted with different numbers of inner (*nCorrectors*) and outer (*nOuterCorrectors*) correctors. Setting *nOuterCorrectors* to 1 makes the PIMPLE algorithm act like the PISO algorithm. The outer correctors specify the number of times the system of equations (coupled pressure and velocity) are solved, and the inner correctors specify the number of times the pressure is corrected per iteration [17]. For this study, the number of outer correctors was set to 2 and the number of inner correctors was set to 3. The time integration scheme (*ddtSchemes*) used was the Euler implicit time scheme (*Euler*). This time-stepping scheme is first-order, implicit, and designed for transient problems [18].

Though the Navier-Stokes equations are valid everywhere in the fluid domain, the results are affected by how the domain is discretized, notably for capturing phenomena such as turbulence. Depending on the size of the mesh and fineness of the grid, effects of turbulence may not be captured well, and a turbulence model is generally required to obtain practical solutions. If no turbulence model is used, the governing equations (2.1 and 2.2) must be solved for an extensive range of temporal and spatial scales to capture turbulence, the degree of which typically ranges from impractical to impossible. This approach is referred to as Direct Numerical Simulation (DNS) [19]. OpenFOAM v2012 supports three types of turbulence models: Reynolds Averaged Simulation (RAS), Detached Eddy Simulation (DES), and Large Eddy Simulation (LES). It should be noted that the RAS model specified by OpenFOAM is more commonly known as Reynolds Averaged Navier-Stokes (RANS). RANS is the most commonly used turbulence model in industrial applications and is the model used for this study [19]. The RANS equations are obtained via Reynolds averaging, which results in a modified form of the Navier-Stokes equations which include additional unknown terms called Reynolds stresses [19, 20]. LES models resolve turbulent phenomena at smaller scales than RANS models, providing greater accuracy; however, they are more computationally costly due to time step and mesh resolution requirements. LES works by filtering out small scale turbulence while the "large eddies," or most energy-intensive turbulence areas, are resolved directly. This requires a higher mesh resolution in order to capture the large eddy turbulence details at small scales, which involves using smaller cells sizes and, thus, a larger number of cells, increasing computational cost significantly [19]. DES is a hybrid RANS-LES approach designed to mitigate the computational drawbacks of LES due to near-wall meshing requirements [12]. It employs RANS modeling near the walls and LES modeling elsewhere in order to reap the benefits of LES while maintaining RAS efficiency [19]. The mean flows computed by RANS are sufficient for determining engineering quantities of interest for an OWSC device in a wave tank with typical waves and help keep the computational costs down. The RANS formulation is detailed in [20].

OpenFOAM includes a selection of RANS turbulence closures based on linear and non-linear eddy viscosity models, and Reynolds stress transport models [20]. Two of the most common turbulence model families are the k-epsilon (k-ϵ) and k-omega (k-ω) models where k is the turbulent kinetic energy, ϵ is the turbulent kinetic energy dissipation rate, and ω is the turbulence specific dissipation rate [12, 21]. Both are two transport-equation linear-eddy-viscosity (Boussinesq hypothesis, [22, 23]) closure models. OpenFOAM v2012 does not offer a standard k-omega model; instead, it offers the Shear Stress Transport (SST) k-omega model, which is essentially a hybrid between the k-epsilon model and the standard k-omega model [12, 9]. The version of the SST k-ω model used in OpenFOAM v2012 is described in [21] and is based on the 2003 model by Menter et al. [24]. Generally, the standard k-omega model works well near walls and the k-epsilon model works well away from walls; thus, the combined SST k-omega model attempts to combine the best performance of each and is the model used in this study [25]. The SST k-ω model is capable of capturing flow separation and, unlike k-ϵ, it does not require any near-wall treatment [12, 21]. Additionally, a comparison study from [26] found that the SST k-ω model produced more accurate fluid force and pressure results on a stationary test structure impacted by both broken and unbroken waves than four different k-ϵ models. The increased accuracy of the SST k-ω model over the k-ϵ models was due primarily to the fact that the SST k-ω model handled flow separation around static bodies (e.g. concrete blocks) better than the k-ϵ models, whose standard versions were originally developed for unseparated flows around airfoils [26].

The solution algorithm setup and controls are specified in the *fvSchemes, fv-Solutions, controlDict,* and *decomposeParDict* files. The *fvSchemes* dictionary specifies the numerical schemes for terms appearing in various applications being run. This includes things like interpolation schemes, including setting the overset interpolation scheme used, derivatives (e.g. the first time derivative), divergence schemes (e.g. a typical convection term), and more [18]. The *fv-Solution* dictionary specifies equation solvers, tolerances, and algorithms. This includes the settings for the PIMPLE algorithm used in this study as well as solver settings for flow field values like the volume fraction (*alpha*), pressure (*p_rgh*), and velocity (U), among others [17]. The files used for this study are included in the appendix and detail all algorithm and control settings.

2.2 OpenFOAM Rigid Body Motion

The body motion solver used in this study is the *sixDoFRigidBodyMotion* solver. This solver allows six degrees of freedom of motion (rotation and translation in the x, y, and z directions) for a rigid body. Restraints (springs) and constraints (rotation constrained to single axis) can be added. The rigid body motion is setup in the *dynamicMeshDict*. There, the mass of the rigid body, mass moment of inertia, acceleration relaxation and damping factors, motion solver, constraints, and restraints must be specified. For the case of an overset grid simulation, the *dynamicFvMesh* type must be set to *dynamicOversetFvMesh*.

The *dynamicFvMesh* type specifies the type of motion to be solved [12].

For the purposes of this study, constraints were specified such that the OWSC hinge location was fixed and rotation was only permitted about the hinge (y-direction). This was done by imposing a *sixDoFRigidBodyMotionConstraint* constraint type in the *dynamicMeshDict*. A fixed line constraint was created by using the *sixDoFRigidBodyMotionConstraint* type *line* with a *centreofrotation* value located at the center of the hinge, (0, 0, 0), rotating about the y-axis, ⟨0,1,0⟩. In addition, a torsional spring restraint was added using the *sixDoFRigidBodyMotionRestraint* of type *linearAxialAngularSpring* about the y-axis with a prescribed stiffness of 5 N/m to represent some bearing friction.

The *sixDoFRigidBodyMotion* solver solves for the motion of a rigid body based on the computed forces acting upon it. These forces are comprised of components such as fluid forces (i.e. pressure, viscous forces), gravity, and spring forces. The tangential and angular accelerations are calculated using Newton's second law from the given state and forces. The new tangential and angular velocities for the next time step are determined by corrections made using the Newmark-beta time integration scheme [27]. Velocity corrections are made in accordance with Equation 2.3 [28].

$$\dot{q} = \dot{q_o} + \Delta t(\gamma \ddot{q} + (1-\gamma)\ddot{q}_o) \tag{2.3}$$

Once the new velocities are known, they can be used to determine the degree of rotation and the translational distance moved over the course of one time step. The new position is determined by Equation 2.4 [28].

$$q = q_o + \Delta t \dot{q}_o + \sqrt{\Delta t}(\beta \ddot{q} + (0.5-\beta)\ddot{q}_o) \tag{2.4}$$

The type of Newmark method used for this simulation was the average constant acceleration method, which uses a gamma value of 0.5 and a beta value of 0.25. This method is unconditionally stable, regardless of the time step used.

For each time step, OpenFOAM reports the new values of angular velocity, linear velocity, center of mass, center of rotation, body orientation, and spring moment in the *overInterDyMFoam* log file. In the case of the OWSC device, only single-axis, rotational movement is of interest.

After calculating the incremental change in rotation of the rigid body, both the rigid body (OWSC) and the overset (body-fitted) grid must be displaced accordingly [29]. There is no mesh deformation in an overset grid simulation, instead interpolation is performed between the overset and background meshes. This will be discussed in more detail in the following chapter.

The other setting of note in the *dynamicMeshDict* are the *accelerationRelaxation* and *accelerationDamping* factors. These factors help maintain the stability of the rigid body motion solver. Such stability issues arise in situations of high acceleration where sudden high acceleration values cause divergence of the solver. The *accelerationDamping* factor is similar to a damping coefficient in that it reduces the computed acceleration on a body proportionally to the body's acceleration magnitude. Meanwhile, the *accelerationRelaxation* factor is a direct reduction on the acceleration. Both factors can range from 0.0 to 1.0 [30]. For this study, the *accelerationRelaxation* factor was set to 0.4 and the *accelerationDamping* factor was set to 0.8. There was also an attempt to conduct the simulation with an *accelerationRelaxation* factor of 0.6; however, this attempt resulted in the solution divergence issues mentioned above where the solution suddenly experienced extremely high values of acceleration and numerical instability. Thus, the *accelerationRelaxation* factor was decreased to 0.4. The application of the acceleration relaxation and damping factors is shown by Equation 2.5 [31].

$$\ddot{q} = a_{Damp}(a_{Relax} * \ddot{q} + (1 - a_{Relax})\ddot{q}_{prev}) \tag{2.5}$$

These kinds of relaxation factors have been used for both fluid flow and body motion in CFD. In terms of body motion, an acceleration relaxation factor functions as an added mass term in the equation of motion (i.e. Newton's second law). An added mass effect must be considered since, in dense fluids, the hydrodynamic force changes over the course of one time step. This force change becomes especially significant for cases of light bodies subjected to high acceleration and can lead to solution divergence [5, 32]. The OpenFOAM default functionality does not include a method for dealing with added mass effects aside from the application of relaxation factors. Accounting for added mass effects without using relaxation factors would involve creating custom code for implementation in OpenFOAM as was done in [5] using one added mass algorithm. Another added mass algorithm is detailed in [33]. Added mass has also been dealt with by determining appropriate relaxation factors for corrector step as was done in [34] using Aitken's dynamic under-relaxation method. In this method, the first under-relaxation factor was set conservatively between 0.1 and 0.5 and then updated each subsequent correction to the body motion to a value between 0.1 and 1.0 based on another corrector equation [34].

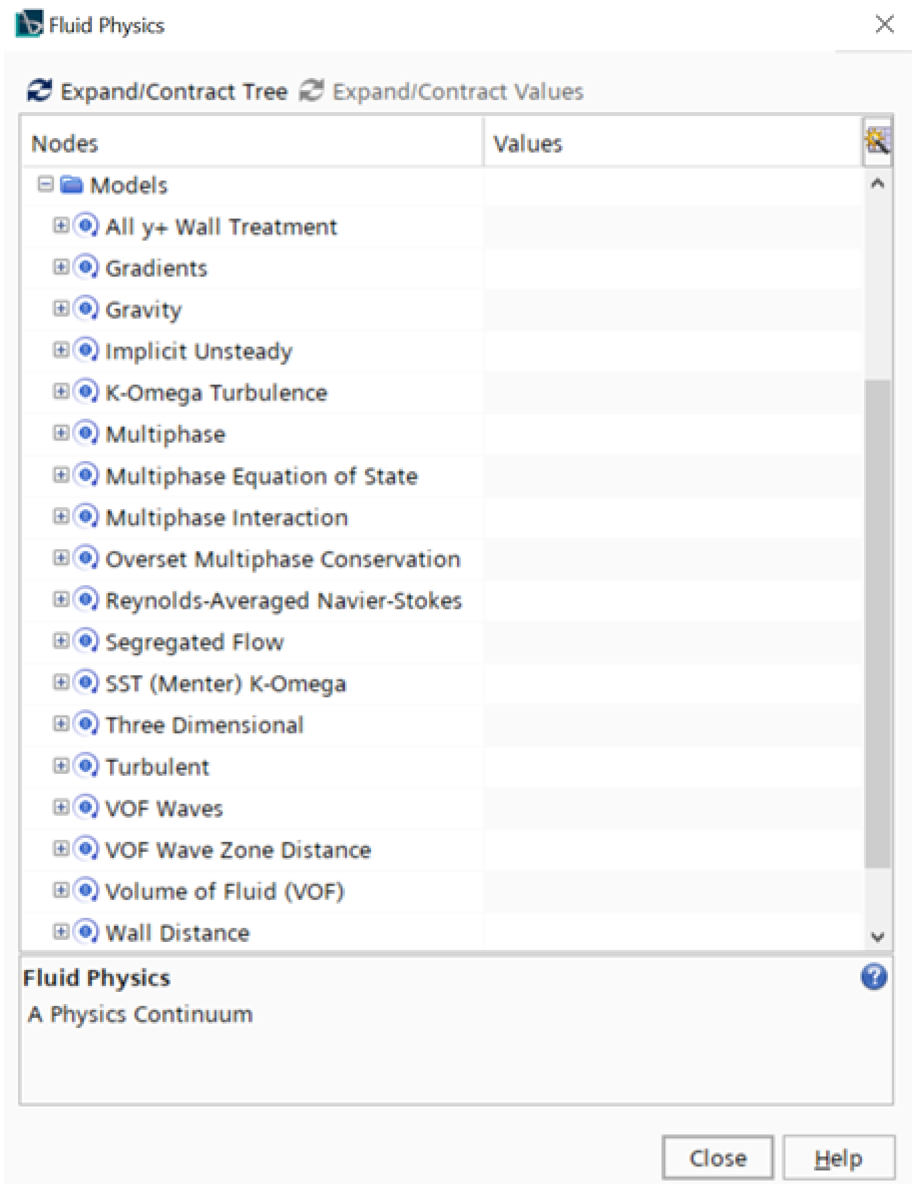
For the sake of simplicity and ease of use for future research, the desire was to work within OpenFOAM's default functionality. This meant using the provided relaxation factor settings and, subsequently, constant values for the relaxation factors themselves. The *accelerationRelaxation* value of 0.4 was chosen for a couple reasons. Firstly, the solution experienced a numerical instability when *accelerationRelaxation* was set to 0.6; changing this value to 0.4 for the exact same simulation resolved this instability. An *accelerationRelaxation* value of 0.4 was also used by Winter & Motley 2020 [1] using a mesh-morphing with re-meshing method for a very similar CFD problem with the same domain to good effect. As another reference point, the *accelerationRelaxation* factors used in the OpenFOAM v2012 tutorial cases for the *overInterDyMFoam* solver were 0.4 (case with a boat propeller: boatAndPropeller) and 0.6 (two floating body cases: floatingBody and floatingBodyWithSpring) [35].

2.3 STARCCM+

The Simcenter STAR-CCM+ Software is proprietary software – originally created by CD-Adapco before being purchased and subsequently developed by Siemens – capable of solving a wide array of multi-physics problems including fluid and solid continuum mechanics, rheology, aeroacoustics, reacting flows (e.g. combustion), electro-chemistry (e.g. batteries), and electromagnetism applications; although, the focus of this study was on its multiphase CFD and rigid-body motion fluid-structure interaction modeling capabilities. The version used herein was STAR-CCM+ 2021.3, which was released in 2021 and supports the overset grid method. Unlike OpenFOAM, STAR-CCM+ has a graphical user interface (GUI) where the model is set up using simulation trees. Since the STAR-CCM+ simulation is being used as a method of model validation for the OpenFOAM simulation, the physics and settings used in STAR-CCM+ are very similar to those detailed in Sections 2.1 and 2.2 by design. Additionally, although the STAR-CCM+ User Guide [36] provides an in-depth look at the theory behind the software, the source code is not publicly available.

As with OpenFOAM, STAR-CCM+ solves the discretized form of the Navier-Stokes partial differential equations and the continuity equation to predict pressure and velocity over the computational domain using the finite volume method. This is discussed in Section 2.1 and will not be repeated. Instead, this section will focus on the various settings and solvers used in by the STAR-CCM+ software to solve these equations.

STAR-CCM+ physics are defined by first creating a physics continuum and selecting the relevant models. One physics continuum is capable of handling multiphase flow. For this study, the physics models specified in Figure 2.1 were used.


Fluid Physics

Expand/Contract Tree | Expand/Contract Values

Nodes	Values
Models	
All y+ Wall Treatment	
Gradients	
Gravity	
Implicit Unsteady	
K-Omega Turbulence	
Multiphase	
Multiphase Equation of State	
Multiphase Interaction	
Overset Multiphase Conservation	
Reynolds-Averaged Navier-Stokes	
Segregated Flow	
SST (Menter) K-Omega	
Three Dimensional	
Turbulent	
VOF Waves	
VOF Wave Zone Distance	
Volume of Fluid (VOF)	
Wall Distance	

Fluid Physics

A Physics Continuum

Close | Help

Figure 2.1: STAR-CCM+ dialog window listing the fluid physics models used.

The Implicit Unsteady model uses the Implicit Unsteady solver, which controls the calculation updates of the flow fields for each physical time. The first-order temporal discretization option was selected which uses the same type of

Euler implicit time integration scheme as OpenFOAM. This solver also controls the time-step size, functioning much like setting the adjustTimeStep setting in OpenFOAM to *yes*. The segregated flow model and solver function similarly to the PIMPLE algorithm used in OpenFOAM, making use of a predictor-corrector method to solve coupled equations of pressure and velocity. However, unlike OpenFOAM, STAR-CCM+ only has two available pressure-velocity coupling algorithms: SIMPLE and PISO. For a segregated flow model that uses the implicit unsteady time integration scheme, the SIMPLE algorithm is automatically invoked [36]. Although the SIMPLE algorithm is geared more toward steady-state solutions, it is capable of providing accurate results for transient solutions so long as the time step is sufficiently small [36, 17]. Indeed, for small time-steps, PISO and SIMPLE have the same level of temporal accuracy [36].

The SST (Menter) K-Omega turbulence model functions essentially the same as the *kOmegaSST* model used in OpenFOAM. Both of these turbulence closure models fall within the Reynolds-Averaged Navier-Stokes (RANS) class of models. The STAR-CCM+ simulation also uses a multiphase model with two phases, and the Volume of Fluid (VOF) method with a convenient VOF Waves model that is useful for setting initial and boundary conditions [36]. The Overset capabilities will be discussed in Chapter 4.

The OWSC device was once again simulated as a rigid body permitted to rotate about its hinge (y-axis). Rigid body motion fluid-structure interaction (FSI) is simulated using the Dynamic Fluid Body Interaction (DFBI) module in STAR-CCM+. For the case of a rigid body permitted to rotate about one axis, a 2D body motion was selected as most appropriate. The 2D body motion option used is called One-DOF Rotating Motion, which is the 2D version of a single degree of freedom rotating motion suitable for 3D simulations [36].

The DFBI module solves the following governing equation, Equation 2.6, to determine the rotation of the rigid body in the case of free motion (i.e. 6-DOF) [10].

$$\mathbf{M}\frac{d\boldsymbol{\omega}}{dt} + \boldsymbol{\omega} \times \mathbf{M}\boldsymbol{\omega} = \mathbf{n} \tag{2.6}$$

where **M** is the symmetric tensor of the moments of inertia, ω is the angular velocity, and **n** is the resultant moment acting on the body [10]. Equation 2.6 uses the local coordinate system of the body where the origin is taken to be the body's center of mass. When simplified to the One-DOF Motion solver, Equation 2.6 becomes Equation 2.7 [10].

$$M\frac{\partial}{\partial t}\omega = n \tag{2.7}$$

where M is now a scalar defined with respect to the axis of rotation.

Chapter 3

NUMERICAL MODEL

This chapter describes the computational domain used in the OpenFOAM and STAR-CCM+ simulations and the experiment the CFD models are attempting to simulate. Wave properties and boundary conditions are explained in detail along with their corresponding software settings. The process of generating a background mesh for the wave tank in OpenFOAM is discussed.

3.1 CFD Model Domain

The CFD models created for this research are compared against the scale model experiment conducted by Schmitt and Elsaesser (2015) using the wave tank at the Queen's University Belfast [5].

The wave tank used in the Schmitt and Elsaesser experiment was 4.58 meters wide and 20 meters long with varying floor elevations. The tank geometry is shown in Figure 3.1.

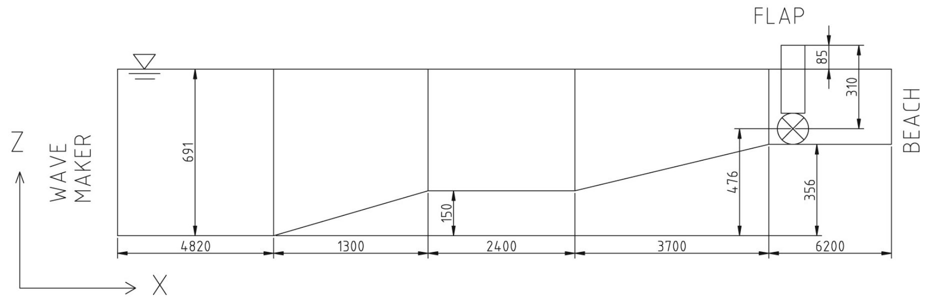


Figure 3.1: Wave Tank Geometry and Water Level (in mm). Figure courtesy of [5].

The OWSC device (or flap) had the dimensions 0.1 m (thickness, x-direction) by 0.65 m (width, y-direction) by 0.341 m (height, z-direction). Additionally, the properties in Table 3.1 were provided [5, 37].

Table 3.1: OWSC device properties. Height measurements in reference to lowest floor. [5, 37]

Hinge Height	0.467	m
Height of CoG	0.53075	m
Mass	10.77	kg
Inertia	0.1161	$kg \cdot m^2$

Further details on the physical setup of the experiment can be found in [5].

For both the OpenFOAM and STAR-CCM+ models, the same wave tank and OWSC geometry as listed above has been used with the following exceptions: the OWSC inertia value, the height of the OWSC center of gravity (CoG), and the initial free surface height of the OpenFOAM simulation. During the initial simulations conducted in OpenFOAM, it was discovered that using the CoG value and mass inertia value as listed in Table 3.1 resulted in numerical instability and solution divergence. Thus, the OWSC inertia value and the CoG height were changed to match the values from another paper written by Benites-Munoz et al. 2020 [38] which also compared results to Schmitt and Elsaesser [5]. Benites-Munoz used a OWSC device with a higher CoG and larger inertia value. The CoG height used in the OpenFOAM and STAR-CCM+ simulations was 0.587 m and the mass moment of inertia used was 0.1750 kg$\cdot m^2$. This change stabilized the solution. The initial free surface height used in the OpenFOAM simulation was also 0.01 meter higher than that used in [5]. While not an exact comparison, then, to the Schmitt and Elsaesser experiment as presented in the literature, the results will be compared for order of magnitude and qualitative purposes.

