Contents

Chapter 1

Introduction

1.1 Introductory Notes

Marine applications often call for computational fluid mechanics. Within the scope of this master thesis a number of tools have been tested and some new created with the aim to gather a library of applications and utilities that can be used in simulation and analysis of fluid solid interaction type problems.

The link between design and the actual simulation is often a large time factor in Computational Fluid Dynamics (CFD). From a CAD[1] generated geometry, often in .igs[2] format, it is necessary to clean up the geometry to obtain a closed topology. Further, work includes meshing the surface and domain around the object to finally set up the case using some CFD software. In the OpenFOAM-1.5 library, an automatic hexagonal volume cell meshing utility is available. From an .StL[3] file, this utility, given a number of parameters, automatically creates a volume mesh of dimensions specified by the user, around the .StL object.

Further, one of the big challenges in CFD today is the implementation of mesh motion in six degrees of freedom, where translation and rotation is allowed in all three directions. That is where the object is allowed to translate and rotate in all three directions, coupled with the simulation of the flow. The applications are numerous where analysis of the fluid-solid interaction can be useful. The study of automatic response systems and vessel induced waves are two. Here a solver has been implemented that combines the *Volume of Fluid*-method (VoF) with translation and rotation in a six degrees of freedom system.

1.1.1 Approach

The model which is central in this master thesis is based on a version of the simplified Navier-Stokes equations which does not account for turbulence, heat transfer nor chemical reactions.

[1] Computer Aided Design
[2] Common format in CAD
[3] Geometry built from triangular elements

The single phase simulations are performed with a transient solver for laminar, incompressible flow of Newtonian fluids. The VoF-method for two fluid flows is a rewritten version of an existing solver where parts of the mesh is moved in accordance to gravity, buoyancy and viscous forces.

Obviously this is a great simplification and needless to say - in most practical applications, turbulence needs to be accounted for. Nevertheless it is still interesting to understand the structure of the computation which is basically the same and in order to add more complexity, a solid foundation is needed.

1.1.2 Objectives

This master thesis is a collaboration between a commercial enterprise, Epsilon, which supplies me with expertise from the industry and on the other hand the IT-department at Uppsala University where support on more theoretical subjects is given. In the last version, to date, of OpenFOAM an automatic hex mesh generator is implemented. The aim is to test this utility to verify the quality of the mesh and to implement a Volume of Fluid solver coupled with mesh movement in six degrees of freedom.

1.1.3 Description of the work

The first weeks were spent on understanding the OpenFOAM library, the structure of the codes and work with tutorials. Also much time was dedicated to reading papers and books to cover the theory. The first task was to understand how to generate volume hex-meshes around an objects imported from a CAD software. Work was done understanding what parameters were needed to be altered in order to generate a mesh that would give sufficient accuracy without using too many cells. This report sums up the work done on the automatic meshing scheme. Three automatically generated meshes will be analyzed and presented. The latter part of the time interval was used to generate code that would improve the VoF solver and allow for dynamic mesh motion. Three cases where a cubic box is floating on water with movement possible in six degrees of freedom are presented.

1.2 Open Source Field Operation and Manipulation

Open Source Field Operation and Manipulation (OpenFOAM) is a C++ library of modules for building applications and utilities used in engineering. It is licenced under GNU license which means that anyone can download, use and manipulate the code as one likes. It has a number of tools that can be used to assemble solvers and utilities aimed at solving and analyzing continuous mechanics problems. The code was under commercial license and owned by the company NABLA up until 2005 when it was released as open source under the GNU license. While it is possible to build solvers from scratch it is a far less time consuming approach to configure one of the existing solvers to suit the problem one is trying to solve. Mainly the work has been with the *InterFoam* and *interDymFoam* solvers, both VoF interface capturing solvers where the latter allows for mesh

motion. The simulations with the cylinder has been made with *icoFoam*, a transient solver for single-phase laminar flow of Newtonian fluids.

The aim was primarily to understand how to automatically mesh the volume of a domain and to develop a code that could combine a VoF-solver with mesh motion in six degrees of freedom to solve marine type problems.

A large part of the work in this master thesis has been dedicated to understand the code. This master thesis has been running parallel with a course in OpenFOAM at Chalmers. Due to the lack of documentation on how to use the code more time than expected was needed to learn how to navigate it.

1.2.1 Structure

After covering some of the basic theory the numerical model of *icoFoam* and *interFoam* is described in Chapter 2 in some detail. The *snappyHexMesh* utility is considered in Chapter 3 before moving on to the results from the qualitative study of the mesh generated in *snappyHexMesh*, later in that same chapter. Last, Chapter 4 the results from simulations with the 6-DoF-solver [4] are presented along with some comments on further work.

[4] Six degrees of freedom, refering to the motion of a rigid body in three-dimensional space.

Chapter 2

Governing Equations and Numerics

2.1 Governing Equations

2.1.1 Single Phase Flow

The flow considered here is that of incompressible Newtonian fluids and thus can be described by the following Navier-Stokes equations

$$\frac{\partial \boldsymbol{U}}{\partial t} + \boldsymbol{U} \cdot \nabla \boldsymbol{U} = -\nabla p + Re^{-1} \nabla^2 \boldsymbol{U} \tag{2.1}$$

where $\boldsymbol{U} = U(x, y, z)$ is the velocity field, Re is the Reynolds number and p is the pressure. The terms in (2.1) are, from the left: acceleration of the flow, the second term is the convective acceleration, caused by a change of velocity over position, for example, close to a cylinder in a flow field the velocity changes due to the change in the path of the flow . On the right hand side first the pressure gradient is found, then viscosity. Under the assumption that the flow is incompressible the continuity equation becomes

$$\nabla \cdot \boldsymbol{U} = 0 \tag{2.2}$$

where $\boldsymbol{U}$ is the velocity field.