3.2 Waves and Boundary Conditions

The Schmitt and Elsaesser [5] experiment and CFD model used a wave period of 2.0625 seconds and an amplitude of 0.038 meters near the OWSC. These values create an approximately 1:40 scale model of a wave with a 13 second period and 1.5 meter height.

OpenFOAM and STAR-CCM+ define the wave height or wave amplitude at the wavemaker location. While this value is not presented by Schmitt and Elsaesser, the authors clarified upon request that a wave height of 0.05 meters was generated near the experimental wavemaker. The corresponding wave length for a wave with a period of 2.0625 seconds and a height of 0.05 meters was calculated to be 4.78 meters using the linear dispersion equation. One version of this equation is shown by Equation 3.1.

$$\lambda = T\sqrt{\frac{g\lambda}{2\pi}\tanh\frac{2\pi h}{\lambda}} \tag{3.1}$$

Additionally, the Ursell number of the waves was given near the wavemaker as 3.4 [5] where the Ursell number is defined by Equation 3.2.

$$U = \frac{\lambda^2 H}{h^3} \tag{3.2}$$

In Equations 3.1 and 3.2, λ is the wavelength, H is the wave height or two times the wave amplitude, and h is the water depth. Thus, Equation 3.2 provides another way to calculate the wavelength used in [1]. The wavelength results are approximately the same whether the Ursell number or the linear dispersion equation is used, aside from rounding errors.

For OpenFOAM v2012, the user must specify the *waveModel, waveHeight, wavePeriod,* and *wavelength* where *waveHeight* is the wave height at the wave-maker (0.05 meters in this case). OpenFOAM v2012 has a variety of wave generation (*waveModel*) types available for different wave theories. The regular wave theories available include Cnoidal, Stokes I, Stokes II, Stokes V, and a Stream Function [39]. This study uses the Stokes I model, which is equivalent to Airy Wave Theory or linear wave theory. The first order Stokes (Stokes I) theory is a small-amplitude wave theory, which describes waves with heights small in comparison to their wavelengths. The Stokes I *waveModel* computes the free surface elevation using Equation 3.3 [39].

$$\eta(x,t) = 0.5H\cos(k_x x + k_y y - \omega t + \phi) \tag{3.3}$$

where k is the angular wavenumber, $k = \frac{2\pi}{\lambda}$, with $k_x = k * \cos\theta$ and $k_y = k * \sin\theta$; ω is the angular frequency, $\omega = \frac{2\pi}{T}$; and ϕ is the phase shift. The *waveAngle*, θ, is set by the user; in the case of this study, θ and ϕ have been set to 0.0 and so Equation 3.3 simplifies to Equation 3.4 [39].

$$\eta(x,t) = 0.5H\cos(kx - \omega t) \tag{3.4}$$

One example of the Airy Wave theory derivation can be found in [40]. All wave properties are specified in the *constant* folder in the *waveProperties* dictionary.

For STAR-CCM+ 2021.3, the wave model type used is the “VOF Waves” model. This type of model can be used to simulate steady surface gravity waves and is convenient for setting certain initial conditions [36]. Like with OpenFOAM, the STAR-CCM+ VOF Waves support a selection of wave models based on different theories. These include Flat Wave, First Order Wave, Fifth Order Wave, Superposition Wave, Cnoidal Wave, and Irregular Wave [36]. The First Order Wave model uses a first order approximation to the Stokes wave theory and corresponds to the Stokes I *waveModel* used in OpenFOAM. Thus, it was the model selected for use in this study. As reported in the STAR-CCM+ User Guide, the free surface elevation for a First Order Wave model is computed using Equation 3.5 [36].

$$\eta = a\cos(\mathbf{K}\cdot\mathbf{x} - \omega t) \tag{3.5}$$

where a is the wave amplitude and the $\mathbf{K}$ and ω variables are the same as in the OpenFOAM Equation 3.3.

The problem domain has seven user-defined boundaries: the two sides of the wave tank (rightWall and leftWall), the floor of the tank (bottomWall), the atmosphere boundary (atmosphere), the inlet (inlet), the outlet (outlet), and the rigid body OWSC (flap). The labels in parenthesis were those used in the OpenFOAM dictionaries and also in STAR-CCM+. The boundary conditions in OpenFOAM are defined by files in an initial and boundary conditions folder, *0.org*. The *0.org* folder contains boundary condition files for volume fraction (*alpha*), turbulent kinetic energy (*k*), turbulent viscosity (*nut*), turbulence specific dissipation rate (*omega*), velocity vector components (*U*), total pressure without the hydrostatic component (*p_rgh*), boundary displacements (*pointDisplacement*), and – for overset grid simulations – *zoneID*. Table 3.2 details the boundary conditions used for each boundary.

Table 3.2: OpenFOAM Boundary Conditions

Boundary	**alpha**	**k**	**nut**	**omega**	**p_rgh**	**pointDisplacement**	**U**
flap	zeroGradient	kqRWallFunction	nutkWallFunction	omegaWallFunction	fixedFluxPressure	calculated	movingWallVelocity
rightWall	zeroGradient	kqRWallFunction	nutkWallFunction	omegaWallFunction	fixedFluxPressure	fixedValue	noSlip
leftWall	zeroGradient	kqRWallFunction	nutkWallFunction	omegaWallFunction	fixedFluxPressure	fixedValue	noSlip
bottomWall	zeroGradient	kqRWallFunction	nutkWallFunction	omegaWallFunction	fixedFluxPressure	fixedValue	noSlip
atmosphere	inletOutlet	inletOutlet	zeroGradient	inletOutlet	totalPressure	fixedValue	pressureInletOutletVelocity
inlet	waveAlpha	inletOutlet	calculated	inletOutlet	fixedFluxPressure	fixedValue	waveVelocity
outlet	variableHeightFlowRate	inletOutlet	calculated	inletOutlet	zeroGradient	fixedValue	inletOutlet

The *zeroGradient* boundary condition works by imposing a zero-gradient condition from the patch internal field to the patch faces [41]. In other words, the values in the cells next to the boundary will be the same as the values on the boundary. The *fixedValue* boundary condition sets a user-specified fixed value constraint on the boundary [41]. In the case of the *pointDisplacement* boundary conditions, all boundaries except for flap were set to a *fixedValue* boundary of zero, where there was no boundary displacement. The *calculated* boundary condition does not actually evaluate anything, instead it refers to a value set by a field assignment. The *inletOutlet* boundary condition will assign a *zeroGradient* condition for outflow cases and a specified *fixedValue* condition for inflow cases [41]. The *fixedFluxPressure* boundary condition chooses the pressure gradient such that the flux on the boundary matches that specified by the velocity boundary condition [41]. The *noSlip* boundary condition sets the velocity to zero at the wall boundaries [41]. The *pressureInletOutletVelocity* boundary condition is used on boundaries where pressure is specified. A *zeroGradient* condition is used for outflow. For inflow, velocity is set equal to the boundary-normal component of the adjacent internal cell value [41]. The *totalPressure* boundary condition provides a user-specified, constant total pressure condition to a boundary [41].

To set up the wave case specified above, velocity and phase fraction field boundary conditions at the inlet must be set to *waveVelocity* and *waveAlpha*, respectively. This indicates the location of the wave maker, where waves should begin propagating. It also tells the simulation to initialize and run the waves specified by the *waveProperties* dictionary [39]. The *variableHeightFlowRate* boundary condition is used for alpha on the outlet since the waves will change the water level and therefore change the distribution of alpha over the outlet boundary. The *movingWallVelocity* velocity boundary condition is applied to cases with moving walls [41]. The remaining boundary conditions are all different types of wall functions: *kqRWallFunction, nutkWallFunction,* and *omegaWallFunction.* The *nutkWallFunction* boundary condition provides a wall constraint on turbulent viscosity based on *k*. The *omegaWallFunction* boundary condition provides a wall constraint on ω and turbulent kinetic energy production contribution. These two boundary conditions are essentially just *fixedValue* boundary conditions specific to the nut and omega boundary files for walls. The *kqRWallFunction* boundary condition is a simple way of wrapping *k*, square-root of turbulent kinetic energy (q), and Reynolds stress tensor fields (R) all under a *zeroGradient* boundary condition as needed [42].

The STAR-CCM+ simulation contains all the same boundaries with the same labels used in OpenFOAM. The stationary tank walls (rightWall, leftWall, bottomWall) and the OWSC faces (flap) are assigned as boundary type Wall. The atmosphere boundary (atmosphere) is defined as boundary type Pressure Outlet, the inlet (inlet) is type Velocity Inlet, and the outlet (outlet) is type Pressure Outlet. Unlike with OpenFOAM, STAR-CCM+ does not require the user to set individual boundary conditions for every field value if "auto-select recommended models" is activated. The boundary condition settings available are dependent on the models selected (see Figure 2.1).

A Wall boundary is used for impermeable surfaces that confine fluid or solid regions. The no-slip condition is applied to all Wall boundaries in viscous flow by default and this setting was not changed. A Velocity Inlet boundary is used for inflow conditions when the velocity and fluid properties over the boundary are known, as was the case for the inlet boundary of the model [36]. In the case of this study, the fluid distribution over the inlet boundary was known (volume fraction) as well as the velocity distribution in the form of First Order waves of known physical properties. The VOF Waves model provides field functions that may be used to specify boundary condition flow field values – in this case velocity – at the inlet and outlet as well as initial condition flow field values throughout the rest of the computational domain. These conditions are prescribed to the Velocity Inlet boundary and used to calculate inlet volume flux, momentum flux, and energy flux [36]. A Pressure Outlet boundary is used for outflow conditions and imposes the working pressure on the boundary. The working pressure in STAR-CCM+ is the sum of the static and hydrostatic pressures. The Pressure Outlet in STAR-CCM+ is similar to the *inletOutlet* boundary condition in OpenFOAM in that other values, such as velocity, may have both outflow and inflow conditions, i.e. backflow is allowed. The velocity values are taken as the values from the adjacent solution domain for outflow conditions (*zeroGradient*). If using the Extrapolated backflow option, the velocity values for inflow conditions are set to the internal velocity values (*fixedValue*) [36]. There are other ways to deal with backflow in STAR-CCM+, but this study uses the Extrapolated option.

Boundary conditions related to the turbulence model (k and ω), were automatically set in STAR-CCM+ based on the boundary type used by allowing the software to automatically select recommended models. The auto-selected turbulence specification for the inlet, outlet, and atmosphere boundaries was the Intensity + Viscosity Ratio method, which calculates the parameters k and ω from a specified turbulence intensity and length scale. For the walls, the blended wall function was used, which is selected for use with no-slip walls when the all-y+ wall treatment model is selected. All wall boundaries were set as smooth, for use with regular wall treatment. Default values were used for all turbulence boundary condition parameters such as E, the log law offset, which was 9.0, and Kappa, the von Karman constant, which was 0.42 [36]. These values were used for the simulation in lieu of any available data to inform turbulence parameters.

3.3 Mesh Generation

The overset grid method uses two meshes: a background mesh and an overset (body-fitted) mesh. The latter will be discussed in more detail in Chapter 4. All meshes used in this study were generated using OpenFOAM's *blockMesh* utility. *blockMesh* is one of the simplest ways to create a mesh in OpenFOAM; it generates a mesh from a user-defined dictionary file, *blockMeshDict.* The *blockMesh* utility reads this dictionary file, generates a mesh, and writes the outputted mesh data to generated files named *points, faces, cells,* and *boundary* [43]. Mesh generation is done as a pre-processing step prior to running any CFD simulations.

To define a mesh in the *blockMeshDict*, the user must define vertices, blocks, and boundaries. Vertices are specified as coordinates and are used to define the blocks. Each block in the *blockMeshDict* is defined with eight vertices arranged as in Figure 3.2.

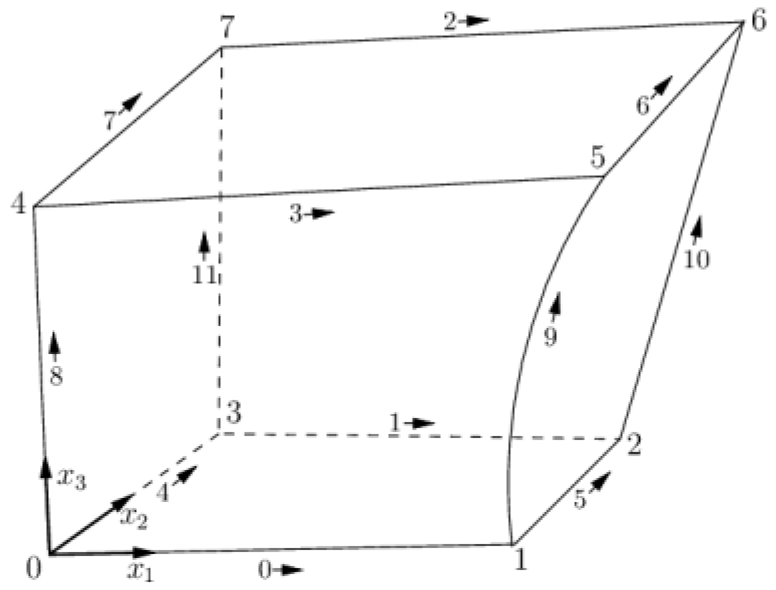


Figure 3.2: A single hexahedral block with vertex labeling convention. Figure courtesy of [43].

The following is an example of a block definition as given by the OpenFOAM User Guide [43].

```
blocks
(
hex (0 1 2 3 4 5 6 7) // vertex numbers
(10 10 10) // numbers of cells in each direction
simpleGrading (1 2 3) // cell expansion ratios
);
```

The block vertex numbers in the example have been ordered as (0 1 2 3 4 5 6 7). Based on Figure 3.2 the first two vertices listed define the x_1 direction, the second and third vertices listed define the x_2 direction, and the fifth and sixth vertices listed define the x_3 direction. Thus, in this example and Figure 3.2, the local x_1 direction is defined from vertex 0 to vertex 1, the local x_2 direction is defined from vertex 2 to vertex 3, and the local x_3 direction is defined from vertex 4 to vertex 5. The local coordinate system must be defined as a right-handed coordinate system. The order of the vertices is important and incorrectly ordering the block vertices will result in inverted blocks. The number of cells in each local block direction are also set in the block definition as well as any cell grading desired [43].

The wave tank (or background) mesh created for this study uses 12 blocks. The large number of blocks was due to cell grading, where it was desired that the mesh be finer near the air-water interface and near the OWSC. If multiple blocks are used, the user must ensure that the faces between blocks match in terms of cell number/dimensions. In OpenFOAM, this is known as "face matching." The blocks used are shown in Figure 3.3.

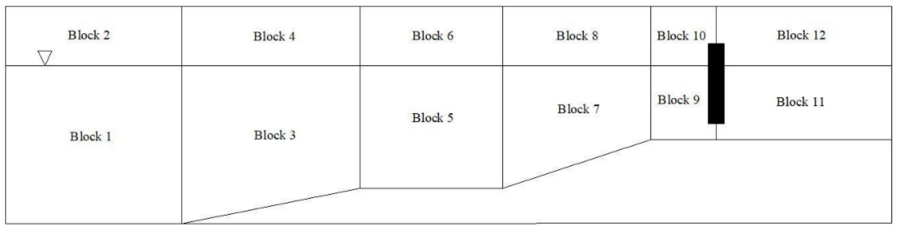


Figure 3.3: Mesh blocks defined in *blockMeshDict*.

The *blockMesh* utility allows for grading the cell dimensions via expansion ratios in each of the three block dimensions. The grading option used to create the mesh was *simpleGrading*. This grading method requires the user to specify uniform expansion ratios for each local direction of a block.

The final component required by the *blockMeshDict* is a list of boundary definitions. Boundaries must also be specified using the user-defined vertices and the vertex ordering convention of Figure 3.2 to define the boundary faces. As an example, the bottomWall boundary requires that 6 faces be defined, as can be seen in Figure 3.3. The faces which define each boundary must be assigned a patch type. For this study, the boundary patch types used were *patch* – a generic patch type that contains no geometric or topological mesh information – and *wall* – a specific type of patch which coincides with a wall – where the *wall* type was applied to all impermeable walls and the *patch* type was used for the inlet, outlet, and atmospheric boundary [43].

The generated background mesh used the same wave tank dimensions as described in Section 3.1 and contains approximately 18.1 million cells. This mesh is shown in Figure 3.4. Additionally, a 3D view of the wave tank is shown in Figure 3.5.

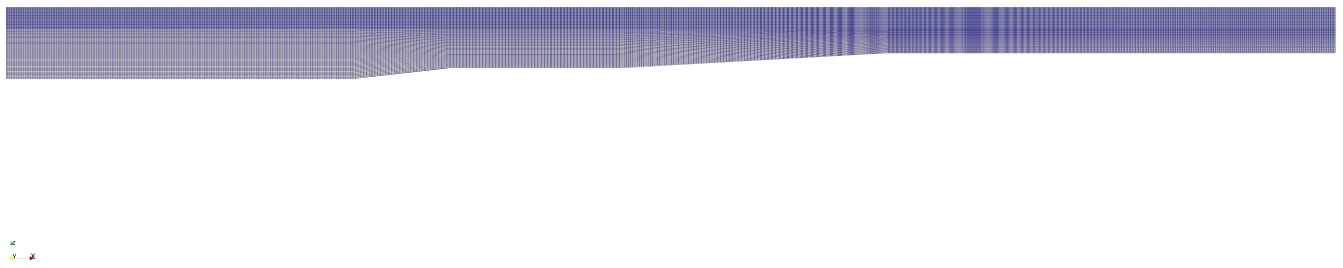


Figure 3.4: Background mesh, generated via the ParaView [44] software.

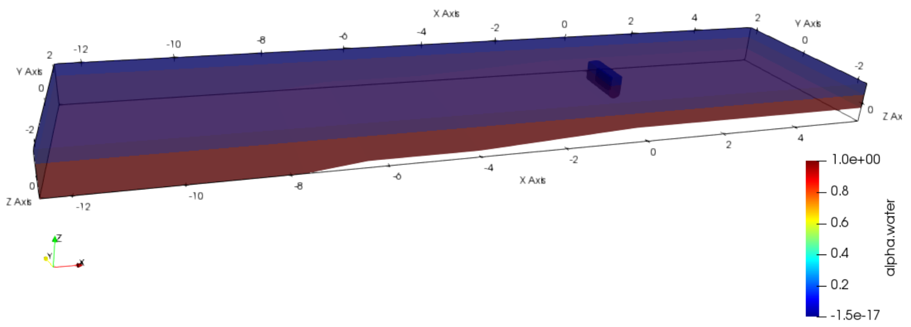


Figure 3.5: 3D wave tank model with overset mesh, generated in the ParaView [44] software.

Instead of generating new meshes in STAR-CCM+ for a comparison simulation, the OpenFOAM meshes were converted to Fluent mesh (.msh) files using the OpenFOAM utility, *foamMeshToFluent*. The Fluent meshes could then be imported into STAR-CCM+ as volume meshes. Thus, the OpenFOAM and STAR-CCM+ simulations use the same background and overset meshes in their analysis.

Chapter 4

OVERSET MESH APPROACH

The CFD analysis performed in this study was conducted using an overset mesh approach. This chapter details the overset mesh method and its implementation in OpenFOAM v2012 and STAR-CCM+ 2021.3. There is also a brief discussion on mesh methods that have previously been used to deal with the same OWSC device. All methods are attempts to effectively simulate the very large rotations an OWSC device experiences, where typical mesh-morphing methods are no longer appropriate.

4.1 Previous Methodology

Since OWSCs experience such large values of rotation, typical mesh-morphing methods are generally not sufficient to model them. In the typical mesh-morphing method, the cells in the mesh are allowed to deform as the body moves. For small rotations or translations, this mesh deformation is small and manageable. However, for large rotations and translations, the cells will be deformed to an unacceptable level that will result in poor mesh quality and, therefore, poor results and/or solution divergence.

Imagine modeling a rigid cylinder immersed in a tank filled with water. There are polyhedral cells surrounding the cylinder. If the cylinder is allowed to move horizontally, the cells to one side of the cylinder would be forced to squash together while the cells to the other side of the cylinder would stretch to fill the void. This significantly changes the cells' volume and shape. This type of behavior was demonstrated by Jasak 2009 [45] and is shown by Figure 4.1.

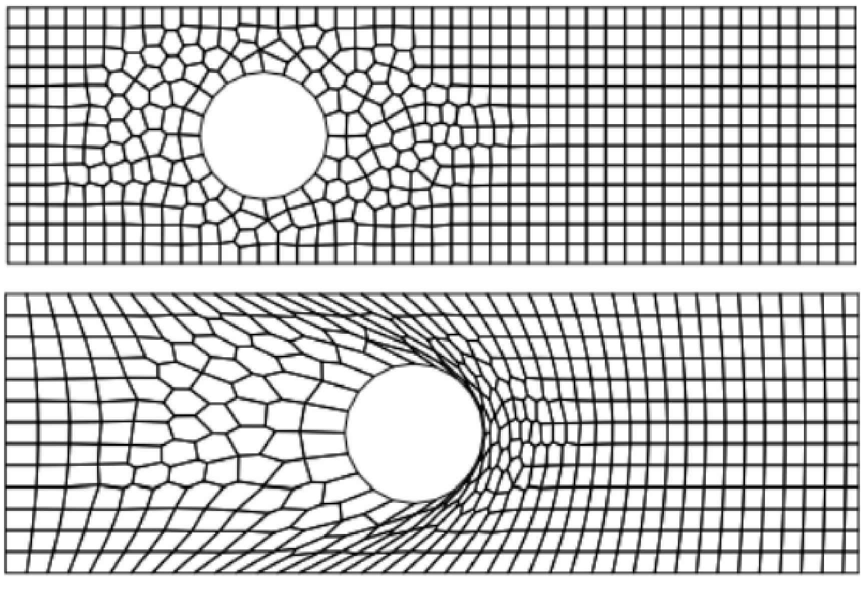


Figure 4.1: Initial mesh (top) and deformed mesh (bottom) for a moving cylinder. Figures courtesy of [45].

As the cylinder in Figure 4.1 travels across the domain, the cells that had originally been to its right will eventually be forced to essentially have zero or near-zero volume and result in solution divergence; though, more likely, the mesh quality would have deteriorated far enough for solution divergence prior to cells approaching too closely to zero volume. The same type of problem occurs for an OWSC device experiencing large rotations. As the device rotates to one side, the mesh cells on that side are forced to contract and form severely deformed shapes. Mesh quality metrics – skewness and non-orthogonality – increase to unacceptable values, and the simulation will fail. Cell skewness and non-orthogonality are two metrics for assessing mesh quality. Skewness between cells is defined as the distance from the intersection of adjacent cell faces and a line connecting cell centers to the midpoint of the adjacent cell faces; in a high-quality mesh this value should be small or zero. Non-orthogonality between cells is defined as the difference between 90 degrees and the angle between a vector normal to the adjacent cell faces and a line connecting the cell centers; ideally, this value should also be zero or small [1].

Two other CFD mesh approaches that deal with this issue were mentioned in Chapter 1 for use in modeling a OWSC device in a wave tank. These were the re-meshing approach and the AMI approach. In particular, this section examines the CFD simulations conducted by Winter & Motley 2020 [1] using a re-meshing approach and Schmitt and Elsaesser 2015 [5] using an AMI approach.

A re-meshing approach was implemented by Winter & Motley [1] using OpenFOAM v7 and modeled the same experimental set-up detailed in [5]. In this model, a simulation was run using the *interFoam* fluid solver and *sixDoFRigidBodyMotion* body motion solver where mesh-morphing occurred as the OWSC rotated. An initial model was set up with the wave tank mesh, the upright OWSC body, and the initial water level; the simulation was run using *interFoam*. At every 0.01 second time step, the reconstructed mesh quality was checked using the *checkMesh* utility. If the mesh met the quality standards set in the *meshQualityDict*, the simulation would continue running *interFoam*. When mesh quality deteriorated and the standards set in the *meshQualityDict* were not met, the simulation would stop, and a new mesh would be generated with the OWSC now starting in the same position it stopped in. The OWSC position for the newly generated mesh had to be calculated from the output from the previous mesh simulation and then incorporated into the new mesh using a Python script, which is to say it is not automatically done by OpenFOAM. Once re-meshing for the new position was completed, the simulation was started again [1].

This method requires many stops, starts, mesh checks, and re-meshing operations, which all increase the overall run time of the simulation. The re-meshing approach also requires that all field variables be interpolated between the previous mesh and each new mesh, introducing additional numerical error to the solution. A more complex meshing utility – *snappyHexMesh* – is required to generate a suitable mesh near the OWSC, which will take longer than a simple *blockMesh* alone. *snappyHexMesh* is a utility designed to ensure mesh quality. It functions by first creating a basic mesh, typically using *blockMesh*, and then incorporating tessellated geometries into this basic mesh near the body [46]. In this way, every new start point in the re-meshing scheme would begin again with a high-quality mesh.

The arbitrary mesh interface (AMI) approach was implemented by Schmitt and Elsaesser (2015) using an unspecified version of OpenFOAM [5]. This AMI method uses a sliding mesh interface where a cylindrical mesh centered at the OWSC hinge is allowed to rotate about this center while the wave tank mesh remains stationary. No mesh-morphing occurs in this method and so re-meshing is not necessary. The wave tank mesh and the rotating cylindrical mesh do not overlap each other; instead, they exchange information via interpolation at this sliding interface. Due to the geometry of the OWSC and wave tank, a cylindrical mesh with a center of rotation located at the OWSC hinge will extend past the wave tank floor boundary. Thus, to use this method, a non-physical fluid region must be introduced beneath the OWSC device large enough to contain the cylindrical AMI mesh [5]. The effects of the non-physical fluid region on the solution are negated using a dissipation parameter that was incorporated into the *interFoam* solver [1].

Like in the re-meshing method, the AMI method introduces additional numerical error during the interpolation of field variables at the interface between the meshes. It also requires the user to modify a default OpenFOAM solver. Schmitt and Elsaesser demonstrated that the dissipation was required to be in the 50 to 100% range to adequately reduce fluid flow velocities in the non-physical fluid region. However, the dissipation level necessary to minimize solution errors has not been extensively studied and may vary depending on the case [1].

The AMI method is similar to the overset method used in this study in that it also utilizes two meshes that do not experience any mesh-morphing. This means that mesh quality does not deteriorate from the original mesh state. Both methods use interpolation between separate mesh regions and incur resulting numerical error. However, an advantage of the overset method is that the overset mesh may be an arbitrary shape; it need not be a cylinder. Thus, no non-physical fluid region or dissipation parameter need to be implemented. The results from the re-meshing and AMI methods are compared to that of the overset method used in this study in Chapter 5. The overset method is detailed below.

4.2 Overset Grid Methodology

The overset mesh method uses two or more non-deforming grids: a background mesh and at least one overset (or body-fitted) mesh. In the case of rigid body motion, no mesh deformation occurs in any of the grids and the original meshes are maintained. This prevents mesh quality concerns due to mesh-morphing and is especially convenient for modeling large translations and/or rotations. These grids arbitrarily overlay each other, moving relative to one another and interpolating information between grids.

The OpenFOAM User Guide [47] gives a good example of how the grids are set up. The overset mesh method requires a background mesh. For this study, the background mesh consists of the wave tank detailed in Chapter 3. However, for demonstrative purposes, a generic background mesh shown in Figure 4.2 is given.

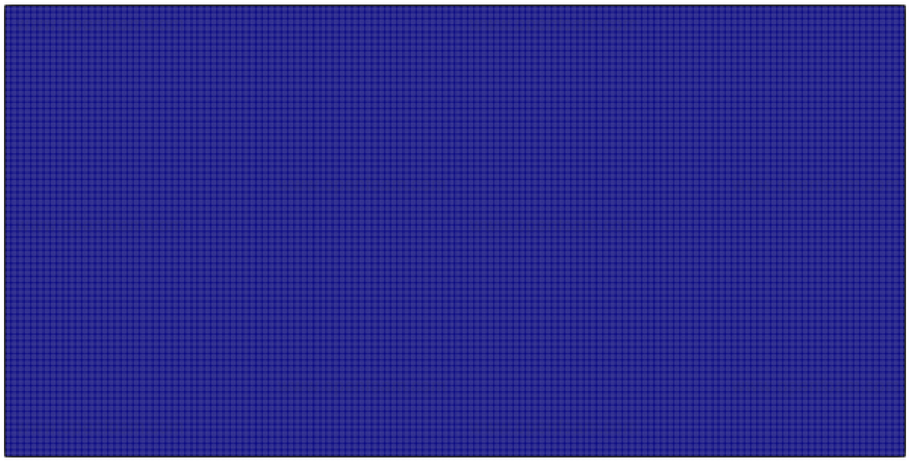


Figure 4.2: A generic background mesh. Figure courtesy of [47].

The OpenFOAM example uses the case of two rotors rotating near one another. In this case, two overset meshes must be used. It becomes apparent why overset meshes are also referred to as body-fitted meshes, since they are fitted around the bodies they move together with. The overset meshes are shown in Figure 4.3.

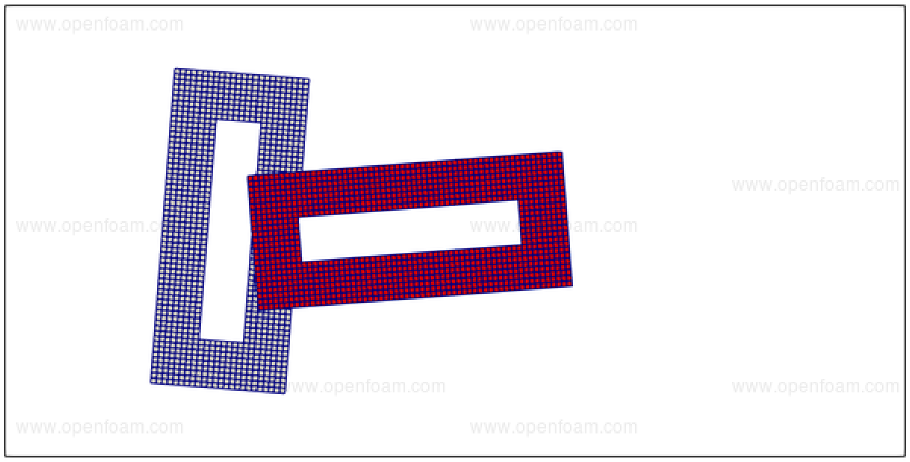


Figure 4.3: Two overset meshes laid over a background mesh. Figure courtesy of [47].

The empty white spaces enclosed by the overset meshes in Figure 4.3 are where the solid bodies (rotors) are located. The cells taken up by the body are called hole cells. Hole cells are blocked out during the simulation and, thus, inactive [47].

The other cell types that must be defined in an overset mesh method are donor and acceptor cells. The donor cells provide flow field values while acceptor cells get their values from interpolation. Acceptor cells are also known as "fringe" cells since they are located at the boundaries or "fringes" of overset meshes. The corresponding donor cells are located adjacent to the acceptor cells, which consist of adjacent overset cells and adjacent background mesh cells, respectively. Interpolation is performed between the donor and acceptor cells [47].

The overset mesh used in this study is shown in Figure 4.4. This figure does not show the entire background mesh; it shows a zoomed in portion near the OWSC.

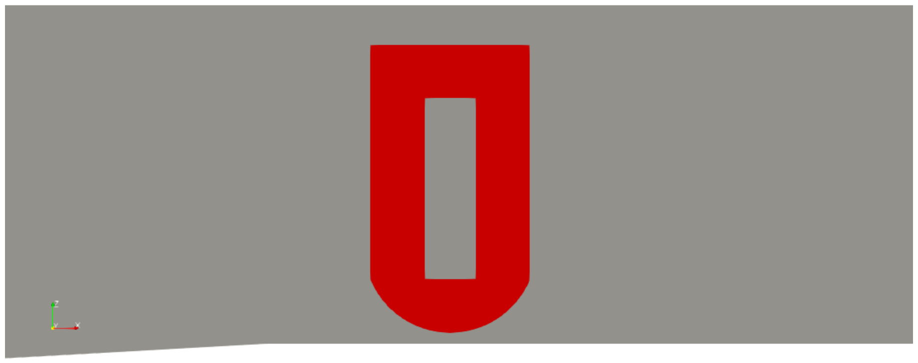


Figure 4.4: OWSC overset mesh generated in the ParaView [44] software.

In Figure 4.4, the grey area is the background mesh and the red area is the overset mesh. The grey area enclosed by the red overset mesh is where the OWSC device is located and contains the hole cells. Interpolation occurs between the background and overset meshes at the boundary of the overset mesh. This interpolation must occur at every time step and adds to computational cost. It can also lead to conservation and convergence issues [48].

To prevent such issues and limit computational error, consideration should be given to both the overset mesh used and the background mesh near the body, in particular. Cell sizes of the background and overset grids should be similar to prevent large interpolation errors. Additionally, at least two acceptor cell layers are needed to adequately represent gradients [49]. Another way to decrease interpolation errors is to use better, typically more complex, interpolation methods [49]. However, these improved interpolation methods can slow down the simulation significantly.

4.2.1 OpenFOAM v2012

Interpolation Method

The OpenFOAM solver, *overInterDyMFoam*, handles overset cell assignments and interpolation. The interpolation method is set in the *fvSchemes* dictionary. For this study, the inverse distance (*inverseDistance*) interpolation method was used. First, OpenFOAM assigns cells as Calculated (donor cells), Interpolated (acceptor cells), or Hole (hole cells) and a cell stencil is constructed. The inverse-distance-weight interpolation stencil finds and marks all boundary/patch faces, marks any cells overlaying these faces, and uses a flood filling algorithm to determine the unreachable (hole) cells [50]. Flood filling algorithms are commonly used in computer graphics to color pixelated enclosed areas. In this case, the algorithm is used to find hole cells enclosed by a boundary. To begin, all cells within the mesh domain are marked as calculated cells. Boundary patches (walls, like the OWSC, and overset patches) are marked and assigned as hole cells. The flood filling algorithm in OpenFOAM is a method of determining which cells lie inside patch boundaries by assigning regions and investigating where blockages occur when "walking" cell-by-cell through the domain. In the case of the overset mesh, the hole cells lie inside and around the OWSC body patch; meanwhile, the background mesh hole cells include all cells located on and inside the overset patch. Interpolation cells are marked as those cells in the overset mesh which border the overset boundary [51]. The hole-cutting process is one of the things that can substantially slow down overset simulations.

The inverse distance method using weighted values of the donor cells to calculate the values on the acceptor cells, as shown by Equation 4.1 [51].

$$x_{interpolated} = \sum W_i x_{i,donor} \tag{4.1}$$

where W_i stands for the weight for each donor cell, i, and the x's represent some field value being interpolated. The weights are determined using the inverse of the distance between the cell center of the donor cell to the cell center of the acceptor cell, Equation 4.2 [51].

$$w_k = \frac{1}{d_k} \tag{4.2}$$

These are summed together over all donor cells surrounding the acceptor cell of interest. The weight for each individual donor cell can then be determined via Equation 4.3 [51, 48].

$$W_i = \frac{w_i}{\sum w_k} \tag{4.3}$$

The inverse distance interpolation scheme is simple and straightforward to implement. However, it has lower accuracy than some other interpolation methods, the degree of which is dependent on how donor cells are distributed and the width of the stencil [49]. It is also significantly faster than other interpolation methods, including the least-squares method. Increasing the resolution of the background and overset meshes has the advantage of also increasing the accuracy of this interpolation method. For fine enough meshes with well-matched cell sizes at and around overlaying grids, the inverse distance interpolation method provides sufficient solution accuracy compared to more sophisticated methods [49].

Overset Simulation Set-Up

To run an overset mesh simulation in OpenFOAM v2012, a specific file structure must be used. This file structure is shown in Figure 4.4. The background and overset meshes require their own folders for block-meshing purposes.

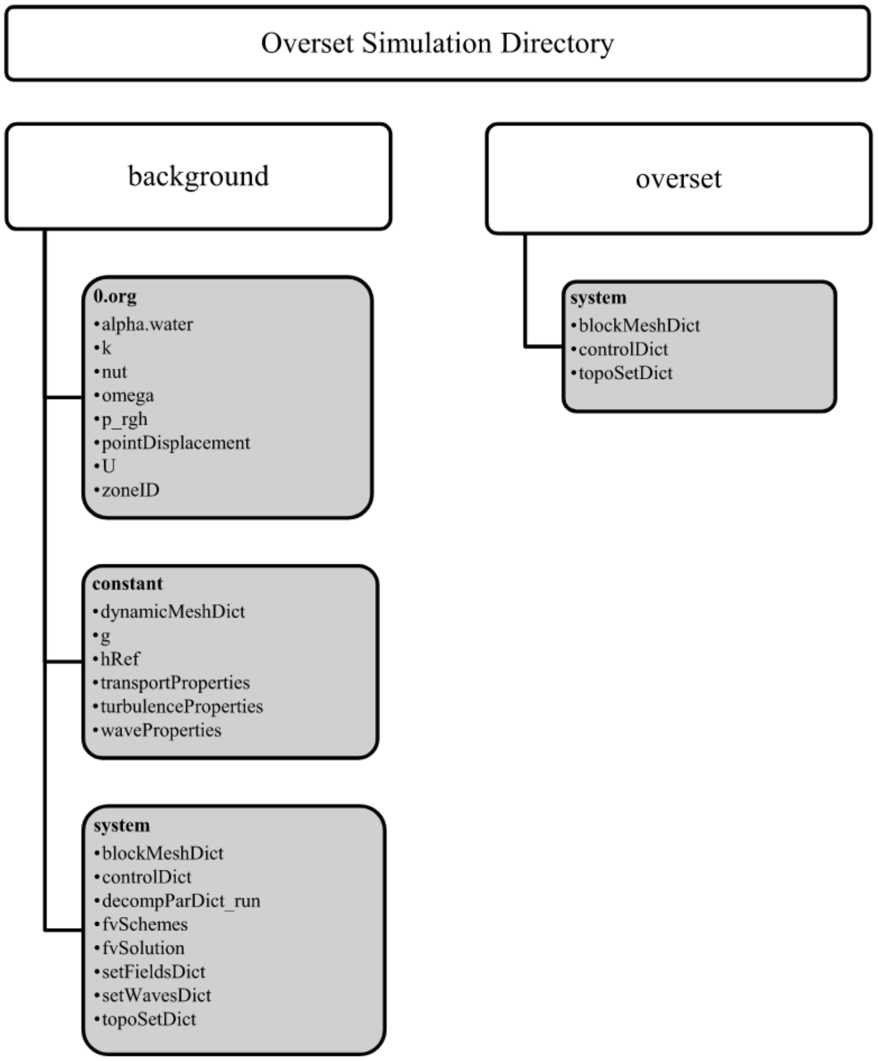
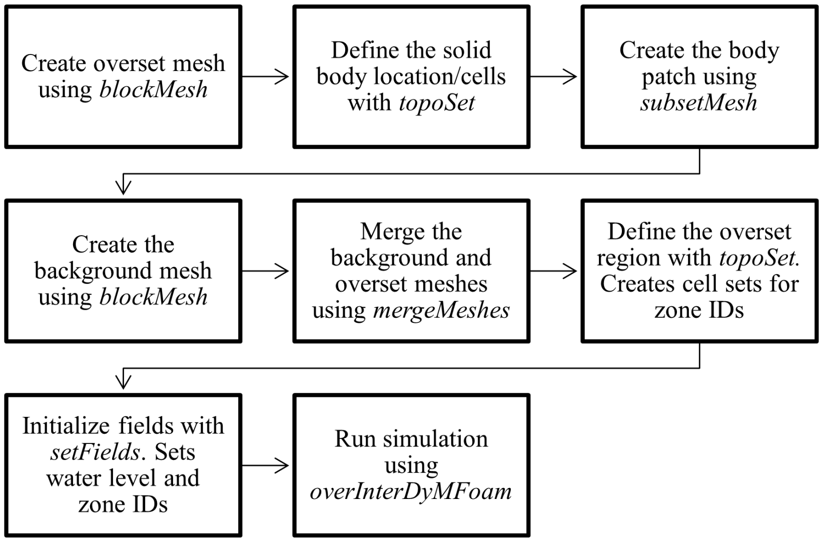


Figure 4.5: OpenFOAM file structure for overset grid simulation.

In Figure 4.5, each box is a folder and each bullet point is a file. Since the majority of these files and folders have been discussed previously as context required, each file and its functions will not be discussed in detail here. Instead, an overview of the overset simulation set up will be outlined briefly.

Prior to running an overset simulation in OpenFOAM, the overset and background meshes must be created. In this study, both meshes were created using the *blockMesh* utility discussed in Chapter 3. The location of the OWSC body within the meshes must also be specified and a patch created. The set-up process is illustrated by Figure 4.6.



Create overset mesh using *blockMesh*
Define the solid body location/cells with *topoSet*
Create the body patch using *subsetMesh*
Create the background mesh using *blockMesh*
Merge the background and overset meshes using *mergeMeshes*
Define the overset region with *topoSet*. Creates cell sets for zone IDs
Initialize fields with *setFields*. Sets water level and zone IDs
Run simulation using *overInterDyMFoam*

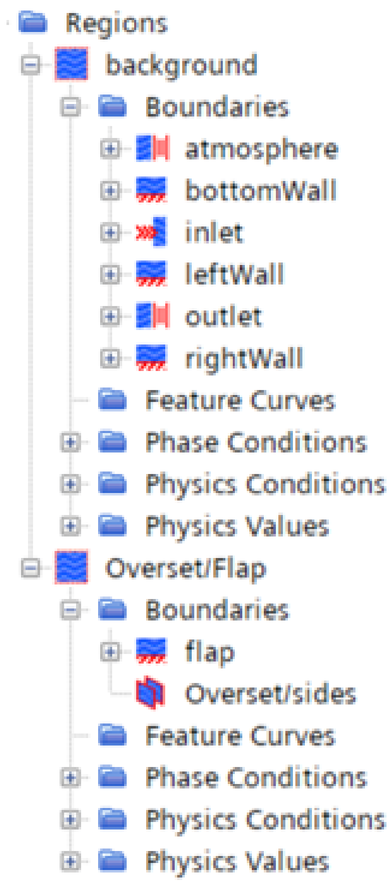
Figure 4.6: OpenFOAM overset simulation set-up process.

In this case, two zone IDs were used: one for the overset cells and the other for the background cells. These IDs are used to keep track of the two regions used in the simulation and which cells belong to which region.

4.2.2 STAR-CCM+ 2021.3

The STAR-CCM+ simulation used the Distance Weighted overset mesh interpolation option. This interpolation method computes interpolated acceptor cell values using weights based on the distances to the four nearest neighbors [36]. It is not the same as the inverse distance interpolation method used in OpenFOAM. The other available interpolation options in STAR-CCM+ are Linear, Linear Quasi2D, and Least Square. The inverse distance method is not used in STAR-CCM+ unless the interpolation is difficult and the simulation reverts to this less accurate interpolation method at times when the conditions for the selected interpolation scheme cannot be met [36].

The overset and background meshes are defined as two separate regions in the Regions branch of the simulation tree. These regions were created when the OpenFOAM overset and background meshes were imported into STAR-CCM+ as volume meshes. The overset mesh also contains the OWSC device and all associated boundaries. The Regions branch of the simulation tree used for this study is shown in Figure 4.7.



```
Regions
  background
    Boundaries
      atmosphere
      bottomWall
      inlet
      leftWall
      outlet
      rightWall
    Feature Curves
    Phase Conditions
    Physics Conditions
    Physics Values
  Overset/Flap
    Boundaries
      flap
      Overset/sides
    Feature Curves
    Phase Conditions
    Physics Conditions
    Physics Values
```

Figure 4.7: Regions branch of STAR-CCM+ simulation tree.