2.1.2 Two Phase Flow

This section outlines the interface capturing two phase metod governed by the Navier Stokes equations. There are different ways of handling the two phases, either they are treated in two equations which are matched at the interface or as here in one single equation using an indicator

function γ.

$$\gamma = \begin{cases} 1 & \text{if in the water} \\ 0 < \gamma < 1 & \text{in the region around the surface} \\ 0 & \text{if in the air} \end{cases}$$

In this single field formulation, material properties and interfacial properties are accounted for in the conservation equations. The continuity equation becomes as in (2.2). The momentum equation becomes

$$\frac{\partial \rho \boldsymbol{U}}{\partial t} + \nabla \cdot (\rho \boldsymbol{U}\boldsymbol{U}) = -\nabla p + \nabla \cdot \boldsymbol{\tau} + \rho \boldsymbol{f} + \int_{S(t)} \sigma \kappa' \boldsymbol{n}' \delta(\boldsymbol{x} - \boldsymbol{x}') dS \tag{2.3}$$

where the last term on the right side is a source term due to the surface tension. In a surface capturing scheme like the VoF-method the shape is not known and this last term is rewritten and approximated as $\sigma\kappa\nabla\gamma$. This needs to be supplemented by the stress tensor

$$\boldsymbol{\tau} = \mu(\nabla \boldsymbol{U} + \nabla \boldsymbol{U}^T) \tag{2.4}$$

In the equations above, t is the time, τ is the stress tensor, μ is the kinematic viscosity, $\boldsymbol{f}$ is the acceleration due to body forces (i.e. in the case gravity is the only force $\boldsymbol{f} = \boldsymbol{g}$). The last part of (2.3) is due to the surface tension and this applies to the whole surface $S(t)$. σ is the surface tension coefficient, κ the curvature and $\boldsymbol{n}$ the normal of the interface.

This mathematical description of the flow of two fluids, both incompressible and with constant viscosity includes surface tension but neglects mass and heat transfer. The idea is to write one set of conservation equations which are discontinuous over the interface between the two fluids with respect to the material properties and to the flow field. For the material properties (i.e density and viscosity) on either side of the interface the Heaviside function is convenient.

$$H = \begin{cases} 1 & \text{on the one side of the interface} \\ 0 & \text{on the other side of the interface} \end{cases}$$

With the Heaviside function, this can be represented as a *single field*, here for the density, ρ

$$\rho(\boldsymbol{x}, t) = \rho_w H(\boldsymbol{x}, t) + \rho_a (1 - H(\boldsymbol{x}, t)) \tag{2.5}$$

where $\boldsymbol{x}$ is the position vector, t is the time, ρ_a is the density for air and ρ_w is the density for water.[1]

It can be noted that the interface is not a sharp surface but rather it is treated as a mixture between the two fluids with respect to the cells near the surface. This can be described [1] by using the *indicator function* (4.2.1) γ which obeys the transport equation in the form,

$$\frac{\partial \gamma}{\partial t} + (\boldsymbol{U} \cdot \nabla)\gamma = 0. \tag{2.6}$$

Then, using this indicator function, the density ρ and the kinematic viscosity μ can be written as

$$\rho = \gamma\rho_w + (1-\gamma)\rho_a \tag{2.7}$$

$$\mu = \gamma\mu_w + (1-\gamma)\mu_a. \tag{2.8}$$

Surface tension and other interfacial phenomena are represented by adding interface terms to the governing equations.

In this scheme, developed by Henry Weller [1], one of the developers of OpenFOAM, the compression of the interface is not achieved by a difference scheme but rather by adding an extra term in the indicator function equation

$$\frac{\partial\gamma}{\partial t} + \nabla\cdot(\boldsymbol{U}\gamma) + \nabla\cdot(\boldsymbol{U}_r\gamma(1-\gamma)) = 0. \tag{2.9}$$

Here the extra term is the third, where $\boldsymbol{U}_r$ is the velocity in the interface region. This term has no significant effect in the domain outside the interface region. It is a safety device to restrict the solution for γ to be bounded between zero and one. It should be pointed out that this description is a rough outline for the mathematics behind the *interFoam* solver on which my*InterDyMFoam* solver is built. There is no up-to-date report on the current implementation but the structure described here is the one always referred to when referring to the mathematics behind *interFoam* (e.g. on the OpenFOAM forum)[2].

2.2 Discretization

Spatial discretization of the domain. The domain is divided into cells and relations between this set of cells is stored.

Governing equations discretization of the set of governing algebraic equations. These are discretized to describe the problem in the discrete positions in time and space.

Temporal discretization of time with steps of dynamic size.

2.3 Spatial Discretization - Meshes in OpenFOAM

As the mesh quality is an important part of the numerical solution, OpenFOAM checks continuously that a number of conditions are satisfied, otherwise the simulation will halt. OpenFOAM works with three-dimensional polyhedral cells but here, since the VoF-method prefers cells to be hexagonal, these will be in focus. Further these are considered as volume cells while the polyhedral cells are used on the boundary. The discretization can be generalized as described by figure 2.1. Here the spatial domain is divided into a number of cells and the temporal domain into time steps Δt. A cell is bounded by a set of faces, some faces of the cells will be faced inward the domain and some are forming the boundaries of the same. Figure 2.2 shows two generalized cells where S is the normal of the face and its magnitude is equal to the area of the face. P is

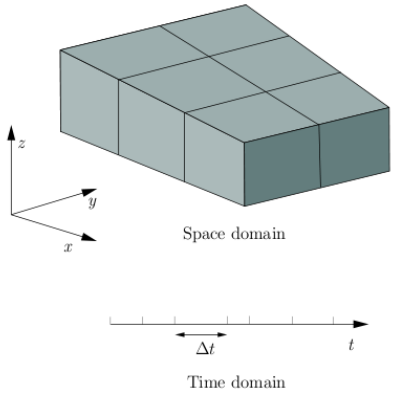

FIGURE 2.1: discretization of the solution domain.