An Overset Mesh Interface was created between the two regions by selecting both regions in the Regions branch, right clicking, selecting Create Interface, and then selecting Overset Mesh in the expanded menu. Additionally, the overset mesh boundary was defined as an Overset Boundary in the Overset/Flap region shown in Figure 4.7.

STAR-CCM+ uses a slightly different cell naming convention than OpenFOAM. It calls both the donor cells and other computed background cells Active Cells, hole cells are called Inactive Cells, and acceptor cells are Acceptor Cells. The hole-cutting method employed is called the Layered Approach. First, a single layer of cells adjacent to the overset boundaries are marked as acceptor cells in the overset region. All other cells remain marked as active cells. The corresponding adjacent cells are marked as potential donor cells. Then, in the background region, a single layer of cells near those potential donor cells are marked as acceptor cells. This layer is located about 4 cell layers away from the overset interface between the regions. These four intermediate cell layers are labeled as donor cells in the background mesh. All background cells located inside the closed overset interface are marked as inactive and do not take part in the solution process [36].

Chapter 5

RESULTS

To examine the accuracy of the overset simulations, the OWSC motion and loading is evaluated. This chapter compares OWSC motion behavior reported by OpenFOAM to a physical experiment conducted in a wave flume. It goes on to compare OpenFOAM simulation results to an equivalent STAR-CCM+ simulation.

5.1 OpenFOAM v2012 Results

OpenFOAM data was post-processed from data output by the solver log file and using a function object called *forces* [52] in the *controlDict*. The solver log file contained information on the angular velocity of the OWSC, the orientation of the OWSC, and the center of mass location for every time step. The *forces* function object generates force and moment data for a specified surface. In this case, the OWSC surface patch was specified, and the function created two files where it stored forces and moments on the OWSC: force.dat and moment.dat. Each .dat file saves the force components in the x, y, and z directions split into categories of total, normal pressure, and tangential viscous forces.

The origin of the coordinate system used for all simulations detailed herein is located at the center of the OWSC hinge. The x-axis points in the same direction as the driving waves, extending the length of the wave tank from inlet to outlet, and shall be referred to as the longitudinal direction. This axis is located at the centerline of the wave tank with a coordinate value of 0 at the hinge. A negative longitudinal coordinate indicates a location on the inlet-side of the OWSC and a positive longitudinal coordinate indicates a location on the outlet-side of the OWSC. The y-direction runs parallel to the OWSC hinge and transverse to the wave tank; this direction shall be referred to as the transverse direction. The z-axis points in the opposite direction of gravity with a coordinate value of 0 at the hinge; the z-direction shall be referred to as the vertical direction.

The rotation of the simulated OWSC device over the course of approximately 36 seconds is shown in Figure 5.1. Positive values of rotation indicate the OWSC is displacing in the negative longitudinal direction and negative values of rotation indicate the OWSC is displacing in the positive longitudinal direction.

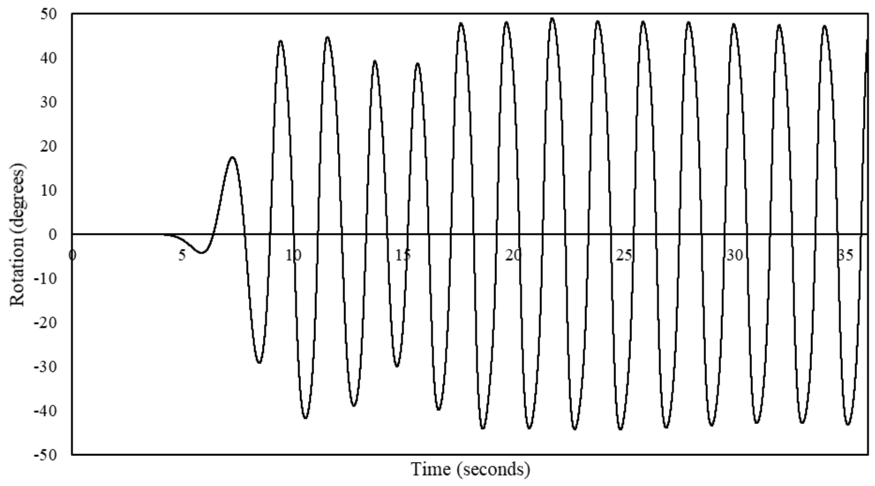


Figure 5.1: OWSC rotation results from OpenFOAM.

From Figure 5.1, the rotation stabilizes around the 20 second mark. A close-up view of a full oscillation of the OWSC is shown in Figure 5.2.

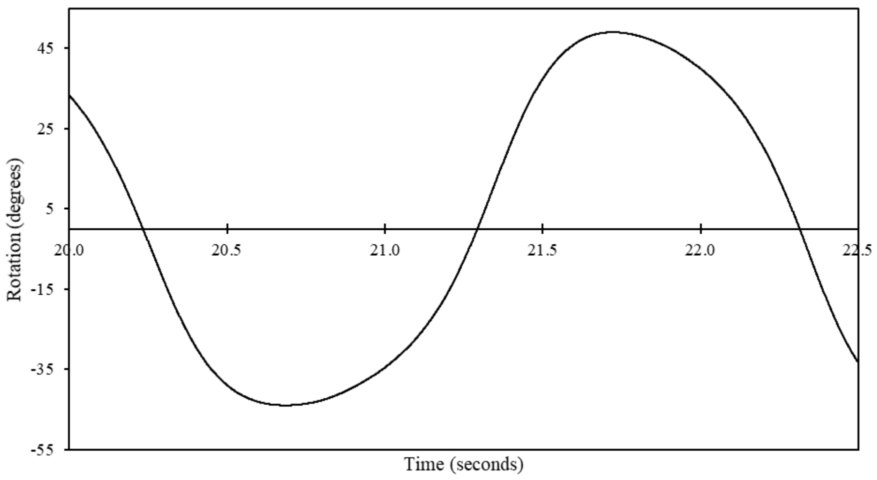


Figure 5.2: One wavelength of OWSC rotation results from OpenFOAM.

Over the course of the approximately 36 second simulation, the maximum rotation reached was approximately +49 degrees and the minimum rotation reached was approximately -44 degrees. The OWSC position at different times during a full rotation were visualized using the ParaView [44] software and are shown in Figure 5.3. In particular, the rotations at 21.65 and 21.9 seconds are shown, the values for which can easily be seen in Figure 5.2.

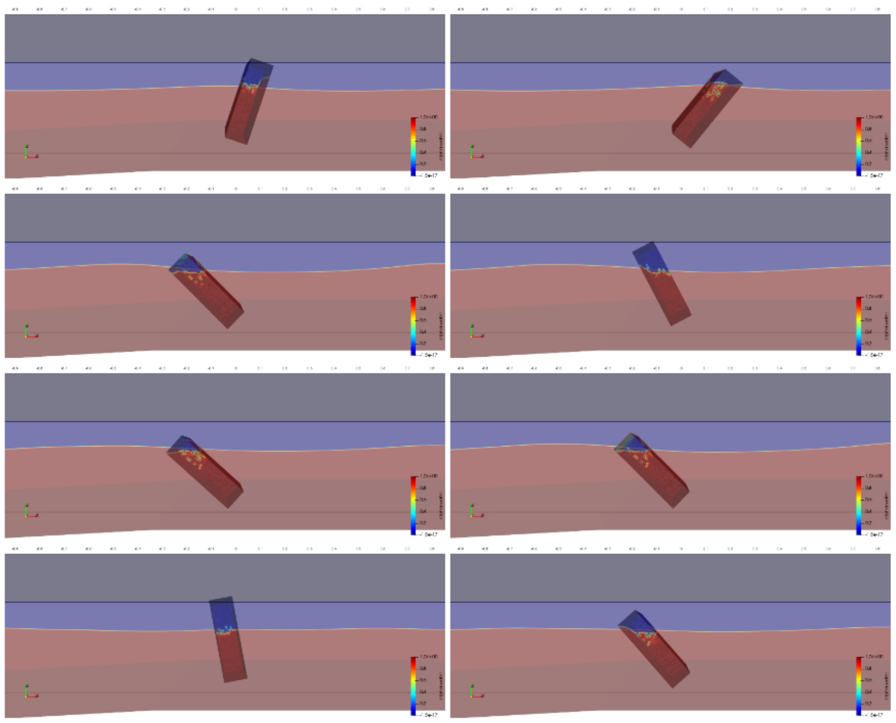


Figure 5.3: OpenFOAM OWSC rotation for the following times (read left to right and top to bottom): 10.15, 10.4, 17.75, 18.0, 21.65, 21.9, 35.75, 35.95 (seconds).

Figure 5.3 shows the OWSC device in a partially-transparent rendering of the volume fraction scalar field, where red indicates water and blue indicates air. All OWSC images were taken at a cut down the centerline of the wave tank. Rotations are measured with respect to the vertical axis.

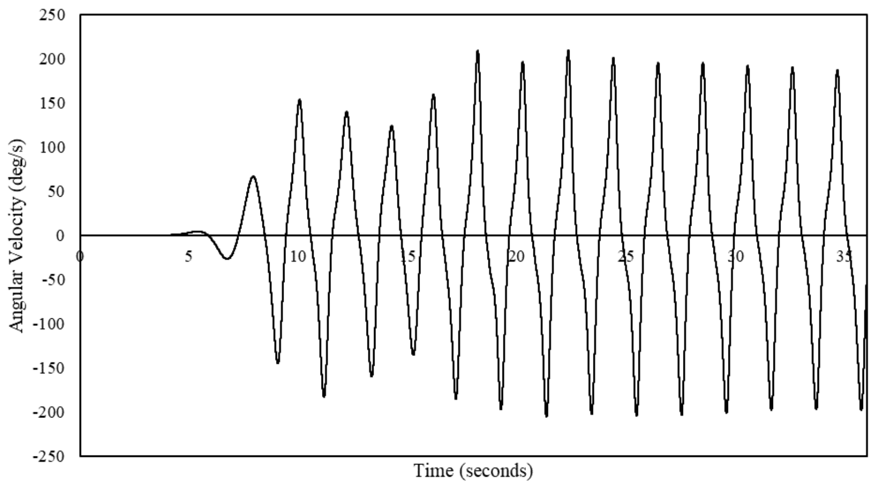


Figure 5.4: Angular velocity of the OWSC from OpenFOAM.

To better understand the behavior the OWSC, the angular velocity is compared to the OWSC rotation in Figure 5.5.

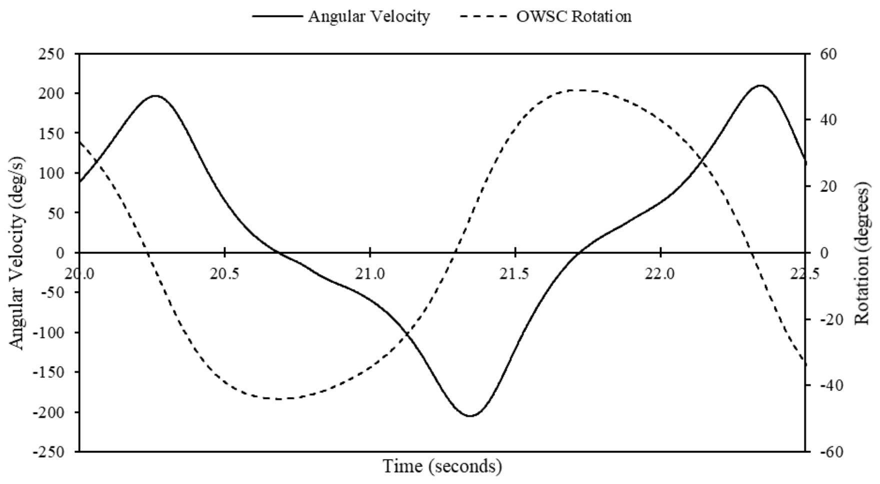


Figure 5.5: OWSC angular velocity and rotation from OpenFOAM.

Figure 5.5 illustrates how the angular velocity decreases in magnitude and flattens out near the maximum and minimum values of rotation. The extreme values of angular velocity occur when the OWSC returns to the vertical position.

Angular acceleration is not computed directly by OpenFOAM or reported in the log file. Instead, the angular acceleration was computed using Equation 5.1.

$$\alpha_{i+1} = \frac{\omega_{i+1} - \omega}{t_{i+1} - t_i} \tag{5.1}$$

where ω is angular velocity and t is time. The resulting angular acceleration values are shown in Figures 5.6 and 5.7.

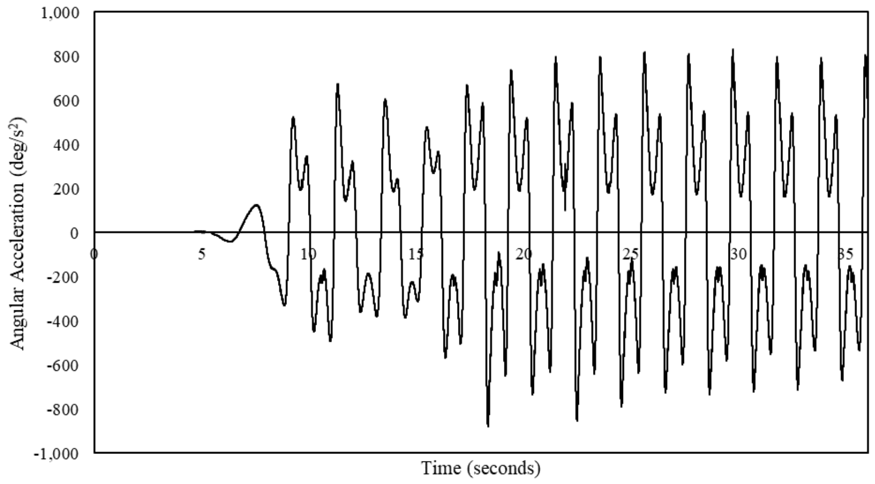


Figure 5.6: Angular acceleration of OWSC from OpenFOAM.

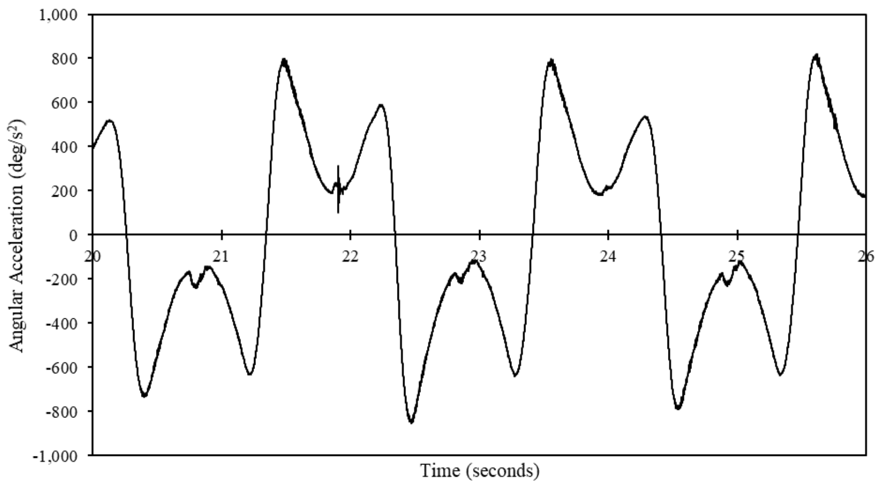


Figure 5.7: Close-up of OWSC angular acceleration from OpenFOAM.

The longitudinal, transverse, and vertical components of force on the OWSC are shown in Figure 5.8 as well as the total magnitude of the force (total).

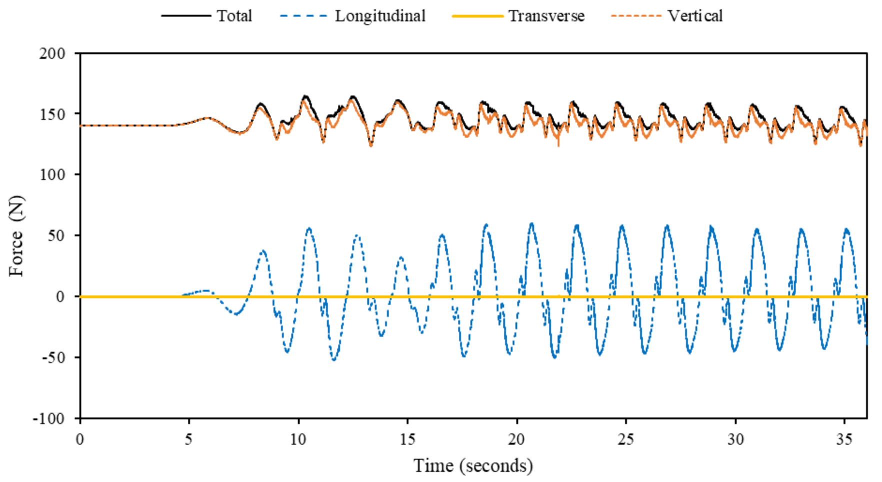


Figure 5.8: Force components on the OWSC from OpenFOAM.

The forces in the vertical direction are primarily buoyancy and gravity forces. The large value of initial vertical force is due to the buoyant force on the OWSC.

For larger values of rotation, there is also a portion of the vertical force due to the weight of the water pressing downwards on the OWSC. The forces in the transverse direction are primarily viscous and friction forces and are negligible. The forces in the longitudinal direction are the forces that act in the same direction as the driving waves and are the primary forces of interest. The longitudinal forces are shown exclusively in Figure 5.9 and 5.10.

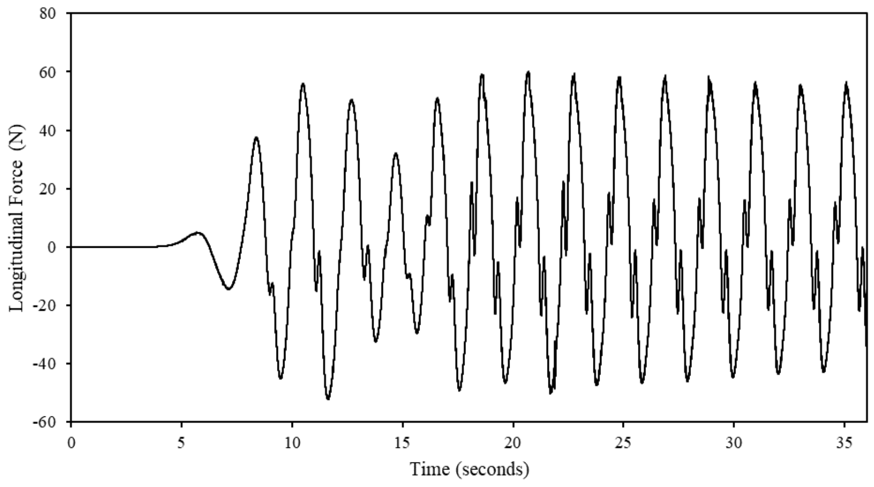


Figure 5.9: Longitudinal (x-direction) force on the OWSC from OpenFOAM.

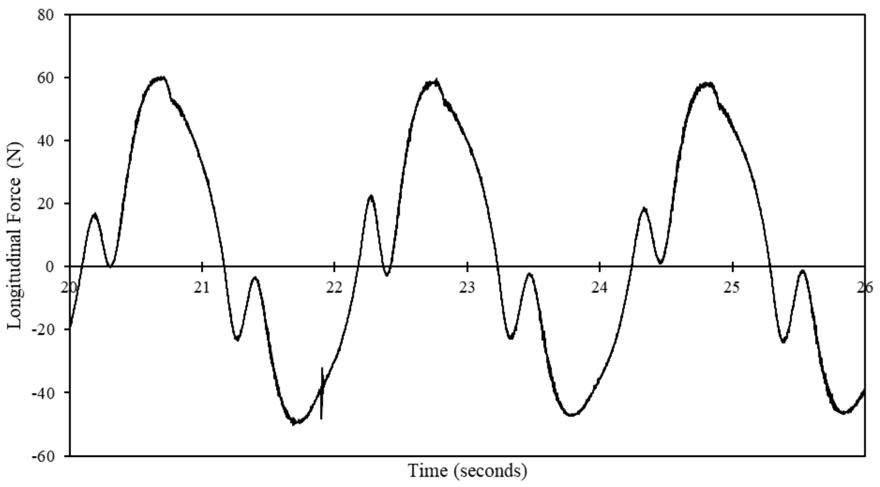


Figure 5.10: Close-up of longitudinal (x-direction) force on the OWSC from OpenFOAM.

To better understand the loading behavior, the longitudinal component of force on the OWSC is compared to the rotation of the OWSC in Figure 5.11.

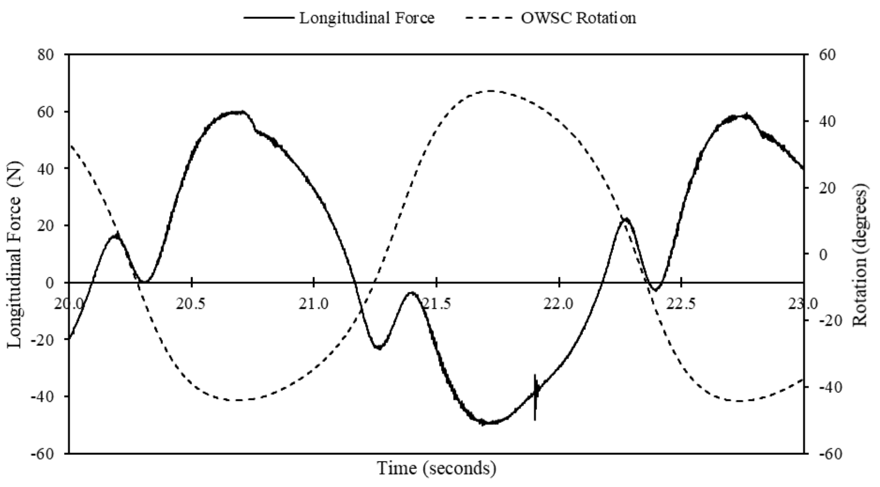


Figure 5.11: Longitudinal force (x-direction) on the OWSC and rotation from OpenFOAM.

The largest values of force in the direction of wave motion occur when the

OWSC is at the maximum and minimum values of rotation. As the OWSC returns to the vertical position, the longitudinal force decreases in magnitude until hitting zero, then begins increasing in the opposite direction. However, this increase experiences a dip very near the vertical position where the magnitude of the force approaches zero before increasing rapidly once more. These dips do not start occurring until further along in the simulation when the waves are hitting at regular intervals, reflections are occurring, and the OWSC behavior has stabilized. When the first wave reaches the OWSC, the longitudinal force is a smooth curve without any such dips. Thus, the dips observed in Figure 5.11 are likely primarily due to the effects of wave reflection.

The moments acting on the OWSC about the longitudinal, transverse, and vertical axes are shown in Figure 5.12.

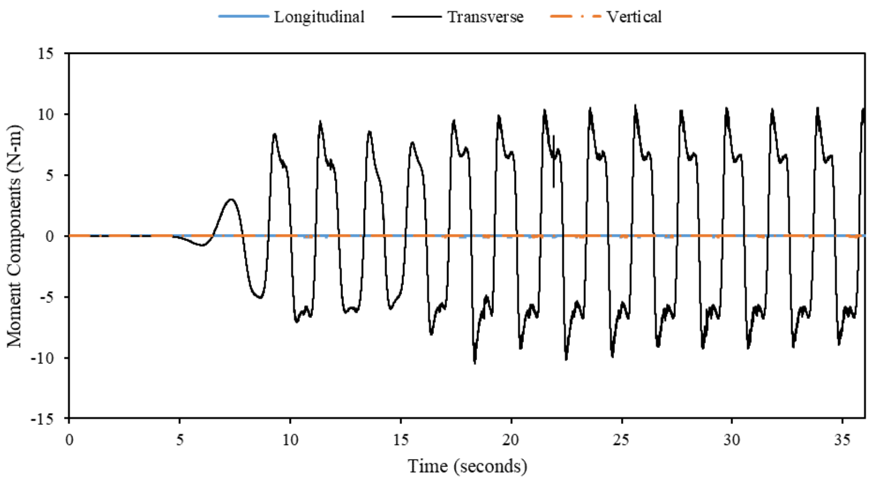


Figure 5.12: Moment components on the OWSC from OpenFOAM.

The moments about the longitudinal and vertical axes are the moments acting perpendicular to the axis of OWSC rotation. These components are negligible. The moment about the transverse axis is the moment that acts on the axis of rotation and is the primary moment of interest. The transverse moment is shown exclusively in Figure 5.13.

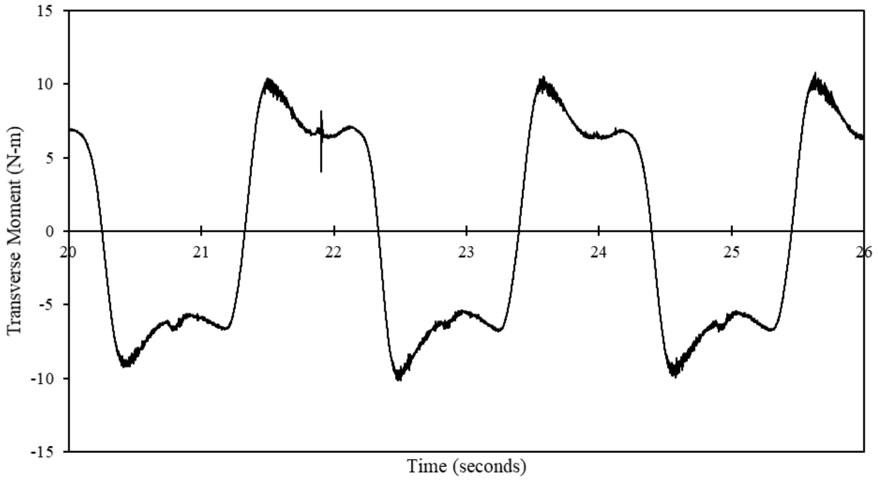


Figure 5.13: Close-up of moment about the axis of rotation on the OWSC for OpenFOAM.

To help better understand the moment loading behavior, the moment about the axis of rotation on the OWSC is compared to the rotation of the OWSC in Figure 5.14.

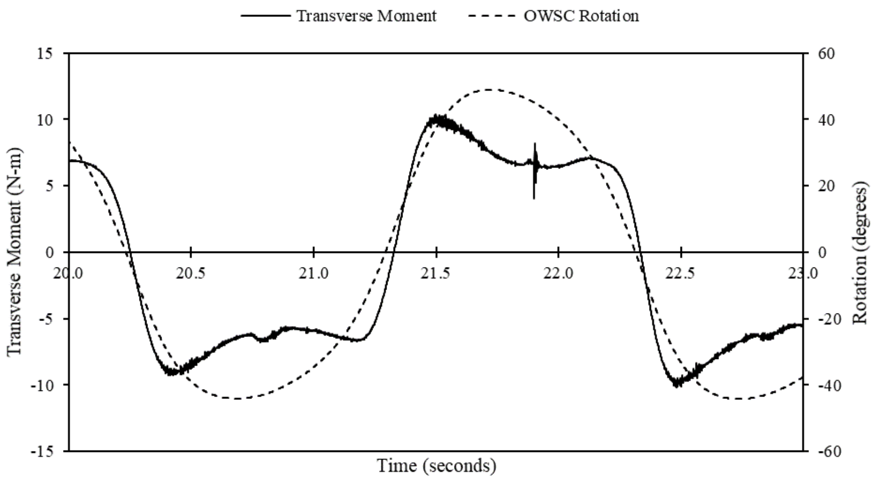


Figure 5.14: Moment (y-direction) on the OWSC and rotation from OpenFOAM.

5.2 OpenFOAM Overset Comparison to Results by Schmitt & Elsaesser 2015

The numerical OpenFOAM results were compared to the results recorded by Schmitt and Elsaesser 2015 [5] for model investigative purposes. However, as was noted earlier, the OpenFOAM simulation uses a higher center of mass and moment of inertia than that used in [5]. Additionally, the initial free surface height in the OpenFOAM model used in this study is 0.01 meter higher than that of the Schmitt and Elsaesser experiment. Due to these model discrepancies, the following comparisons are made more for behavior and qualitative purposes rather than precise numerical agreement.

The free surface elevations between the Schmitt and Elsaesser wave flume experiment and the OpenFOAM simulation conducted are compared in Figure 5.15. The free surface elevation reported by Schmitt and Elsaesser was measured next to the hinge (on the y-axis) 1.0 meter from the wave tank centerline (x-axis). Meanwhile, the OpenFOAM simulation's free surface elevation recorder was located on the centerline, -4.0 meters away from the OWSC in the longitudinal direction.

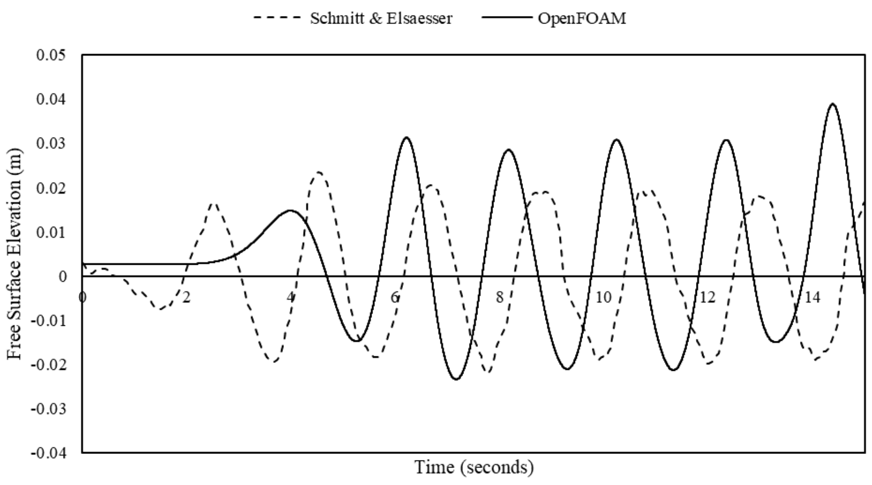


Figure 5.15: Free surface elevation changes from initial height near the OWSC.

To better compare the wave amplitude and period, the free surface elevation changes have also been plotted with the OpenFOAM data phase shifted to align with the Schmitt and Elsaesser data in Figure 5.16.

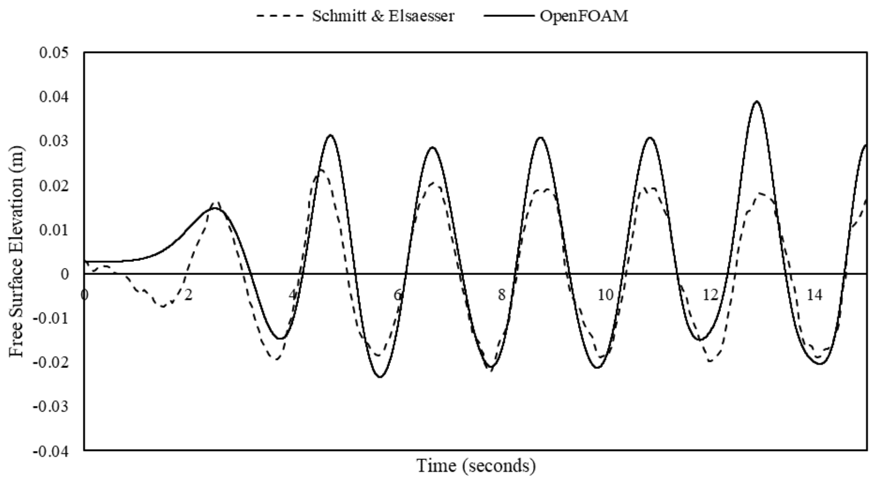


Figure 5.16: Free surface elevation changes from initial height near the OWSC with phase-shifted OpenFOAM data.

Due to the differences in the location of the free surface elevation recorders and the initial free surface elevations, it is unsurprising the free surface data sets are not exact matches. While the wave periods match up reasonably well, the free surface elevation begins to change much more quickly for the wave tank experiment than for that seen in the OpenFOAM simulation. Additionally, the wave amplitude is significantly higher for the OpenFOAM simulation, particularly the maximum positive elevations. However, the amplitude discrepancies could be due in part to the fact that the free surface recorded by OpenFOAM is 4.0 meters away from the OWSC (and closer to the wavemaker) whereas the wave tank data was recorded at the OWSC.

Schmitt and Elsaesser reported OWSC rotation values from their AMI simulation over the course of one wave period. These rotation values have been compared to those found using the OpenFOAM overset approach in Figure 5.17.

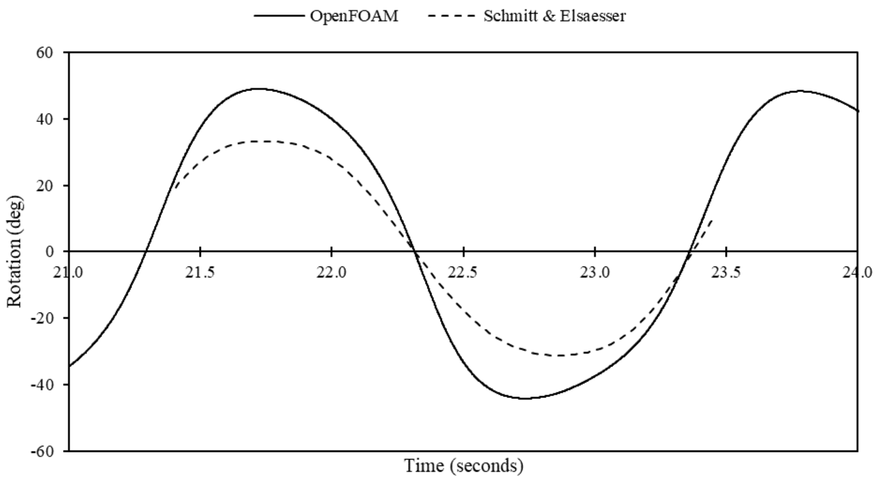


Figure 5.17: Rotation comparison of OpenFOAM overset vs. Schmitt & Elsaesser AMI.

While the periods of the OWSC oscillations match up well, the Schmitt and Elsaesser rotation data has smaller amplitudes than the overset OpenFOAM simulation. The maximum and minimum rotation angle for the Schmitt & Elsaesser data was between approximately +33.3 degrees and -31.2 degrees. This compares to the maximum and minimum rotation angle values of approximately +49 degrees and -44 degrees reached by the OpenFOAM overset simulation. This difference is likely due in part to the differences in the center of mass and initial free surface elevations between the Schmitt and Elsaesser OWSC model and the OWSC model used in this study. Additionally, it should be noted that Schmitt and Elsaesser did not report the precise time that the rotation data in Figure 5.14 was collected; it was only stated that it was taken over one wave period. Thus, the phases between the overset OpenFOAM data and the Schmitt and Elsaesser data do not necessarily match.

The center of mass discrepancy was a significant point of concern for this study. Using the center of mass value reported by Schmitt and Elsaesser [5] in an OpenFOAM simulation resulted in solution divergence issues. This was the case for multiple models including OpenFOAM v2012 overset simulations and an OpenFOAM v7 mesh-morphing with re-meshing simulation (as conducted by Winter and Motley 2020). A variety of changes to the simulation settings were looked into to avoid these divergence issues including, but not limited to, adjusting the number of inner and outer PIMPLE correctors; reducing the maximum allowable time step; changing the overset mesh dimensions, shape, and cell sizes; using the unconditionally stable Newmark-beta solver for rigid body motion; and adding in torque springs and dampers at the hinge. While some of these changes delayed the instability issue, the instability would still occur. To investigate this issue further, OpenFOAM overset simulations were conducted using another flat-bed wave tank and OWSC detailed in [38]. The waves and dimensions of the OWSC in [38] were very similar to that used in [5] and used a higher center of mass value. OpenFOAM overset simulations ran successfully for the case detailed in [38] without experiencing solution divergence issues. In lieu of the availability of better matched experimental results, the OWSC with the higher center of mass and moment of inertia detailed in [38] was simulated in the wave tank detailed in [5] and the OpenFOAM CFD results were compared to the results of [5] for qualitative and behavioral purposes. To help validate the OpenFOAM overset results, another overset CFD model was created in STAR-CCM+ for additional comparison.

Schmitt and Elsaesser conducted a wave flume scale model test of an OWSC device with an accelerometer attached to the edge of the OWSC farthest from the hinge. The tangential accelerations were reported in [5]. The tangential accelerations for the OpenFOAM overset simulation was compared to the values recorded by the accelerometer in Figure 5.18.

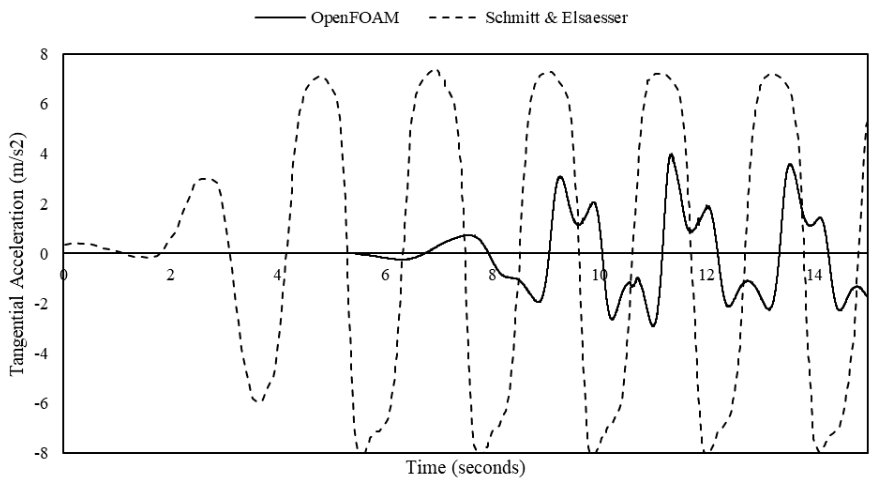


Figure 5.18: Tangential acceleration comparison of OpenFOAM overset vs. Schmitt & Elsaesser [5] accelerometer data.

The Schmitt and Elsaesser data is only available for approximately the first 15 seconds of the test. The OWSC in their experiment experiences acceleration quicker than the OpenFOAM model, which likely has to do with the differences between the OWSC's center of mass and moment of inertia. For a magnitude comparison, the OpenFOAM data was time shifted to match up with the experimental data in Figure 5.19.

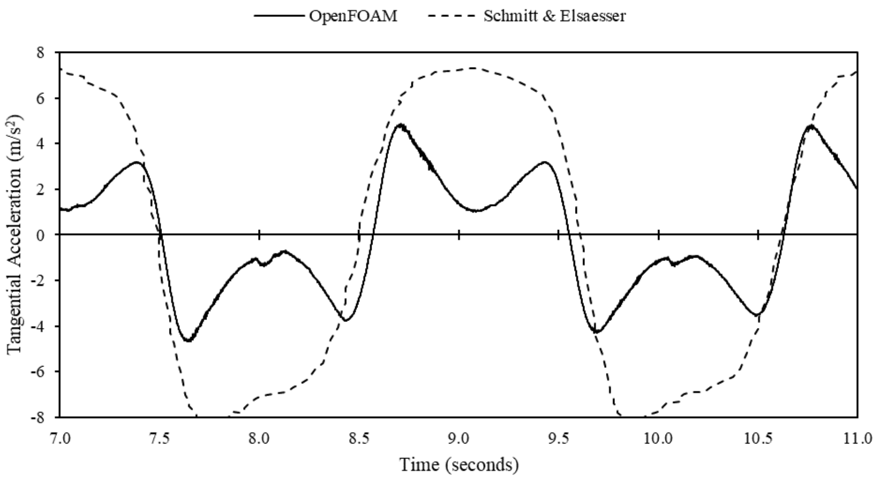


Figure 5.19: Amplitude and behavior comparison of time-shifted OpenFOAM tangential acceleration data to Schmitt & Elsaesser [5] experiment.

To better understand when the changes in acceleration are occurring, the tangential acceleration and OWSC rotation have been plotted together in Figure 5.20.

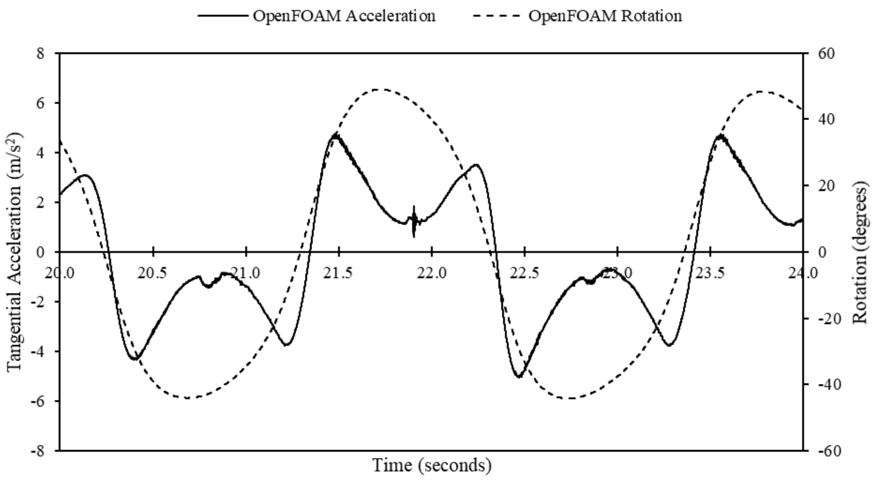


Figure 5.20: Comparison of tangential acceleration and rotation of the OWSC in OpenFOAM.

Based on Figure 5.20, the acceleration of the OWSC decreases as it approaches the maximum values of rotation in either direction. As the OWSC reverses direction, the acceleration rises again, then decreases rapidly as the OWSC approaches the vertical position. From Figures 5.18 and 5.19, it is clear that these dips in acceleration near the rotation crests do not occur in the experimental data. The troughs of the Schmitt and Elsaesser experiment do experience a small dip in acceleration near the maximum negative acceleration values, a phenomenon that was more pronounced in their numerical AMI data as well [5]. It should be noted that the numerical tangential acceleration data reported by Schmitt and Elsaesser was computed from their numerical rotation data [5].

These dips seem to be tied to the amount of damping that has been introduced to the system in order to maintain solution stability. The rigid body motion has been damped with an *accelerationDamping* factor of 0.8. As discussed in Chapter 2, this factor reduces the computed acceleration on a body proportionally to the body's acceleration magnitude, and its implementation can be seen in Equation 2.5. At higher values of acceleration, the damping factor will decrease the computed acceleration values more significantly. Similar acceleration dips near the maximum positive values of acceleration were reported by Benites-Munoz et al. 2020 [38] for an OWSC CFD model with a flat-bed wave flume (see Figure 5.21).

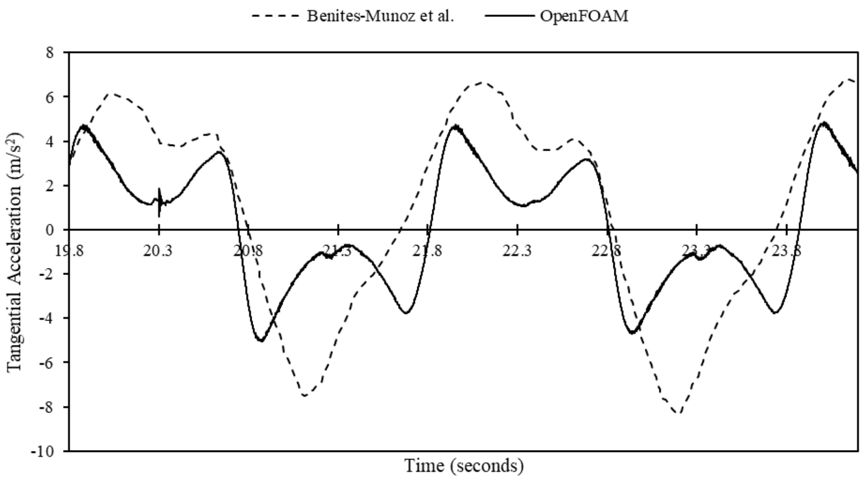


Figure 5.21: Tangential acceleration results for Benites-Munoz CFD [38] and the OpenFOAM simulation.