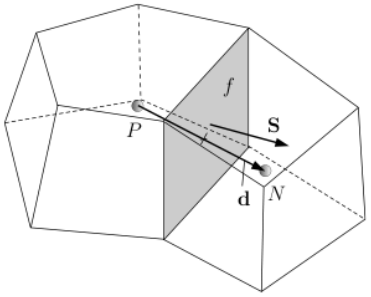

FIGURE 2.2: P is the center of the cell of interest, N the neighboring cell, and d the distance vector.

the cell of interest and N is the neighboring cell, d is the vector from the center of P to the center of N. [1] Further the cells are required to be [3]

Contiguous: the cells completely cover the domain and do not overlap,

Convex: the cells are convex and the cell center must be inside the cell,

Closed: both in geometrical sense (i.e. if face area vectors S are set to point outward their sum should be zero and topological sense (i.e all the edges in a cell i used by two faces),

Orthogonal: i.e. the angle between the face area vector S and the center-to-center vector d between to neighboring cells must always be less than 90 degrees.

In OpenFOAM the mesh are built from a number of objects. These are required to fulfill some specifications and are defined as follows.

Points: two points, specified by a vector, can not have the exact same position.

Faces: faces are made up from a list of points and can be of two different types: *internal*, connecting two cells or *external*, making up the boundary.

Cells: a cell is a list of, arbitrarily ordered, faces. The cells need must be *contiguous* thus cover the whole domain, be *convex*, *closed* and *orthogonal*.

Boundary: the boundary is made up from a list of patches and each boundary must have a specific boundary condition, it must be closed and must contain no internal faces.

Further, a patch is made up from a list of faces and given a name specified by the user. There are a number of base type patches onto which more specific information can be added. Used in this work is the *wall* boundary condition, where the cells next to the wall are considered as part of the patch, used for the *no-slip* condition. The *patch* boundary condition is used for *inlet* and *slip* walls.

2.4 Governing Equation Discretization

After setting up the mesh points, the governing analytical equations need to be discretized and solved at the points defined by the mesh. Consider a generalization of (2.1) as this representation of the transport equation

$$\frac{\partial \rho \Phi}{\partial t} + \nabla \cdot (\rho \boldsymbol{U} \Phi) = \nabla \cdot (\Gamma \nabla \Phi) - \nabla p \tag{2.10}$$

with Φ as a tensorial quantity, Γ diffusivity, ρ the density, $\boldsymbol{U}$ the velocity and p is the pressure.

(2.10) is the integrated over the control volumes V, or cells, defined by the mesh, as

$$\int_V \frac{\partial \rho \Phi}{\partial t} + \int_V \nabla \cdot (\rho \boldsymbol{U} \Phi) = \int_V \nabla \cdot (\Gamma \nabla \Phi) - \int_V \nabla p \tag{2.11}$$

Gauss's theorem (2.13) makes it possible to work with values at the faces rather than with the volumes and the values are interpolated from the cell centers to the face centers. The discretization of the different parts of (2.11) are here described, leaving the first term for the next subsection the second term. The convection term is treated as follows

$$\int_V \nabla \cdot (\rho \boldsymbol{U} \Phi) dV = \int_S (\rho \boldsymbol{U} \Phi) \cdot dS \approx \sum_{face} \boldsymbol{S} \cdot (\rho \boldsymbol{U} \Phi) \tag{2.12}$$

The Gauss theorem has the form

$$\int_V \nabla \cdot A \cdot dV = \int_S A d\boldsymbol{S} \tag{2.13}$$

The diffusion term is discretized in a similar manner

$$\int_V \nabla \cdot (\Gamma \nabla \Phi) dV = \int_S (\Gamma \nabla \Phi) \cdot dS \approx \sum_{face} \Gamma (\boldsymbol{S} \cdot \nabla \Phi) \tag{2.14}$$

Here the divergence term needs to be treated further and this is done by differencing as

$$\boldsymbol{S} \cdot \nabla \Phi = \nabla^{\perp} \Phi \approx \frac{\Phi_N - \Phi_P}{|\boldsymbol{d}|} \tag{2.15}$$

where the $\nabla^{\perp}$ gives the orthogonal contribution, $\boldsymbol{d}$ is the vector between the center of the cell P and the neighboring cell N, see figure 2.2. This works for orthogonal meshes, which is the type used here. [1] The pressure term is discretized as

$$\int_V \nabla p dV \approx \sum_{face} \boldsymbol{S}_f p \tag{2.16}$$

where S_f is the face area vector.

2.4.1 Boundary Conditions

In order to solve the equations boundary conditions have to be specified. The two basic ones, fixed value and zero gradient are the basis of other varieties. Here all the ones used in the following sections are described.

2.4.2 Fixed Value

Also called Dirichlet boundary condition. A variable is assigned a specific value Σ_b at the boundary [4]. For the no-slip boundaries the fixed value is set to zero in all three directions.

2.4.3 Slip

The slip boundary condition is set for the patches where it is desirable that the flow is passing along the walls. In theory this works by letting the velocity be interpolated from the cells just next to it where quantities normal to the wall are set to zero.

2.4.4 Zero Gradient

Or a specific case of the Neumann condition. The gradient normal to the boundary is set to zero. [4]

2.4.5 Symmetry Plane

It is assumed that there is no flow over the symmetry boundary and thus the normal velocity component at the symmetry plane is zero, $\phi_b \cdot n_{sym} = 0$. More, there is no diffusion. Thus, the normal gradients of the flow variables are zero at the symmetry plane, $\nabla\phi_b \cdot n_{sym} = 0$. [5]