Note that in Figure 5.21, the OpenFOAM data has been time-shifted to better match the phase of the Benites-Munoz data.

5.3 STAR-CCM+ 2021.3 Results

The STAR-CCM+ data was collected from plots generated from corresponding monitors and reports, which were created prior to beginning the simulation. Results were compared to the OpenFOAM simulation data. STAR-CCM+ data was collected for up to 25 seconds of physical time.

Figure 5.22 compares the free surface elevations between STAR-CCM+ and OpenFOAM at a position 4.0 meters in front of the OWSC. The initial free surface elevation of the OpenFOAM simulation was 0.01 meters higher than the STAR-CCM+ simulation.

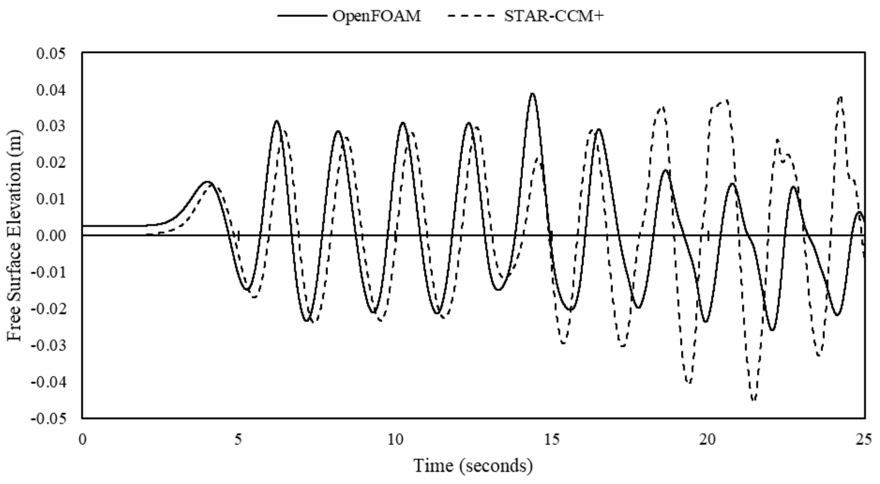


Figure 5.22: Free surface elevation changes from initial height near the OWSC for STAR-CCM+ and OpenFOAM.

From Figure 5.22, the free surface elevations match up fairly well between the two software packages in the first half of the STAR-CCM+ simulation. However, the second half data deviate significantly from each other both in magnitude and phase.

Figure 5.23 compares the OWSC rotation angles between the STAR-CCM+ and OpenFOAM simulations. Unlike the OpenFOAM data, STAR-CCM+ reports rotation in the positive longitudinal direction as positive and rotation in the negative longitudinal direction as negative. However, for a consistent comparison, the signs of the STAR-CCM+ data have been flipped in Figure 5.23. Physically, the rotation results match up as in Figure 5.23, it is simply a matter of post-processing convention.

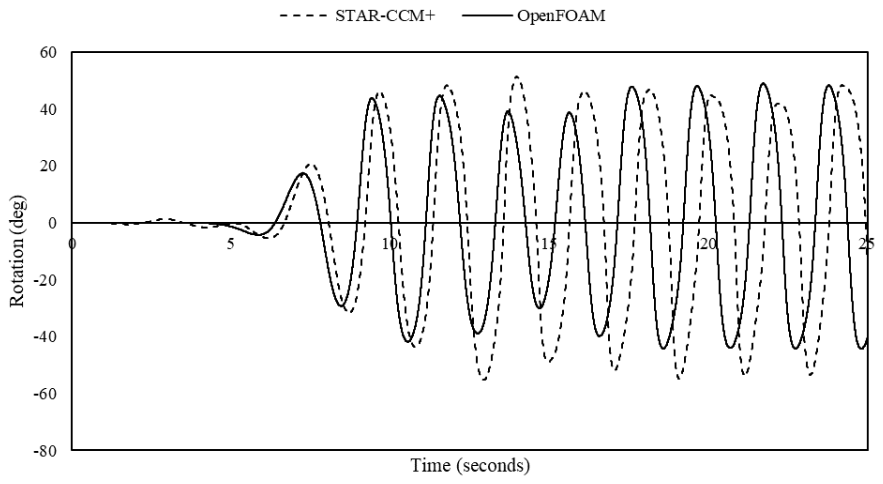


Figure 5.23: OWSC rotation comparison of OpenFOAM vs. STAR-CCM+.

The rotation angles for both simulations match up fairly well towards the beginning of the simulation, with deviation increasing as time goes on. As far as magnitude goes, the two simulations are in fair agreement; however, phases vary. The maximum and minimum values of OWSC rotation reported by STAR-CCM+ were +51 degrees and -55 degrees, respectively. This compares to the reported maximum and minimum values of rotation by OpenFOAM of +49 degrees and -44 degrees. The differences between OWSC rotation results later in the simulation are almost certainly primarily due to the differences in free surface elevations shown in Figure 5.22. While the free surface elevation in the STAR-CCM+ simulation maintained wave amplitudes around or above 0.02 meters, the OpenFOAM simulation saw amplitude decreases.

The simulation data comparisons for angular velocity and angular acceleration are shown in Figure 5.24 and 5.25, respectively.

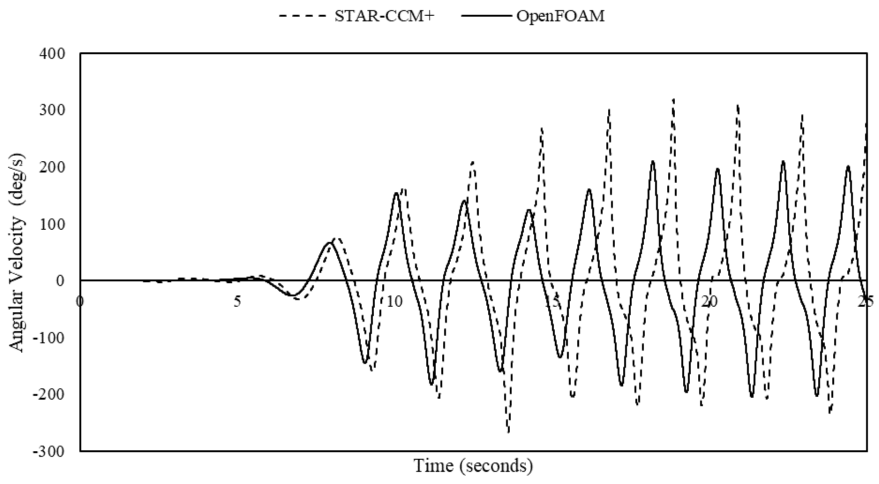


Figure 5.24: OWSC angular velocity comparison of OpenFOAM vs. STAR-CCM+.

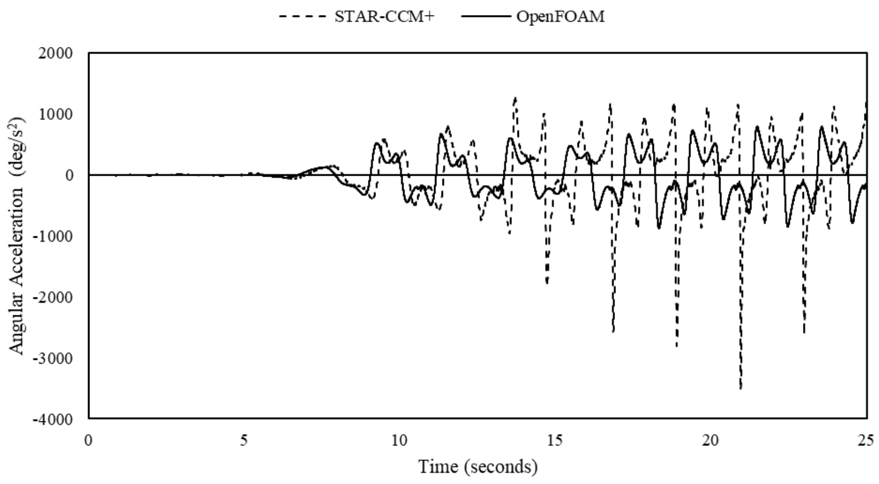


Figure 5.25: OWSC angular acceleration comparison of OpenFOAM vs. STAR-CCM+.

The STAR-CCM+ angular velocity and acceleration data display similar behavior to the OpenFOAM data, experiencing the same acceleration dipping near maximum values of rotation due to damping effects. One of the most significant differences between the OpenFOAM results and the STAR-CCM+ results is the sharp spikes in negative acceleration that occur after the OWSC returns to the vertical position and begins rotating in the direction of the driving waves.

Close-up comparisons of rotation, angular velocity, and angular acceleration are shown for a full oscillation of the OWSC in Figures 5.26, 5.27, and 5.28.

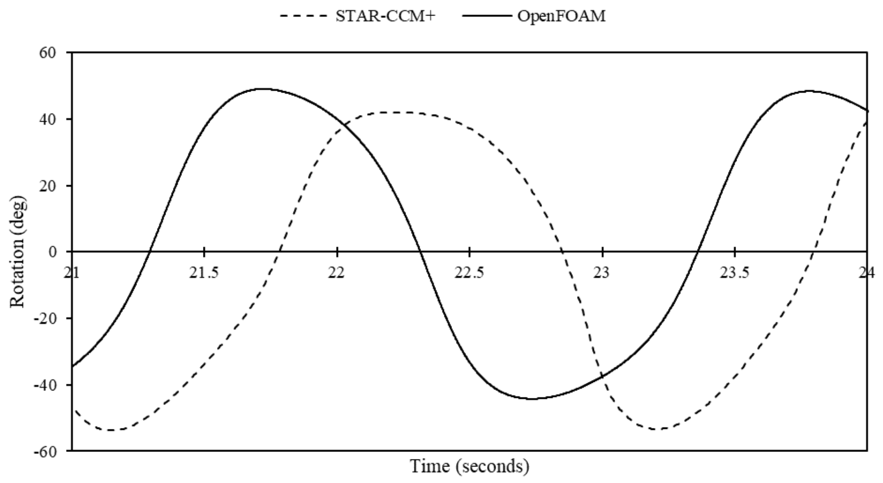


Figure 5.26: Close-up of OWSC rotation comparison of OpenFOAM vs. STAR-CCM+.

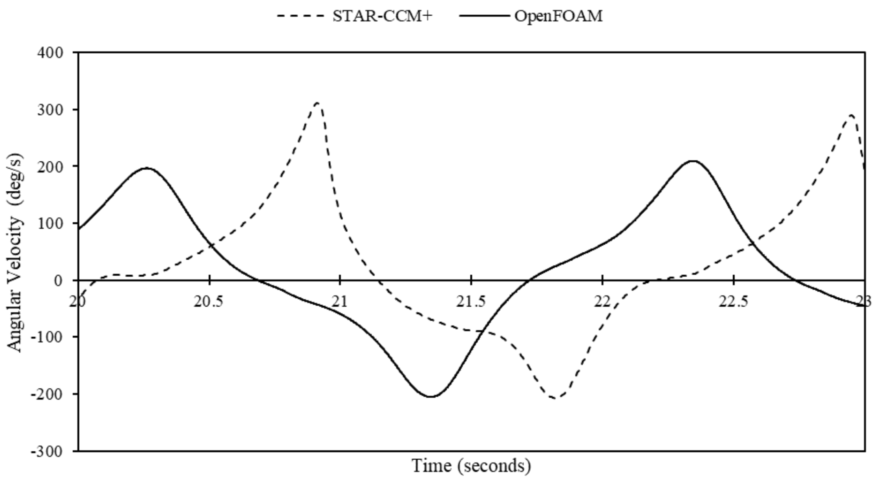


Figure 5.27: Close-up of OWSC angular velocity comparison of OpenFOAM vs. STAR-CCM+.

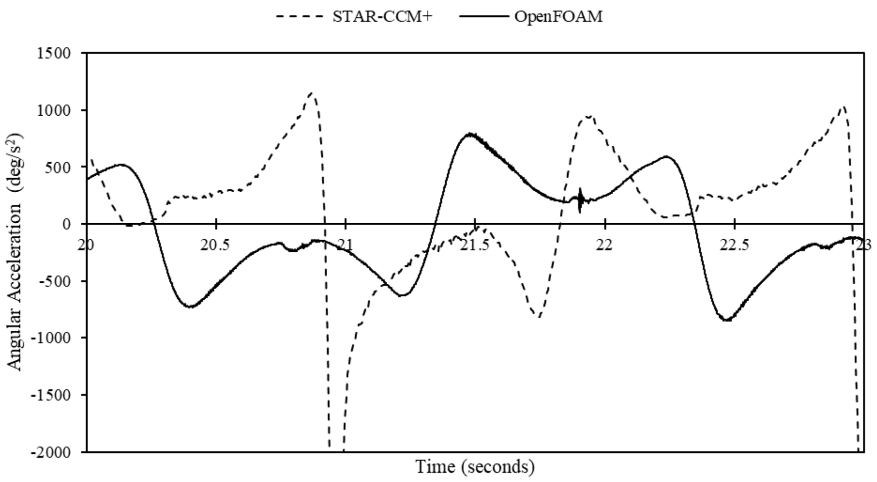


Figure 5.28: Close-up of OWSC angular acceleration comparison of OpenFOAM vs. STAR-CCM+.

The proceeding figures illustrate that the same type of motion behavior is observed in both the OpenFOAM and STAR-CCM+ CFD models. However, the STAR-CCM+ rotation results are less sinusoidal than the OpenFOAM results and, in general, the STAR-CCM+ results experience more extreme versions of the OpenFOAM behavior. Overall, the STAR-CCM+ model experiences larger rotations and higher velocities and accelerations than the OpenFOAM model. The differences are likely due in part to the difference in the initial free surface elevations and the differences in relaxation and damping factors.

Unlike OpenFOAM, STAR-CCM+ does not apply acceleration relaxation and damping factors to the rigid body motion. Instead, it applies under-relaxation factors – which function much like OpenFOAM's *accelerationRelaxation* factor – to velocity and pressure in the segregated flow solver to help maintain solution stability. STAR-CCM+ takes added mass effects into account in its momentum equation [10].

The forces and moments acting on the OWSC are compared for both software packages in Figures 5.29 and 5.30.

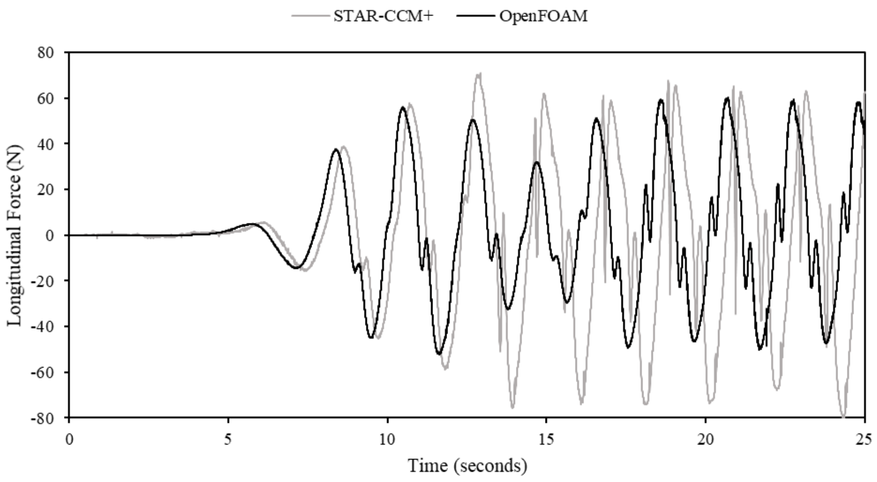


Figure 5.29: Longitudinal force on the OWSC comparison between STAR-CCM+ and OpenFOAM.

As with most of the data comparisons between STAR-CCM+ and OpenFOAM in this Section, the longitudinal force data matches up fairly well near the first half of the STAR-CCM+ simulation, after which results begin to deviate from each other. Later in the simulation, the dips in force noted in Section 5.1 are much more extreme for the STAR-CCM+ simulation than those observed in the OpenFOAM simulation. Since these dips are likely due to wave reflections, it follows that the magnitude of the dips would deviate between the simulations. As seen in Figure 5.22, the free surface elevation results begin to differ significantly between the STAR-CCM+ and OpenFOAM simulations for later times in the simulation.

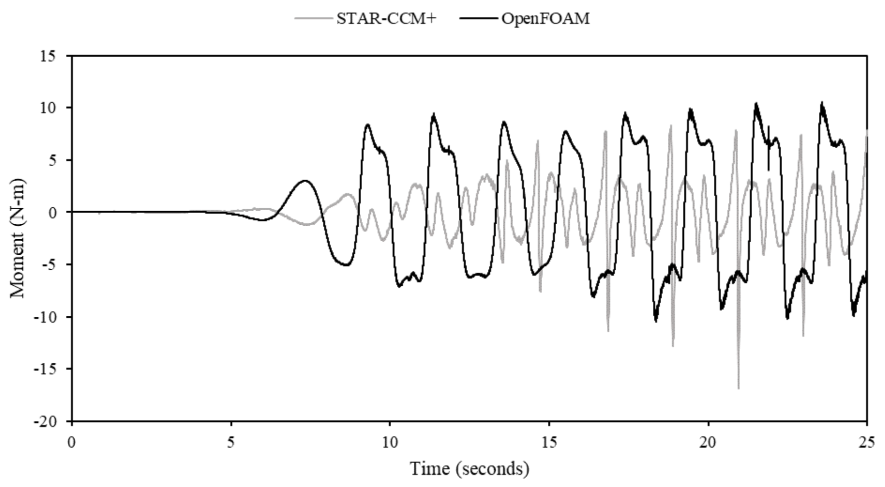


Figure 5.30: Moment on the OWSC comparison between STAR-CCM+ and OpenFOAM.

In general, the STAR-CCM+ simulation reports smaller values of moment on the OWSC than OpenFOAM. However, like with acceleration, the STAR-CCM+ moment values experience larger spikes in negative moment. Unlike most of the previous comparisons, the moment data does not match up well over the first half of the STAR-CCM+ simulation.

To better compare the moment behavior, a close-up of the moment results for STAR-CCM+ and OpenFOAM are shown in Figure 5.31. In addition, the

STAR-CCM+ data has been time-shifted so that the rotation results for the STAR-CCM+ and OpenFOAM simulations match crests.

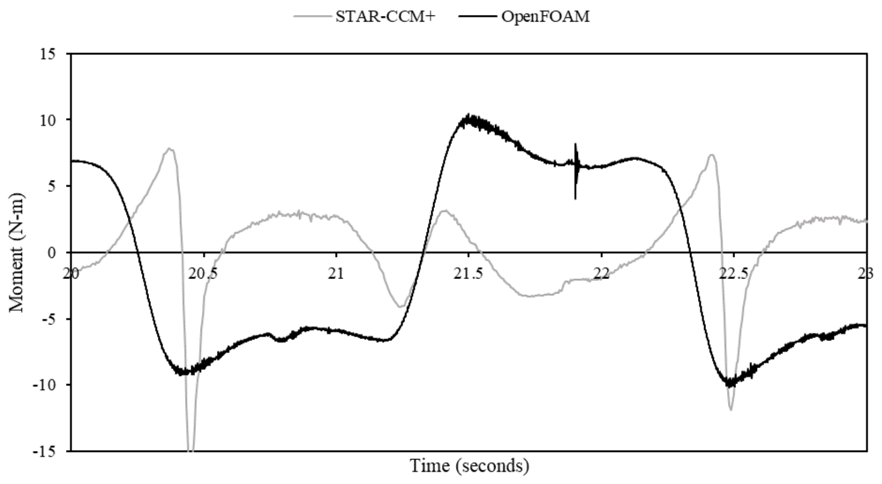


Figure 5.31: Close-up comparison of moment behavior between STAR-CCM+ and OpenFOAM.

While the same type of behavior is observed, STAR-CCM+ reports more extreme shifts in loading as the OWSC rotates.

Chapter 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

This study investigates the overset mesh approach in CFD modeling of an OWSC device. Due to the high degree of rotation experienced by these devices, typical mesh-morphing methods – the default method in most CFD software packages – are untenable. While there are a variety of methods available for dealing with large body motions, the overset method can be used without requiring any code modifications or re-meshing schemes.

Two overset CFD simulations were developed using OpenFOAM and STAR-CCM+. The results from the OpenFOAM simulation were compared to a wave tank experiment conducted by Schmitt & Elsaesser 2015 [5] and the STAR-CCM+ simulation equivalent. However, due to persistent numerical stability issues encountered in OpenFOAM when using the low center of mass reported by Schmitt & Elsaesser, the simplified OpenFOAM OWSC device was modeled after the OWSC reported by Benites-Munoz et al. 2020 [38], which had a higher center of mass and mass moment of inertia. This was a puzzling phenomenon since it was expected that a higher center of mass would decrease the stability of a buoyant body, much like in ship stability.

The maximum and minimum OWSC rotation values for the overset OpenFOAM and STAR-CCM+ simulations were approximately 1.4-1.5 and 1.5-1.8 times greater than the magnitude of the rotational values obtained by Schmitt & Elsaesser's AMI OpenFOAM simulation, with the STAR-CCM+ rotations being the largest overall. The difference in the maximum rotation magnitudes are likely due in part to the fact that the overset models both used OWSC center of mass values that were approximately two times higher than the Schmitt & Elsaesser model. However, rotational periods between the overset OpenFOAM model and the Schmitt & Elsaesser model were closely matched. The maximum and minimum OWSC rotation angles were reported for each model in Table 6.1.

Table 6.1: Maximum and minimum OWSC rotation angles.

CFD Model	Wave Opposing Rotation (deg)	Wave Direction Rotation (deg)
Schmitt & Elsaesser, AMI	33.3	31.2
OpenFOAM Overset	49.0	44.3
STAR-CCM+ Overset	51.3	55.3

The tangential acceleration values reported by the OWSC accelerometer in the Schmitt & Elsaesser experiment did not experience the same decreases in acceleration near the maximum and minimum values of OWSC rotation. These dips in acceleration occur in both the overset OpenFOAM and STAR-CCM+ simulations and are likely a result of damping that has been introduced to the system for increased solution stability.