2.5 Temporal Discretization

The two solvers that are treated in the next section are both transient. Thus, time needs to be accounted for. Before moving on, the time derivative from (2.11) is treated as

$$\int_V \frac{\partial \rho\Phi}{\partial t} dV \approx \frac{\rho_P^n \Phi_P^n - \rho_P^{n-1}\Phi_P^{n-1}}{\Delta t} V_P \tag{2.17}$$

where $\Phi^n = \Phi(t + \Delta t)$ and $\Phi^{n-1} = \Phi(t)$. Integration of (2.11) in time gives (2.18)

$$\int_t^{t+\Delta t} \int_V \frac{\partial \rho\Phi}{\partial t} dV + \int_V \nabla \cdot (\rho \boldsymbol{U}\Phi) dV dt = \int_t^{t+\Delta t} \int_V \nabla \cdot (\Gamma \nabla\Phi) dV - \int_V \nabla p dV dt \tag{2.18}$$

where ϕ is the flux. The control volumes are assumed to be constant in time and a semi-discretization can be formed by combining (2.12),(2.14), (2.17) and (2.16)

$$\int_t^{t+\Delta t} \frac{\rho_P^n \Phi_P^n - \rho_V^{n-1} \Phi_P^{n-1}}{\Delta t} V_P + \sum_{face} \boldsymbol{S} \cdot (\rho \boldsymbol{U} \Phi) dt = \int_t^{t+\Delta t} \sum_{face} \Gamma(\boldsymbol{S} \cdot \nabla \Phi) - \sum_{face} p \boldsymbol{S} dt \quad (2.19)$$

Two ways are used to treat the integral: an explicit method where care needs to be taken to the Courant number and the implicit Euler method which is always stable. The two methods will be discussed in the following sections. [1]

2.6 Numerical Approach

2.6.1 IcoFoam

IcoFoam is a transient, single phase, laminar flow solver for Newtonian fluids. It uses the explicit discretization of (2.19) which then becomes using $\Phi = U$,

$$V \rho \frac{\boldsymbol{U}^n - \boldsymbol{U}^{n-1}}{\Delta t} + \sum_{face} \boldsymbol{S} \cdot (\rho \boldsymbol{U}^{n-1}) = \sum_{face} \Gamma(\boldsymbol{S} \cdot \nabla \boldsymbol{U}^{n-1}) - \sum_{face} p \boldsymbol{S} \quad (2.20)$$

The user specifies the pressure p and the velocity $\boldsymbol{U}$ as well as the kinematic viscosity along with the schemes to be used by the solver. The procedure is explained below in steps.

The equation (2.20) and the continuity equation (2.2) is solved for U^n and p using the PISO algorithm [6].

2.6.2 myInterDyMFOAM

The present implementation is an addition to the original code which makes it possible to adjust the mesh motion during simulation.

The *myinterDyMFOAM* solver in *OpenFOAM* is using an interface capturing, VoF-method. Here the two phases are considered to make up one single fluid as a mixture of the two. The control volumes or the cells are then assigned a value between 0 and 1. The cells at some distance from the surface will be likely to have either of the extremes. A pressure-correction equation, obtained from the mass and momentum equations is used to correct the velocity components which are first assigned a value by estimation from the momentum equation. It accounts for mesh motion and if needed, turbulence by specifying a turbulence model. [7]

myinterDyMFoam solver can be described by the following steps

- The forces are calculated and the mesh moved, see 4.2.1.
- The mesh is updated.

- The gravity volume and the surface fields are calculated.
- The pressure and fluxes are corrected.
- The Courant number is updated for the new mesh.
- The γ-sub cycle where γ is explicitly solved for using the MULES limiter scheme. The flows are first calculated and then MULES solves it explicitly
- The $\boldsymbol{U}$ equation is set as in *icoFoam* but with the addition of the Laplacian of the velocity and the laminar kinematic viscosity.
- The PISO-loop for pressure p_d, where $p_d = p - g * z * \rho$ to make the boundary conditions simpler to implement in cases where gravity is used. Apart from this the process is the same as the implementation in *icoFoam*.
- Reconstruct p once more from gravity, from the equation above.

What also has to be added is that for the moving object a special boundary condition has to be prescribed to the moving wall of the object to make sure that there is no flow through it. Stated differently, wall normal component (to the wall) of the velocity and the mesh motion flux need to cancel out.

Chapter 3

Automatic Mesh Generation

3.1 Flow around a Cylinder with Vortex Shedding

The study of flow around cylinders has been, experimentally at least, investigated in the last 100 years. It offers almost all of the dynamic phenomena usually of interest such as turbulent and laminar flow, boundary layer separation, unsteady lift and drag and vortices. Essentially these all depend on the Reynolds number(3.1). The Strouhal number, used here to study the flow and compare with previous results, can be calculated from the flow, see equation 3.2. [8]

$$Re = \frac{|U_\infty|L}{\mu} \tag{3.1}$$

$$St = \frac{fL}{U_\infty} \tag{3.2}$$

here U_∞ is the *mean velocity* of the flow, f is the shedding frequency of vortices, L is the *characteristic length*, here the diameter of the cylinder and μ the *kinematic viscosity* [9].

At low Reynolds numbers (< 10) the flow is determined, largely, by viscous effects. It is then said to be stationary. With increasing Reynolds number the cylinder starts shedding unsteady wakes. This phenomenon, in the region downstream of the cylinder is known as the Bernard - Von Karman vortex street [10]. The flow past the cylinder becomes unsteady and shedding wakes at a Reynolds number of around 50. The flow is behaving as mostly two-dimensional up to a Reynolds number of around 180. [9]

The difference of simulating two-dimensional and three-dimensional flow in problematized in [8], where the three-dimensional simulations well agree with the experimental data but shows more dynamic effects than the two-dimensional flow at the same Reynolds number. The drag coefficient C_d can be extracted from the following equation

$$F_d = \frac{1}{2}\rho U_\infty^2 C_d A \tag{3.3}$$

where F_d is the drag force, U_∞ is the mean velocity and A is the reference area. Numerically this is done by calculating the forces and using the values specified by the user as input data.

3.2 SnappyHexMesh

A new feature for the version 1.5 of OpenFOAM is a automatic hex mesh utility, *snappyHexMesh*, which given a background mesh generated in *blockMesh*, and an StL file creates a volume mesh around the StL object in 3D. One of the main goals with the master thesis is to implement a utility that can take a file generated in a CAD software and generate a mesh in a domain around it. After some time it became clear that the requirement that the geometry is closed, in other words that the work done in the CAD file is accurate enough, was too hard to automate. Some preprocessing in a mesh program, ANSA for example, has to be done and the file has to be exported as *.StL*.

The *snappyHexMesh*-utility requires the following:

- An StL file where the object is built by triangles, represented by the three point position vectors and a surface normal vector, either in binary or in ASCII.
- A background mesh. This limits the computational domain and the quality of the initial mesh is a measure of the final quality of the mesh after running *snappyHexMesh*.
- A dictionary that contains details on how to mesh the domain with the StL object.

Important to note is that if the StL object is detailed and a high level of refinement is required on the surface, then the background mesh needs to be finer or the refinement box larger. This of course requires more cells and the amount of cells grow rapidly in three dimensions. In fact, if the background mesh is too coarse then cell splitting effects like in figure 3.1 can be seen. These are a result of the cell splitting scheme that allows the cells to grow to meet the background mesh.

3.3 Results

When the flow past a cylinder reaches a critical state, vortex shedding occurs due to the instability of the wake downstream of the cylinder [11]. This type of flow, simple as it is to set up, has been investigated both experimentally and numerically for decades. An extensive database of information on the subject and numerous articles have been written describing the problem.

The aim here is to establish whether the mesh, generated as described in the previous section, is of good quality. A number of simulations are performed on this particular problem with differently refined meshes in order to establish the mesh quality as good or bad. The solver is a transient solver for laminar single-phase flow of Newtonian fluids.

Here the correlation between the Reynolds number and the drag coefficient C_d will be compared with exiting data and later discussed in detail. Three simulations with different mesh quality have been performed at $Re = 100$ and even though the visual result is convincing, C_d is not as expected for the mesh with the lower number of cells.

3.3.1 Computational Domain

The intention of this investigation is to examine the quality of the mesh generated by the *snappyHexMesh*-utility by comparing to existing data in [9], [10],[8] and [11]. The domain is bounded by the points in $(-22, -6, 0)$ and $(12, 6, 3)$, at $(0, 0, 0)$ a cylinder with the diameter 2 m is centered and it extends through the domain in the z-direction. An additional two, more refined meshes are also generated in order to compare with the first one and make sure that the quality of the computation behaves as expected with a more refined mesh. One of the refined meshes contains a refinement box around the cylinder extending downstream. The other also has a second refinement box around the cylinder, further refining the mesh close to the cylinder. Table 3.3.1 gives information on the details of the meshes, here the first column refers to the cell length at the domain boundary.

Cell Length	Nb. Hex cells	Nb. polyhedral cells	mesh
0.2m	332520	6780	A
0.2m	1083702	26225	B
0.4m	1139072	27656	C

The hexagonal cells are prefered for VoF-simulations and the polyhedral cells are needed for the transition between the cylinder and the initial domain. The computational domain, close to the cylinder for mesh A is depicted in figure 3.1. The fine cells close to the cylinder quickly expands by means of splitting the larger cells surrounding the cylinder. It does not give a significant deterioration of the quality in this kind of simulation when using *icoFoam*.

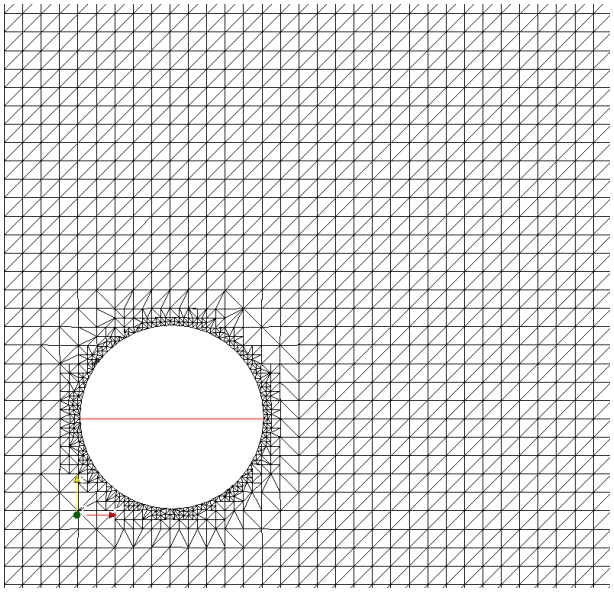

FIGURE 3.1: Mesh A: close to the cylinder, the cells are split up when the background mesh meets the finer mesh on the cylinder

3.3.2 Boundary Condition

In OpenFOAM the boundary conditions for the variables p and U, required by the *icoFoam* solver, are specified in different files, and further it is set separately for the different patches. For p, *zeroGradient*, explained in the previous section, has been used at all patches including the cylinder. U is set to a fixed value at the inlet. The walls are set to slip condition and the cylinder to no-slip [9] by specifying the fixed value zero at the patch. At the outlet the boundary condition for U is set to *zeroGradient*, as recommended in the OpenFOAM user's guide [3].

3.3.3 Mesh A

Initial velocity and velocity at the inlet is set to be 0.5 m/s (in the negative x-direction) giving a Reynolds number of 20, see 3.1 when using the diameter 2m as characteristic length. μ, the kinematic viscosity, is set to 0.010 m^2/s. The mesh is shown in figure 3.1 and as can be seen the mesh is refined just close to the cylinder, situated at the origin.

The simulation ran for 150 seconds on an intel Xenon 2gb memory, single processor, which consumed approximately 64 hours to simulate the flow. In order to compare with existing data the velocity profile is plotted for a number of positions in the flow field. In addition, streamlines for the flow shows that the mesh is accurate enough to give results that are comparable. The streamlines are produced by integrating for the stream function Φ over the y-axis. Consider

$$u = \frac{\partial \Phi}{\partial y}, \qquad v = -\frac{\partial \Phi}{\partial x} \tag{3.4}$$

where, u is the x-component of the velocity and v is the y-component. The boundary conditions are, $\Phi(x, -6) = 0, \Phi(x, 6) = U * 6, \forall x \in [-22, 12]$. With the assumption $\nabla \cdot \boldsymbol{U} = 0$ the stream function can be calculated from

$$\Phi(x, y) = \int_{-6}^{y} u(x, s)ds \tag{3.5}$$

which becomes, as the mesh is a discrete domain

$$\Phi(i, j) = h \sum_{k=0}^{j} u(i, k), \qquad \phi(i, 0) = 0, \forall i = \{1, \dots M\}, j \leqq N, \tag{3.6}$$

where h is the length of the cells (0.2m, at the boundary of the box) and N the total number of grid points on the Y-axis and M the total number on the X-axis. [10] These results can be seen to compare well with the results in [10], [9] and [12] for the Strouhal number and visual behaviour but the drag coefficient is off the charts and this is due to the numerical method used to compute it which requires an extremely fine mesh. The fact that the bubbles behind the cylinder goes from being symmetric at time 35 seconds and before, Figure 3.2 and becomes more asymmetric along the x-axis, Figure 3.3 and that the bubble is shrinking and the flow become more oscillatory, Figure 3.4 as the time increases.