Acceleration relaxation factors were applied to the OpenFOAM rigid body motion to take into account added mass effects not otherwise accounted for. A brief investigation into the effects of what acceleration relaxation factor was necessary for solution stability was conducted. The OpenFOAM overset simulation experienced solution divergence at approximately 10.25 seconds of physical time with an acceleration relaxation factor of 0.6, while the same simulation was run with an acceleration relaxation factor of 0.4 without encountering instability issues for 36 seconds of physical time.

In addition, based on the wave height and water depth at the wavemaker, linear wave theory is not the most applicable model. Using second-order Stokes wave theory would be more accurate and should be used in the future. [53]

This study has demonstrated that modeling an OWSC device remains a challenging CFD problem with a wide range of factors that must be considered. Key takeaways include the following:

- The overset mesh method is capable of simulating an OWSC rotating about a bottom hinge without significant code modifications or mesh quality issues due to large deformations.

- Care must be taken when creating the overset and background meshes to ensure cells sizes do not vary too significantly near the mesh interfaces. This helps ensure interpolation errors between meshes are minimized.

- While the inverse distance overset interpolation method is less accurate than more sophisticated methods, it is significantly faster, and accuracy is comparable if the previous bullet point is considered.

- The center of mass location on the OWSC device played an unexpected role in solution stability, where smaller distances between the center of mass and the rotation axis resulted in solution divergence issues.

- OpenFOAM accounts for added mass effects using an acceleration relaxation factor on the rigid body motion. This value must be low enough to prevent numerical instabilities.

- Damping introduced to the OpenFOAM and STAR-CCM+ model systems resulted in acceleration magnitude dips near the largest values of OWSC rotation that were not present in the experimental accelerometer data.

- While the OpenFOAM and STAR-CCM+ overset simulation data generally matched well in terms of behavior and magnitude near the beginning of the simulation, results began to differ more substantially later in the simulation when more complex phenomena such as wave reflections and increased turbulence effects are present.

- Additionally, differences between OpenFOAM and STAR-CCM+ results may be due to differences in how the free-surface is tracked (i.e. interface capturing methods), differences between surface tension models, and differences between wave absorption models.

6.2 Future Work

The results presented herein demonstrate that the overset mesh method is a promising CFD method for modeling OWSC devices and other WECs which experience high values of rotation. Possible extensions of this research would be to conduct a parametric study on the effects of the acceleration relaxation factor on an OpenFOAM overset simulation to determine by how much the solution is influence by this factor. Additionally, a parametric study on the acceleration damping factor would be of interest to examine the influence of this factor of the dips in acceleration observed in Figures 5.6, 5.19, and 5.28.

If possible, it would be beneficial to conduct another wave flume experiment where more experimental data could be collected for CFD comparison purposes, particularly for times greater than 14 seconds of running the wavemaker. Testing simplified OWSC devices with differing center of mass locations would be of interest as well.

The next major step in this research would be to introduce elastic deformations to the OWSC to determine the effects on body motion and the OWSC loading (forces and moments). OpenFOAM is not currently capable of modeling elastic deformations in multiphase flow, but STAR-CCM+ does support such capabilities. Using an overset mesh method with body rotations and elastic deformations would require using an overset mesh which was also capable of mesh-morphing. While the background mesh would not need to deform, the overset mesh would have to deform with the elastic body deformations. This may be achieved using the DFBI module in STAR-CCM+ with DFBI Morphing motion for the overset mesh and the Flexible DFBI Motion model [10]. So long as the elastic deformations are not overly large, the overset mesh would not encounter internal mesh quality issues.

BIBLIOGRAPHY

[1] Winter, A. O., and Motley, M. R. (2020). “Development of a fluid-structure interaction model of an oscillating wave surge converter using OpenFOAM.” *Proceedings of the 39th International Conference on Ocean, Offshore & Arctic Engineering, OMAE2020-19145.* American Society of Mechanical Engineers (ASME).

[2] Drew, B., Plummer, A. R., and Sahinkaya, M. N. (2009). “A review of wave energy converter technology.” *Proceedings of the Institution of Mechanical Engineers, Part A: Journal of Power and Energy, 223*(8), 887-902. https://journals.sagepub.com/doi/10.1243/09576509JPE782

[3] Czech, B., and Bauer, P. (2012). "Wave Energy Converter Concepts : Design Challenges and Classification." *IEEE Industrial Electronics Magazine, 6*(2), 4-16. DOI: 10.1109/MIE.2012.2193290.

[4] Thies, P. R., Johanning, L., Karikari-Boateng, K. A., Ng, C., and McKeever, P. (2014). “Component reliability test approaches for marine renewable energy.” *Proceedings of the Institution of Mechanical Engineers Part O, Journal of Risk and Reliability, 229*(5). DOI: 10.1177/1748006X15580837. https://www.researchgate.net/publication/276174922_Component_reliability_test_approaches_for_marine_renewable_energy

[5] Schmitt, P., and Elsaesser, B. (2015). “On the use of OpenFOAM to model oscillating wave surge converters.” *Ocean Engineering, 108*, 98-104. https://doi.org/10.1016/j.oceaneng.2015.07.055

[6] Whittaker, T., and Folley, M. (2012). “Nearshore oscillating wave surge converters and the development of Oyster.” *Phil. Trans. R. Soc. A., 370,* 345–364. http://doi.org/10.1098/rsta.2011.0152

[7] Biswas, R., Dunbar, J., Hardman, J., Bailey, F. R., Wheeler, L., and Rogers, S. (2012). *NTRS - NASA Technical Reports Server.* https://ntrs.nasa.gov/citations/20120016743

[8] OpenCFD (2022). OpenFOAM. OpenCFD Ltd. https://www.openfoam.com/

[9] OpenCFD (2022). OpenFOAM: User Guide v2112. OpenCFD Ltd. https://www.openfoam.com/documentation/guides/latest/doc/index.html

[10] Siemens Industries Digital Software. Simcenter STAR-CCM+ User Guide, version 2021.3. In *Theory*, pages 7763-8910. Siemens, 2021.

[11] Munson, B. R., Okiishi, T. H., Huebsch, W. W., and Rothmayer, A. P. (2013) *Fundamentals of fluid mechanics* (7th ed.). Don Fowley.

[12] OpenCFD Ltd. OpenFOAM Documentation: User Guide. OpenCFD Ltd, 2022. https://www.openfoam.com/documentation/user-guide

[13] Unofficial OpenFOAM wiki. “OpenFOAM guide/Finite volume method (OpenFOAM).” https://openfoamwiki.net/index.php/OpenFOAM_guide/Finite_volume_method_(OpenFOAM)#:~:text=OpenFOAM's%20finite%20volume%20method,can%20be%20selected%20at%20runtime.

[14] Eymard, R., Gallouet, T., and Herbin, R. (1997). “Finite volume method.” P.G. Ciarlet, J.L. Lions eds. *Handbook of Numerical Analysis, 7,* 713-1020.

[15] OpenCFD (2022). OpenFOAM: User Guide v2112. In *Files, transportModels, interfaceProperties, interfaceCompression.* OpenCFD Ltd.

https://www.openfoam.com/documentation/guides/latest/api/interfaceCompression_8H.html

[16] DeVore, R. A., Jawerth, B., and Lucier, B. J. (1992). “Surface compression.” *Computer Aided Geometric Design, 9*, 219-239. https://www.math.tamu.edu/~rdevore/publications/67.pdf

[17] OpenCFD Ltd. OpenFOAM Documentation: User Guide. In *Solution and Algorithm Control*. OpenCFD Ltd, 2022. https://www.openfoam.com/documentation/user-guide/6-solving/6.3-solution-and-algorithm-control

[18] OpenCFD Ltd. OpenFOAM Documentation: User Guide. In *Numerical Schemes*. OpenCFD Ltd, 2022. https://www.openfoam.com/documentation/user-guide/6-solving/6.2-numerical-schemes

[19] IdealSimulations. “Turbulence Models in CFD.” IdealSimulations Ltd. https://www.idealsimulations.com/resources/turbulence-models-in-cfd/

[20] OpenCFD (2022). OpenFOAM: User Guide v2112. In *Physical Modeling, Turbulence, Reynolds Averaged Simulation (RAS)*. OpenCFD Ltd, 2022. https://www.openfoam.com/documentation/guides/latest/doc/guide-turbulence-ras.html

[21] OpenCFD (2022). OpenFOAM: User Guide v2112. In *Physical Modeling, Turbulence, Reynolds Averaged Simulation (RAS), k-omega Shear Stress Transport (SST)*. OpenCFD Ltd. https://www.openfoam.com/documentation/guides/latest/doc/guide-turbulence-ras-k-omega-sst.html

[22] Schmitt, F. (2007). “About Boussinesq's turbulent viscosity hypothesis: historical remarks and a direct evaluation of its validity.” *Comptes Rendus Mecanique, 335*(9). Doi: 10.1016/j.crme.2007.08.004

[23] J. Boussinesq (1877). *Essai sur la théorie des eaux courantes, Mémoires présentés par divers savants à l’Académie des Sciences, 23*(1), 1–680.

[24] F.R. Menter, M. Kuntz, and R. Langtry (2003). “Ten years of industrial experience with the SST turbulence model.” In *Proceedings of the fourth international symposium on turbulence, heat and mass transfer,* pages 625–632, Antalya, Turkey. Begell House.

[25] Wasserman, S. (2016). “Choosing the right turbulence model for your CFD simulation.” https://www.engineering.com/story/choosing-the-right-turbulence-model-for-your-cfd-simulation

[26] Winter, A. (2019). *Effects of flow shielding and channeling on tsunami-induced loading of coastal structures* [Doctoral dissertation, University of Washington]. https://www.researchgate.net/publication/338375751_Effects_of_Flow_Shielding_and_Channeling_on_Tsunami-Induced_Loading_of_Coastal_Structures

[27] Newmark, Nathan M. (1959). "A method of computation for structural dynamics." *Journal of the Engineering Mechanics Division, 85*(EM3): 67–94. https://doi.org/10.1061/JMCEA3.0000098

[28] OpenFOAM Development Repository. “Newmark.C.” OpenCFD Ltd. https://develop.openfoam.com/Development/openfoam/-/blob/master/src/rigidBodyDynamics/rigidBodySolvers/Newmark/Newmark.C

[29] Urquhart, M. (2016). “A tutorial of the sixDofRigidBodyMotion library with multiple bodies.” *A course at Chalmers University of Technology Taught by Hakan Nilsson*.

[30] Unofficial OpenFOAM wiki. “Parameter Definitions – dynamicMotionSolverFvMesh.” https://openfoamwiki.net/index.php/Parameter_Definitions_-_dynamicMotionSolverFvMesh#Acceleration_Relaxation_and_Damping

[31] OpenFOAM Development Repository. “rigidBodyMotion.C.” OpenCFD Ltd. https://develop.openfoam.com/Development/openfoam/-/blob/master/src/rigidBodyDynamics/rigidBodyMotion/rigidBodyMotion.C

[32] Hadzić I., Hennig J., Perić M. and Xing-Kaeding Y. (2005). “Computation of flow-induced motion of floating bodies.” *Appl. Math. Model, 29*(12), 1196–1210. https://doi.org/10.1016/j.apm.2005.02.014

[33] Söding, H. (2001). “How to integrate free motions of solids in fluids added mass.” *Nutts Conference*, Hamburg. https://www.researchgate.net/publication/336553077_How_to_Integrate_Free_Motions_of_Solids_in_Fluids

[34] Chow, J. H., and Ng, E. Y. K. (2016). “Strongly coupled partitioned six degree-of-freedom rigid body motion solver with Aitken's dynamic under-relaxation.” *International Journal of Naval Architecture and Ocean Engineering, 8*(4), 320-329. https://doi.org/10.1016/j.ijnaoe.2016.04.001

[35] OpenFOAM v2012. Download. OpenCFD Ltd. https://dl.openfoam.com/source/v2012/

[36] Siemens Industries Digital Software. Simcenter STAR-CCM+ User Guide, version 2021.3. Siemens, 2021.

[37] Schmitt, P., and Elsaesser, B. (2019). “Corrigendum to ’On the Use of OpenFOAM to Model Oscillating Wave Surge Converters’ Ocean Engineering 108 (2015) 98 - 104”. *Ocean Engineering, 171*, 708. https://doi.org/10.1016/j.oceaneng.2019.01.006

[38] Benites-Munoz, D., Huang, L., Anderlini, E., Marin-Lopez, J. R., and Thomas, G. (2020). “Hydrodynamic modeling of an oscillating wave surge converter including power take-off.” *J. Mar. Sci. Eng., 8*(10), 771. https://doi.org/10.3390/jmse8100771

[39] OpenCFD (2022). OpenFOAM: User Guide v2112. In *Physical modelling, Wave modelling.* OpenCFD Ltd, 2022. https://www.openfoam.com/documentation/guides/latest/doc/guide-wave-modelling.html

[40] MIT Abaqus Documents. *Airy wave theory*. https://abaqus-docs.mit.edu/2017/English/SIMACAETHERefMap/simathe-c-airywave.htm

[41] OpenCFD Ltd. OpenFOAM Documentation: User Guide. In *A.4 Standard boundary conditions.* OpenCFD Ltd, 2022. https://www.openfoam.com/documentation/user-guide/a-reference/a.4-standard-boundary-conditions

[42] OpenCFD (2022). OpenFOAM: User Guide v2112. In *Modules, Turbulence models, Wall functions.* OpenCFD Ltd, 2022. https://www.openfoam.com/documentation/guides/latest/api/group__grpWallFunctions.html

[43] OpenCFD Ltd. OpenFOAM Documentation: User Guide. In *Mesh Generation and Conversion.* OpenCFD Ltd, 2022. https://www.openfoam.com/documentation/user-guide/4-mesh-generation-and-conversion/4.3-mesh-generation-with-the-blockmesh-utility

[44] Kitware, Inc. (2022). ParaView. paraview.org.

[45] Jasak, H. (2009). "Dynamic Mesh Handling in OpenFOAM." *47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition.* 5-8 January 2009, Orlando, Florida. https://doi.org/10.2514/6.2009-341

[46] OpenCFD Ltd. OpenFOAM Documentation: User Guide. In *Mesh Generation and Conversion.* OpenCFD Ltd, 2022. https://www.openfoam.com/documentation/user-guide/4-mesh-generation-and-conversion/4.4-mesh-generation-with-the-snappyhexmesh-utility

[47] OpenCFD (2022). OpenFOAM: User Guide v2112. In *Numerics, Overset.* OpenCFD Ltd, 2022. https://www.openfoam.com/documentation/guides/latest/doc/guide-overset.html

[48] Windt, C., Davidson, J., Chandar, D., Ringwood, J., and Faedo, N. (2020). "Evaluation of the overset grid method for control studies of wave energy converters in OpenFOAM numerical wave tanks." *Journal of Ocean Engineering and Marine Energy*. Doi: 10.1007/s40722-019-00156-5

[49] Chandar, D. (2019). "On overset interpolation strategies and conservation on unstructured grids in OpenFOAM." *Computer Physics Communications, 239*, 72-83. https://doi.org/10.1016/j.cpc.2019.01.009

[50] OpenCFD (2022). OpenFOAM: User Guide v2112. In *Classes, cellCellStencils, inverseDistance.* OpenCFD Ltd, 2022.

https://www.openfoam.com/documentation/guides/latest/api/classFoam_1_1cellCellStencils_1_1inverseDistance.html#details

[51] OpenFOAM Development Repository. “inverseDistanceCellCellStencil.C.” OpenCFD Ltd. https://develop.openfoam.com/Development/openfoam/-/blob/master/src/overset/cellCellStencil/inverseDistance/inverseDistanceCellCellStencil.C

[52] OpenCFD (2022). OpenFOAM: User Guide v2112. In *Post-processing, Function objects, Forces*. OpenCFD Ltd, 2022. https://www.openfoam.com/documentation/guides/latest/doc/guide-fos-forces-forces.html

[53] Le Mehaute, B. (1976). *An introduction to hydrodynamics and water waves.* Springer Science+Business Media New York. Doi: 10.1007/978-3-642-85567-2

Appendix A
OpenFOAM Scripts: Background Mesh

A.1 0.org/alpha.water

```
dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    bottomWall
    {
        type            zeroGradient;
    }
    rightWall
    {
        type            zeroGradient;
    }
    leftWall
    {
        type            zeroGradient;
    }

    atmosphere
    {
        type            inletOutlet;
        inletValue      $internalField;
        value           $internalField;
    }
    inlet
    {
            type                waveAlpha;
        value            $internalField;
    }
    outlet
    {
        type         variableHeightFlowRate;
            lowerBound      0;
            upperBound      1;
            value           $internalField;
    }
    flap
    {
            type            zeroGradient;
    }

    defaultFaces
    {
            type            empty;
    }

}
```

A.2 0.org/k

```
dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0.000001;

boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    bottomWall
    {
        type         kqRWallFunction;
        value         $internalField;
    }
    rightWall
    {
        type         kqRWallFunction;
        value          $internalField;
    }
    leftWall
    {
        type          kqRWallFunction;
        value          $internalField;
    }

    atmosphere
    {
        type          inletOutlet;
        inletValue     $internalField;
        value          $internalField;
    }
    inlet
    {
            type                inletOutlet; //fixedValue;
            inletValue          $internalField;
            value               $internalField;  // uniform 0.00375;
    }

    outlet
    {
            type                inletOutlet;
            inletValue          $internalField;
            value               $internalField;
    }
    flap
    {
            type                kqRWallFunction;
            value               $internalField;
    }

}
```

A.3 0.orig/nut

```
dimensions      [0 2 -1 0 0 0 0];
```

```
internalField   uniform 0; //5e-07;

boundaryField
{
   //- Set patchGroups for constraint patches
   #includeEtc "caseDicts/setConstraintTypes"

   bottomWall
   {
      type           nutkWallFunction;
      Cmu            0.09;
      kappa          0.41;
      E            9.8;
      value          $internalField;
   }
   rightWall
   {
      type           nutkWallFunction;
      Cmu            0.09;
      kappa          0.41;
      E            9.8;
      value          $internalField;
   }
   leftWall
   {
      type           nutkWallFunction;
      Cmu            0.09;
      kappa          0.41;
      E            9.8;
      value          $internalField;
   }

   atmosphere
   {
      type           zeroGradient;
   }
   inlet
   {
          type                  calculated;
```

```
        value               $internalField;
    }

    outlet
    {
        type                calculated;
        value               $internalField;
    }
    flap
    {
        type                nutkWallFunction;
        Ks                  uniform 100e-6;
        Cs                  uniform 0.5;
        value               $internalField;
    }

}
```

A.4 0.orig/omega

```
dimensions      [0 0 -1 0 0 0 0];

internalField   uniform 0.001;

boundaryField
{
    //- Set patchGroups for constraint patches
    #includeEtc "caseDicts/setConstraintTypes"

    inlet
    {
      type          inletOutlet;
        inletValue        $internalField;
      value         $internalField;
    }

    outlet
    {
     type          inletOutlet;
```

```
        inletValue      $internalField;
        value          $internalField;
    }

    atmosphere
    {
        type           inletOutlet;
        inletValue      $internalField;
        value          $internalField;
    }

    flap
    {
        type           omegaWallFunction;
        value          $internalField;
    }

    rightWall
    {
            type                omegaWallFunction;
            value               $internalField;
    }

    leftWall
    {
        type           omegaWallFunction;
        value          $internalField;
    }

    bottomWall
    {
        type           omegaWallFunction;
        value          $internalField;
    }

}
```

A.5 0.orig/p_rgh

```
dimensions      [1 -1 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
   //- Set patchGroups for constraint patches
   #includeEtc "caseDicts/setConstraintTypes"

   oversetPatch
   {
          type      overset;
   }

   bottomWall
   {
      type            fixedFluxPressure;
      value           $internalField;
   }
   rightWall
   {
      type            fixedFluxPressure;
      value           $internalField;
   }
   leftWall
   {
      type            fixedFluxPressure;
      value           $internalField;
   }

   atmosphere
   {
      type            totalPressure;
          U                     U;
          phi                   phi;
          rho                   rho;
          psi                   none;
          gamma                 1;
      p0        uniform 0;
```

```
        value           uniform 0;
    }
    inlet
    {
        type            fixedFluxPressure;
        value           $internalField;
    }

    outlet
    {
        type            zeroGradient;
    }
    flap
    {
        //type          zeroGradient;

        type            fixedFluxPressure;
        value           $internalField;
    }

    overset
    {
        patchType       overset;
        type            fixedFluxPressure;
    }

}
```

A.6 0.orig/pointDisplacement

```
dimensions      [0 1 0 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    //- Set patchGroups for constraint patches
    #includeEtc "caseDicts/setConstraintTypes"
```

```
oversetPatch
{
        patchType        overset;
        type             zeroGradient;
}

sides
{
        patchType        overset;
        type             zeroGradient;
}

bottomWall
{
    type           fixedValue;
    value          uniform (0 0 0);
}
rightWall
{
    type           fixedValue;
    value          uniform (0 0 0);
}
leftWall
{
    type           fixedValue;
    value          uniform (0 0 0);
}

atmosphere
{
    type           fixedValue;
    value          uniform (0 0 0);
}
inlet
{
        type             fixedValue;
        value            uniform (0 0 0);
}
```

```
    outlet
    {
            type                fixedValue;
            value               uniform (0 0 0);
    }
    flap
    {
            type                calculated;
    }

}
```

A.7 0.orig/U

```
#include        "$FOAM_CASE/constant/waveProperties";

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    //- Set patchGroups for constraint patches
    #includeEtc "caseDicts/setConstraintTypes"

    inlet
    {
        type           waveVelocity;
            value               $internalField;
    }
    outlet
    {
            type                inletOutlet;
            inletValue          $internalField;
            value               $internalField;
    }

    atmosphere
    {
        type            pressureInletOutletVelocity;
        tangentialVelocity $internalField;
        value           uniform (0 0 0);
    }

    bottomWall
    {
        type            noSlip;
    }
    rightWall
    {
        type            noSlip;
    }
    leftWall
    {
        type            noSlip;
    }
    flap
    {
            type                movingWallVelocity;
            value               uniform (0 0 0);
    }


}
```