Figure 3.5 shows the velocity profile along a line passing through the center of the cylinder. Notice the sharp gradient of the top curve at $y = 1$ and $y = 11$. This is due to the boundary

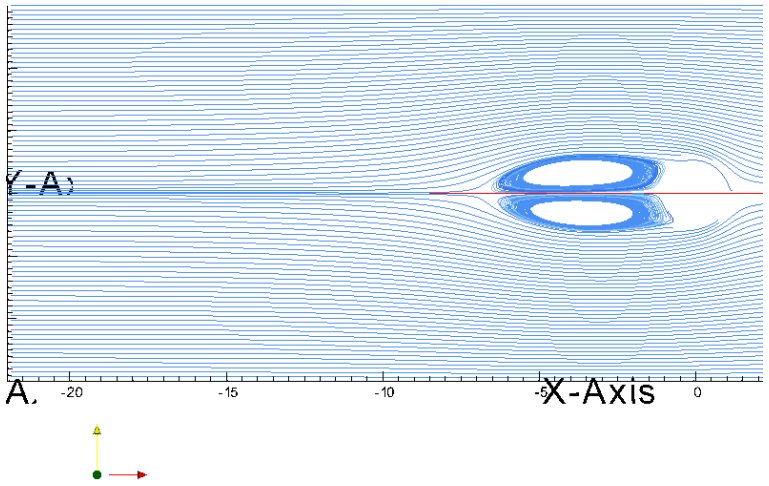

FIGURE 3.2: Mesh A: streamlines for the flow in after 35 s, showing the two bubbles quite symmetric

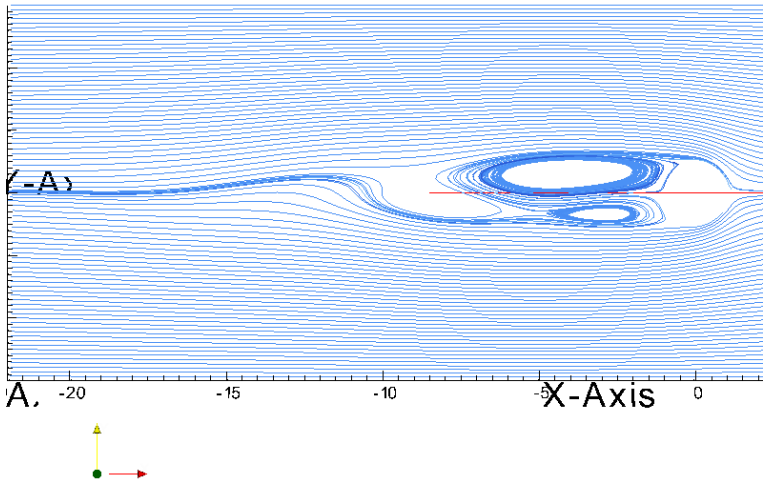

FIGURE 3.3: Mesh A: after 79 s, the flow just starts to shed vortices.

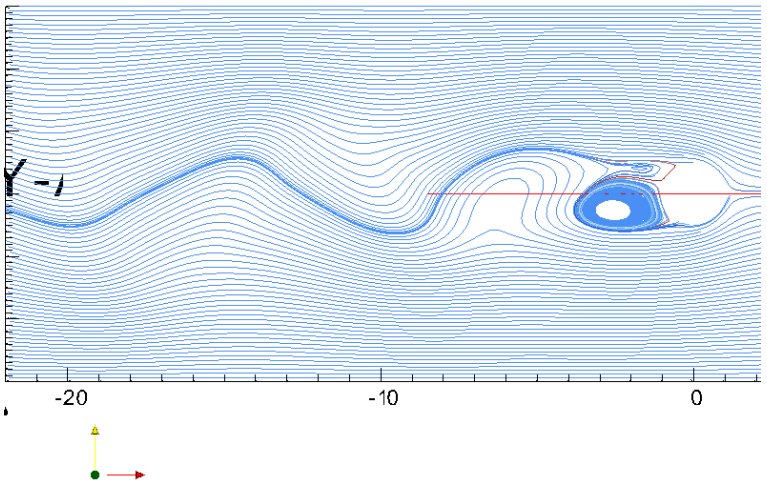

FIGURE 3.4: Mesh A: after 150 s the street of vortices can be seen downstream.

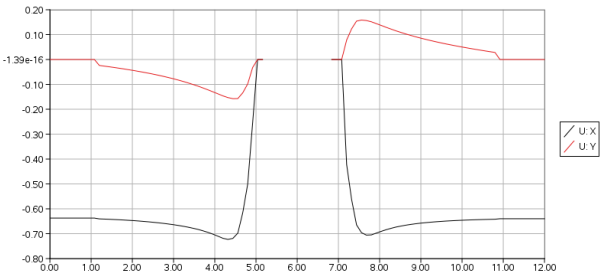

FIGURE 3.5: Mesh A: velocity at a line through the domain $x = 0m$, and $z = 3m$, top curve is v, bottom curve is u at 150 seconds.

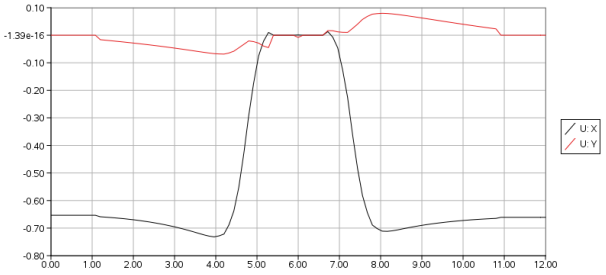

FIGURE 3.6: Mesh A: velocity at a line through the domain $x = -1m$, and $z = 3m$, top curve is v, bottom curve is u at 150 seconds. Just behind the cylinder the velocities are zero and are changing fast close to it.

condition but should not interfere with the results at the cylinder. The velocities are here presented for different positions along a plane. In figure 3.5 the lower curve, u changes sign close to the cylinder indicating that the field is behaving as expected [10]. Figures 3.6 to 3.10 show the velocity profile for a number of positions along the x-axis. The flow becomes more homogenous while moving further away from the cylinder. At $x = -22$, see figure 3.10 the velocity profile indicates that a larger domain could have been used to capture more information. From rewriting some utilities, it was possible to calculate the mean drag coefficient C_d. The output from the utility used to calculate the mean drag coefficient is 3.47, 3.43 and 3.23 for Mesh A, B and C respectivly which are very high values. It is found that the value of C_d is not very convincing. It is about two and a half times the value given in [11], but as is stated in that article, it is

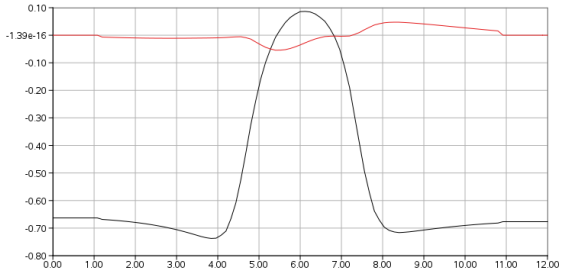

FIGURE 3.7: Mesh A: velocity at a line through the domain $x = -2m$, and $z = 3m$, top curve is v, bottom curve is u at 150 seconds.

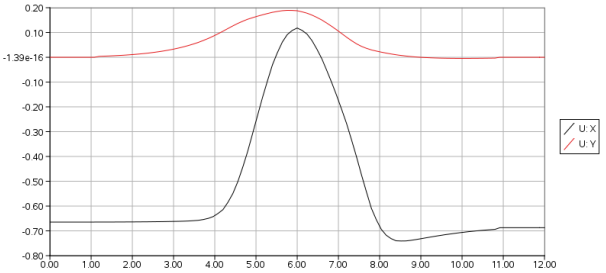

FIGURE 3.8: Mesh A: velocity at a line trough the domain $x = -4m$, and $z = 3m$, top curve is v, bottom curve is u at 150 seconds.

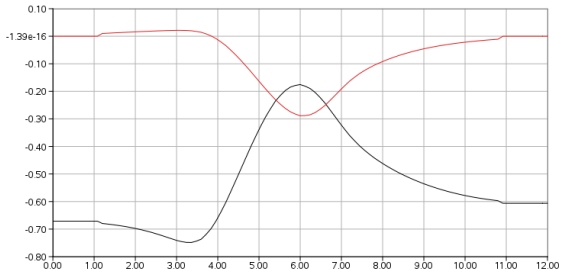

FIGURE 3.9: Mesh A: velocity at a line through the domain $x = -9m$, and $z = 3m$, top curve is v, bottom curve is v at 150 seconds. Here the u velocity is positive relative the direction of the stream

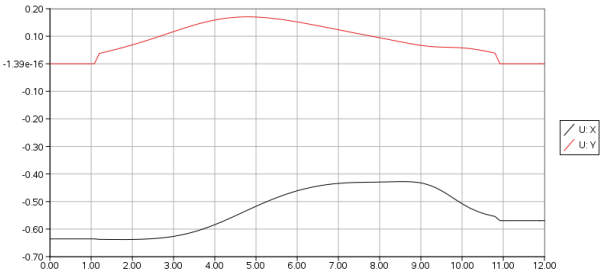

FIGURE 3.10: Mesh A: velocity at a line through the domain $x = -21.5m$, and $z = 3m$, top curve is v, bottom curve is u at 150 seconds. the velocity field is more stable this far down stream.

very difficult measure this quantity and it is stated on the OpenFOAM forum [2] in a number of threads that the *drag*, which is the utility rewritten, only gives good results for meshes with extremely good quality (i.e millions of cells).

The Strouhal number is extracted from the simulation by measuring in the a slice of the flow field. Studying Figure 3.11 it shows that between the darker field closer to the cylinder and the first vortex which is the next dark field it is half a period. One period extends over 10 m. Considering that the flow speed is 0.5 m/s, the frequency is $\frac{1}{20}$. (3.2) gives then a value for St of about 0.2 which is higher than the value expected for a successful simulation, which is around 0.12 [13] . Other methods, including looking at the pressure variation would also be possible.

3.3.4 Mesh B

One refinement box was defined in the domain as can be seen in Figure 3.14. One can see very fine cells growing out from the cylinder. The refinement box makes the mesh of better quality by means of using more and smaller cells. Here $C_d = 3.43$ and it is not lower than the value in mesh A and considerably higher than the 1.2 that is expected. What is interesting to notice is that the flow start shedding vortices after 144 seconds. As can be seen in Figure 3.15 it is just

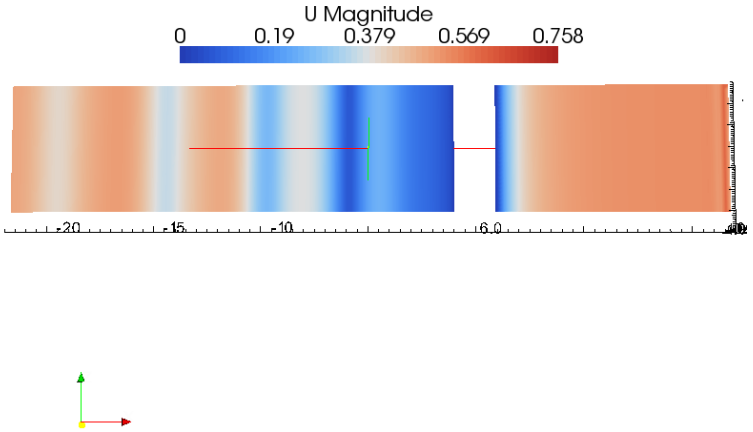

FIGURE 3.11: Mesh A: a slice through the domain cutting the cylinder in half and along the long side of the domain from inlet to outlet showing the velocity field. It is possible to read the distance between the vortices, as very close to 5 meter between the darker lines which gives a period over 10 meters.