A.8 0.orig/zoneID

```
dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    rightWall
    {
        type            zeroGradient;
    }

    leftWall
    {
        type            zeroGradient;
    }

    bottomWall
    {
        type            zeroGradient;
    }

    atmosphere
    {
        type            zeroGradient;
    }

    flap
    {
            type                zeroGradient;
    }

    inlet
    {
            type                zeroGradient;
    }

    outlet
    {
            type                zeroGradient;
    }

}
```

A.9 constant/dynamicMeshDict

```
motionSolverLibs    (sixDoFRigidBodyMotion);

dynamicFvMesh               dynamicOversetFvMesh;
```

```
solver            sixDoFRigidBodyMotion;

sixDoFRigidBodyMotionCoeffs
{
   patches        (flap);
   innerDistance   2;
   outerDistance   2.1;

   centreOfMass    (0 0 0.12);

   // Cuboid mass
   mass           10.77;

   // Cuboid moment of inertia about the centre of mass
   momentOfInertia (1e10 0.1750 1e10); //From Benitese-Munoz Paper
   //momentOfInertia (1e10 0.1161 1e10);  //From Schmitt & Elsaesser Paper

   report         on;
   accelerationRelaxation 0.4;
   accelerationDamping 0.8;

   solver
   {
      type Newmark;
          gamma 0.5;
          beta      0.25;
   }

   constraints
   {
       fixedPoint
       {
          sixDoFRigidBodyMotionConstraint point;
          centreOfRotation (0 0 0);
       }

      fixedLine
      {
        sixDoFRigidBodyMotionConstraint line;
        centreOfRotation (0 0 0);
        direction (0 1 0);
    }

    fixedAxis
    {
        sixDoFRigidBodyMotionConstraint axis;
        axis (0 1 0);
    }
}

restraints
{
      torsionalspring
      {
              sixDoFRigidBodyMotionRestraint          linearAxialAngularSpring;
              axis      (0 1 0);
              referenceOrientation (1 0 0 0 1 0 0 0 1);
              stiffness            5;
              damping                        0;
      }
}

}
```

A.10 constant/g

```
dimensions      [0 1 -2 0 0 0 0];
value           (0 0 -9.81);
```

A.11 constant/hRef

```
dimensions      [0 1 0 0 0 0 0];
value           0;
```

A.12 constant/transportProperties

```
phases (water air);
```

```
water
{
   transportModel  Newtonian;
   nu              1.09e-06;
   rho             998.2;
}

air
{
   transportModel  Newtonian;
   nu              1.48e-05;
   rho             1;
}

sigma           0.07;
```

A.13 constant/turbulenceProperties

```
simulationType  RAS;

RAS
{
   RASModel        kOmegaSST;

   turbulence      on;

   printCoeffs     on;
}
```

A.14 constant/waveProperties

```
inlet
{
   alpha           alpha.water;
   waveModel               StokesI;
   nPaddle                 6;
   waveHeight              0.05;
   waveAngle               0;
   activeAbsorption       yes;
   rampTime               0;
   wavePeriod             2.0625;
   waveLength             4.782550298061209;
}
```

A.15 system/blockMeshDict

```
// Pal Schmitt Channel

convertToMeters 1;

// Define some dimensions
hinge_x 12.5525;
hinge_z 0.476;
L1 4.82;
L2 1.3;
L3 2.4;
L4 3.7;
L5 6.2;
H1 0.691; //deepest depth
H2 0.541; //middle depth
H3 0.335; //shallow depth
H4 0.30; //H4 0.24; //height to atmosphere boundary
W 4.58; //flume width

// Coordinates with origin at center of oswc
WL #calc "$H1-$hinge_z";  //to waterline
x1 #calc "-$hinge_x";
x2 #calc "$x1+$L1";
x3 #calc "$x2+$L2";
x4 #calc "$x3+$L3";
x5 #calc "$x4+$L4";
x6 #calc "$x5+$L5";
y1 #calc "-$W/2.0";  //right wall
y2 #calc "$W/2.0";   //left wall
z1 #calc "-$hinge_z"; //to bottom of flume
z2 #calc "$WL-$H2";  //to step 1
z3 #calc "$WL-$H3";  //to step 2
```

```
z4 #calc "$WL";      //waterline
z5 #calc "$WL+$H4";  //to top of owsc

vertices
(
  //inlet
  ($x1 $y1 $z1) //0
  ($x1 $y1 $z4) //1
  ($x1 $y1 $z5) //2
  ($x1 $y2 $z1) //3
  ($x1 $y2 $z4) //4
  ($x1 $y2 $z5) //5
  //Start step 1
  ($x2 $y1 $z1) //6
  ($x2 $y1 $z4) //7
  ($x2 $y1 $z5) //8
  ($x2 $y2 $z1) //9
  ($x2 $y2 $z4) //10
  ($x2 $y2 $z5) //11
  //End Step 1
  ($x3 $y1 $z2) //12
  ($x3 $y1 $z4) //13
  ($x3 $y1 $z5) //14
  ($x3 $y2 $z2) //15
  ($x3 $y2 $z4) //16
  ($x3 $y2 $z5) //17
  //Start Step 2
  ($x4 $y1 $z2) //18
  ($x4 $y1 $z4) //19
  ($x4 $y1 $z5) //20
  ($x4 $y2 $z2) //21
  ($x4 $y2 $z4) //22
  ($x4 $y2 $z5) //23
  //End Step 2
  ($x5 $y1 $z3) //24
  ($x5 $y1 $z4) //25
  ($x5 $y1 $z5) //26
  ($x5 $y2 $z3) //27
  ($x5 $y2 $z4) //28
  ($x5 $y2 $z5) //29
  //To OSWC
  (0   $y1 $z3) //30
  (0   $y1 $z4) //31
  (0   $y1 $z5) //32
  (0   $y2 $z3) //33
  (0   $y2 $z4) //34
  (0   $y2 $z5) //35
  //outlet
  ($x6 $y1 $z3) //36 //30
  ($x6 $y1 $z4) //37  //31
  ($x6 $y1 $z5) //38  //32
  ($x6 $y2 $z3) //39 //33
  ($x6 $y2 $z4) //40  //34
  ($x6 $y2 $z5) //41  //35
);

blocks
(
   hex (0 6 9 3 1 7 10 4)        (200 200 50) simpleGrading (1 1 0.5)
   hex (1 7 10 4 2 8 11 5)       (200 200 50) simpleGrading (1 1 1)
   hex (6 12 15 9 7 13 16 10)    (60 200 50) simpleGrading (1 1 0.5)
   hex (7 13 16 10 8 14 17 11)   (60 200 50) simpleGrading (1 1 1)
   hex (12 18 21 15 13 19 22 16) (120 200 50) simpleGrading (1 1 0.5)
   hex (13 19 22 16 14 20 23 17) (120 200 50) simpleGrading (1 1 1)
   hex (18 24 27 21 19 25 28 22) (195 200 50) simpleGrading (0.5 1 0.5)
   hex (19 25 28 22 20 26 29 23) (195 200 50) simpleGrading (0.5 1 1)
   hex (24 30 33 27 25 31 34 28) (30 200 50) simpleGrading (1 1 0.5) // last flat sect to
flap (wat)
   hex (25 31 34 28 26 32 35 29) (30 200 50) simpleGrading (1 1 1) //last flat sect to flap
(atm)
   hex (30 36 39 33 31 37 40 34) (300 200 50) simpleGrading (3 1 0.5) //flap to outlet
(wat)
   hex (31 37 40 34 32 38 41 35) (300 200 50) simpleGrading (3 1 1) //flap to outlet (atm)
);

edges
(
);
```

```
boundary
(
    oversetPatch //dummy patch to trigger overset interpolation before any other bcs
    {
            type      overset;
            faces
            ();
     }

    bottomWall
    {
        type wall;
        faces
        (
            //Bottom
           (0 6 9 3)
           (6 12 15 9)
           (12 18 21 15)
           (18 24 27 21)
           (24 30 33 27)
                (30 36 39 33)
            );
    }
    rightWall
    {
            type wall;
            faces
            (
            //right wall
                (0 6 7 1)
                (1 7 8 2)
                (6 12 13 7)
                (7 13 14 8)
                (12 18 19 13)
                (13 19 20 14)
                (18 24 25 19)
                (19 25 26 20)
                (24 30 31 25)
                (25 31 32 26)
                (30 36 37 31)
                (31 37 38 32)
            );
     }
     leftWall
     {
            type wall;
            faces
            (
            //left wall
                (3 4 10 9)
                (4 5 11 10)
                (9 10 16 15)
                (10 11 17 16)
                (15 16 22 21)
                (16 17 23 22)
                (21 22 28 27)
                (22 23 29 28)
                (27 28 34 33)
                (28 29 35 34)
                (33 34 40 39)
                (34 35 41 40)
        );
    }
    atmosphere
    {
        type patch;
        faces
        (
           (2 8 11 5)
                (8 14 17 11)
                (14 20 23 17)
                (20 26 29 23)
                (26 32 35 29)
                (32 38 41 35)
        );
    }
    inlet
```

```
    {
            type patch;
            faces
            (
               (0 1 4 3)
               (1 2 5 4)
            );
    }
    outlet
    {
            type patch;
            faces
            (
               (36 39 40 37)
               (37 40 41 38)
               //(30 33 34 31)
               //(31 34 35 32)
            );
    }
);

mergePatchPairs
(
);
```

A.16 system/controlDict

```
libs                 (overset fvMotionSolvers);

application     overInterDyMFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         60;
```

```
deltaT          0.00001;

writeControl    adjustable;

writeInterval   0.05;

purgeWrite      6;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable yes;

adjustTimeStep  yes;

DebugSwitches
{
 level   2;
 lduMatrix 2;
}

maxCo           10;
maxAlphaCo      5;
maxDeltaT       0.001;

libs (
        waveModels
        overset
        sixDoFRigidBodyMotion
        fvMotionSolvers);

functions
```

```
{

    interfaceHeight1
    {
            type                interfaceHeight;
            libs                (fieldFunctionObjects);

            locations           ((-12.1 0 0.01) (-6 0 0.01) (-4 0 0.01) (4 0 0.01));

            alpha               alpha.water;
            liquid  true;

            writePrecision 8;
            writeToFile                 true;
            useUserTime                 true;

            writeControl        timeStep;//adjustable;
            writeInterval                   1;
            executeControl      timeStep; //adjustable;
            executeInterval                 1;

    }


  probes
  {
     type           probes;
     libs          (sampling);

     // Name of the directory for probe data
     name           probes;

     // Write at same frequency as fields
     writeControl    timeStep;
     writeInterval   1;

     // Fields to be probed
     fields         (p U);
     // Optional: interpolation scheme to use (default is cell)
     interpolationScheme cell;

     probeLocations
     (
        (-12  0 0.1)
            (4  0 0.1)
            (-1   0 0.1)
     );
  }

  alphaVol
  {
     libs            (utilityFunctionObjects);
     type             coded;
     name             alphaVolume;
     writeControl    timeStep;
     writeInterval   10;

     codeWrite
     #{

         const volScalarField& alpha =
            mesh().lookupObject<volScalarField>("alpha.water");

         Info<< "Alpha volume = " << alpha.weightedAverage(mesh().Vsc())
            << endl;
     #};
  }

  forces
  {
         type                 forces;
         libs                 (forces);
         writeControl         timeStep;
         timeInterval         1;
         log                  yes;
         patches              ("flap");
         rho                  rhoInf;
         rhoInf          1;
         CofR            (0 0 0);
         pitchAxis       (0 1 0);
  }

  surfaces
  {
         type            surfaces;
         libs            (sampling);
         writeControl    writeTime;
         surfaceFormat   vtk;
         fields          (p p_rgh U alpha.water);

         surfaces
         {
            flap
            {
                type            patch;
                patches         ("flap");
                interpolate     true;
                surfaceFormat   vtk;
            }
         }
  }

}
```

A.17 system/decompParDict_run

```
numberOfSubdomains  40;

method          scotch;

simpleCoeffs
{
    n           (4 3 1);
    delta        0.001;
}
```

```
hierarchicalCoeffs
{
   n             (2 2 1);
   delta          0.001;
   order          xyz;
}

manualCoeffs
{
   dataFile       "";
}

distributed     no;

roots          ( );
```

A.18 system/fvSchemes

```
ddtSchemes
{
   default        Euler;
}

gradSchemes
{
   default        Gauss linear;
   limitedGrad       cellLimited Gauss linear 1;
}

divSchemes
{
   div(rhoPhi,U)  Gauss linearUpwind grad(U);
   div(phi,alpha)  Gauss vanLeer;
   div(phirb,alpha) Gauss linear;
   div(phi,k)      Gauss linearUpwind limitedGrad;
   div(phi,omega) Gauss linearUpwind limitedGrad;
   div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}
```

```
laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

oversetInterpolation
{
    method              inverseDistance; //leastSquares; //cellVolumeWeight;
}

oversetInterpolationSuppressed
{
    grad(p_rgh)
    surfaceIntegrate(phiHbyA);
}

wallDist
{
    method meshWave;
}
```

A.19 system/fvSolution

```
solvers
{
    "cellDisplacement.*"
    {
        solver              PCG;
        preconditioner  DIC;

        tolerance       1e-06;
        relTol          0;
        maxIter         100;
}

"alpha.water.*"
{
    nAlphaCorr      3;
    nAlphaSubCycles 2;
    cAlpha          1;
        icAlpha         0;

    MULESCorr       yes;
    nLimiterIter    15;
    alphaApplyPrevCorr  yes;

    solver          smoothSolver;
    smoother        symGaussSeidel;
    tolerance       1e-8;
    relTol          0;
        minIter         1;
}

"pcorr.*"
{
    solver          PCG;
        preconditioner  DIC;

    tolerance       1e-08;
    relTol          0;
}

p_rgh
{
    solver          PBiCGStab;
        preconditioner  DILU;
    tolerance       5e-8;
```

```
        relTol          0.01;
    }

    p_rghFinal
    {
           $p_rgh

        relTol          0;
    }

    "(U|k|omega)"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-5;
        relTol          0.01;
        nSweeps         1;
           minIter            1;
    }

    "(U|k|omega)Final"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-8;
        relTol          0;
        nSweeps         1;
           minIter            1;
    }
}

PIMPLE
{
    momentumPredictor   no;
    nOuterCorrectors    2;
    nCorrectors         3;
    nNonOrthogonalCorrectors 1;
    ddtCorr                    yes;
    correctPhi          no;
```

```
   moveMeshOuterCorrectors no;
   turbOnFinalIterOnly no;
   oversetAdjustPhi        no;
}

relaxationFactors
{
   equations
   {
      ".*" 1;
   }
}

cache
{
  grad(U);
}
```

A.20 system/setFieldsDict

```
defaultFieldValues
(
   volScalarFieldValue alpha.water 0
   volScalarFieldValue zoneID 123
);

regions
(
   boxToCell
   {
          box (-15 -5 -10) (10 5 0.225);
      fieldValues
          (
             volScalarFieldValue alpha.water 1
          );
   }

   cellToCell
   {
          set c0;
          fieldValues
          (
             volScalarFieldValue zoneID 0
          );
   }
   cellToCell
   {
          set c1;
          fieldValues
          (
             volScalarFieldValue zoneID 1
          );
   }

);
```

A.21 system/setWavesDict

```
alpha   alpha.water;
```

A.22 system/topoSetDict

```
actions
(
   {
        name    c0;
        type    cellSet;
        action  new;
        source  regionToCell;
        insidePoints ((-4 0.1 0.3));
   }
   {
        name    c1;
        type    cellSet;
        action  new;
        source  cellToCell;
        set     c0;
   }
```

```
    {
        name    c1;
        type    cellSet;
        action  invert;
    }
);
```

Appendix B

OpenFOAM Scripts: Overset Mesh

B.1 system/blockMeshDict

```
scale 1;

patch (sides);

radius      0.255;
radiusNeg  -0.255;
box         0.085;
boxNeg     -0.085;
zMax        0.6;
zMin       -0.6;

nR         30;
nZ         30;


zBox            0.305;
zBoxNeg                 -0.02;
radz            0.36;
radzNeg                 -0.1;
yMax            0.6;
yMin            -0.6;

nY              60;

cylrad          0.1625;
ncylrad         -0.1625;

geometry
{
    cylinder
    {
      type cylinder;
          point1     (0 -1 0.0625);
          point2     (0 1 0.0625);
      radius     $cylrad;

    }
};



vertices
(

    (-0.15  $yMin  0.44)
    (0.15   $yMin  0.44)
    (-0.15  $yMin  0.0)
    (0.15   $yMin  0.0)

    (-0.15  $yMax  0.44)
    (0.15   $yMax  0.44)
    (-0.15  $yMax  0.0)
    (0.15   $yMax  0.0)

    project (-0.15  $yMin -0.1) (cylinder)
    project (0.15   $yMin -0.1) (cylinder)

    project (-0.15  $yMax -0.1) (cylinder)
    project (0.15   $yMax -0.1) (cylinder)




);

blocks
(

    hex (0 1 3 2 4 5 7 6) (50 62 120) simpleGrading ( 1 1 1 )
```

```
   hex (2 3 9 8 6 7 11 10) (50 12 120) simpleGrading (1 ( (40 60 1) (60 40 1.5) ) 1)

);

edges
(

   project  8  9 (cylinder)
   project  10  11 (cylinder)
   project  2  8   (cylinder)
   project  3  9   (cylinder)
   project  6  10  (cylinder)
   project  7  11  (cylinder)
);

boundary
(
   sides //oversetFlap
   {
          type      overset;
          faces
          (
           // End caps of cyl
           (4 0 1 5)
           (4 6 2 0)
           (3 7 5 1)
           (0 2 3 1)

           (2 8 9 3)
           (4 5 7 6)
           (6 7 11 10)
           (8 10 11 9)

           (2 6 10 8)
           (9 11 7 3)


          );
   }
   flap
   {
         type wall;
         faces ();
   }

);

mergePatchPairs
(
);
```

B.2 system/controlDict

```
application     subsetMesh;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         30;

deltaT          0.00001;

writeControl    adjustable;

writeInterval   0.1;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;
```

```
timeFormat      general;

timePrecision   6;

runTimeModifiable yes;

adjustTimeStep  yes;

DebugSwitches
{
 level   2;
}

maxCo           10;
maxAlphaCo      5;
maxDeltaT       0.001;
```

B.3 system/fvSchemes

```
ddtSchemes
{
   default         Euler;
}

gradSchemes
{
   default         Gauss linear;
   limitedGrad       cellLimited Gauss linear 1;
}

divSchemes
{
   div(rhoPhi,U)  Gauss linearUpwind grad(U);
   div(phi,alpha)  Gauss vanLeer;
   div(phirb,alpha) Gauss linear;
   div(phi,k)      Gauss linearUpwind limitedGrad;
   div(phi,omega) Gauss linearUpwind limitedGrad;
   div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}
```

```
laplacianSchemes
{
   default        Gauss linear corrected;
}

interpolationSchemes
{
   default        linear;
}

snGradSchemes
{
   default        corrected;
}


wallDist
{
   method meshWave;
}

fluxRequired
{
   default          no;
   p_rgh;
   pcorr;
   alpha.water;
}
```

B.4 system/fvSolution

```
solvers
{
   "alpha.water.*"
   {
      nAlphaCorr      1;
      nAlphaSubCycles 1;
      cAlpha          1;
```

```
        icAlpha             0;

    MULESCorr       yes;
    nLimiterIter    15;
    alphaApplyPrevCorr  yes;

    solver          smoothSolver;
    smoother          symGaussSeidel;
    tolerance        1e-8;
    relTol          0;
        minIter             1;
}

"pcorr.*"
{
    solver          PCG;

        preconditioner
        {
            preconditioner GAMG;
            tolerance       1e-5;
       relTol          0;
       smoother        DICGaussSeidel;
       nPreSweeps      0;
       nPostSweeps     2;
       nFinestSweeps   2;
       cacheAgglomeration false;
       nCellsInCoarsestLevel 10;
       agglomerator    faceAreaPair;
       mergeLevels     1;
        }

    tolerance       1e-05;
    relTol          0;
        maxIter             100;
}

p_rgh
{
    solver         GAMG;
    tolerance       5e-8;
    relTol         0.01;
    smoother       DIC;
        nPreSweeps      0;
    nPostSweeps     2;
    nFinestSweeps   2;
    cacheAgglomeration true;
    nCellsInCoarsestLevel 10;
    agglomerator    faceAreaPair;
    mergeLevels     1;

}

p_rghFinal
{
        solver         PCG;
    preconditioner
    {
       preconditioner  GAMG;
       tolerance       1e-8;
       relTol          0;
       nVcycles        2;
       smoother        DICGaussSeidel;
       nPreSweeps      2;
       nPostSweeps     2;
       nFinestSweeps   2;
       cacheAgglomeration true;
       nCellsInCoarsestLevel 10;
       agglomerator    faceAreaPair;
       mergeLevels     1;
        }

        tolerance       1e-8;
    relTol         0;
    maxIter        20;

}
```

```
    "(U|k|omega)"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-5;
        relTol          0.01;
        nSweeps         1;
            minIter             1;
    }

    "(U|k|omega)Final"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-8;
        relTol          0;
        nSweeps         1;
            minIter             1;
    }
}

PIMPLE
{
    momentumPredictor   no;
    nOuterCorrectors    3;
    nCorrectors         2;
    nNonOrthogonalCorrectors 0;
    correctPhi          yes;
    moveMeshOuterCorrectors yes;
    turbOnFinalIterOnly no;
}

relaxationFactors
{
    fields
    {
    }
    equations
    {
        ".*" 1;
    }
}
```

B.5 system/topoSetDict

```
actions
(
    {
            name    c0;
            type    cellSet;
            action  new;
            source  boxToCell;
            box     (-0.05 -0.325 0) (0.05 0.325 0.34);
    }

    {
        name    c0;
        type    cellSet;
        action  invert;
    }

);
```