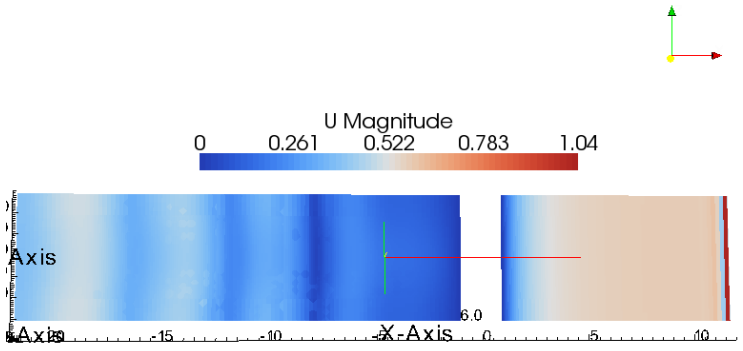

FIGURE 3.12: Mesh B: a slice through the domain cutting the cylinder in half and along the long side of the domain from inlet to outlet showing the velocity field. It is possible to read the distance between the vortices, as very close to 5 meter between the darker lines, giving a period over 10 m.

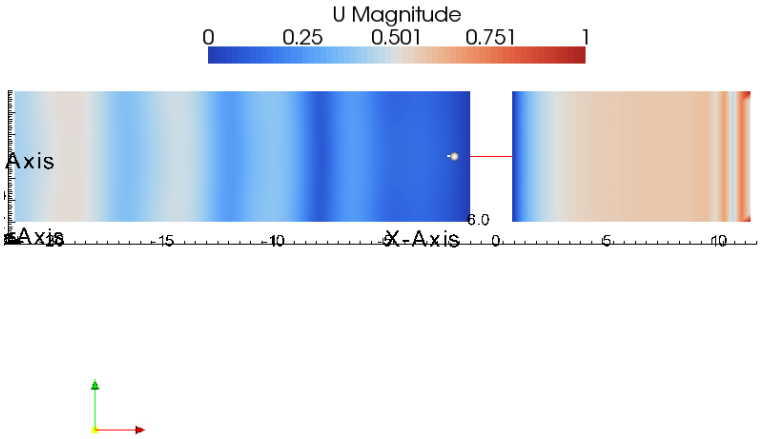

FIGURE 3.13: Mesh C: a slice through the domain cutting the cylinder in half and along the long side of the domain from inlet to outlet showing the velocity field. It is possible to read the distance between the vortices, as very close to 5 meter between the darker lines, giving a period over 10 m.

starting to shed. The Strouhal number of about 0.2 for the flow in mesh B is about the same as in mesh A and still a bit high. The vertical cut through the domain, showing the darker fields as vortices show that the distance is very similar to that in figure 3.12. The simulation took about 5 days before showing these kind of results with the same setup of the system as in mesh A.

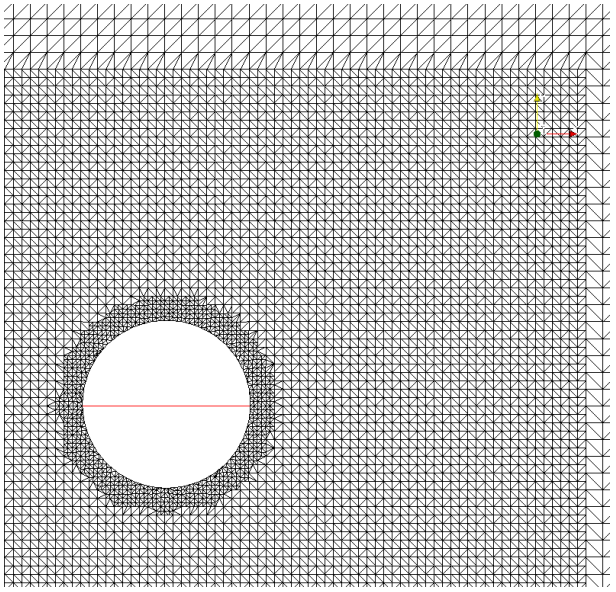

FIGURE 3.14: Mesh B: the mesh close to the cylinder in. The mesh shows that the splitting of cells is made with smaller cells the in mesh A.

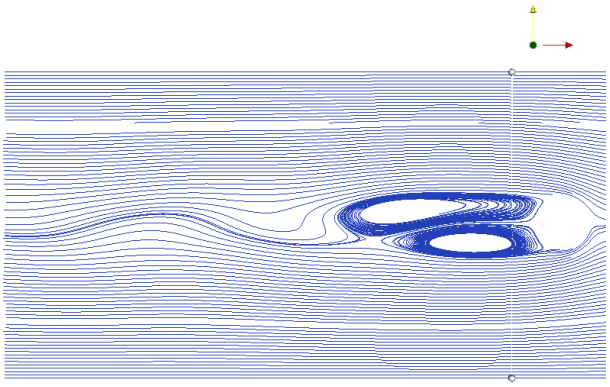

FIGURE 3.15: Mesh B: streamlines for the flow, time 144, this when the flow starts shedding votices.

3.3.5 Mesh C

The mesh in this case has been constructed somewhat differently. The cell size at the domain boundary is four times larger than in the previous cases but instead two refinement boxes have been constructed, one similar to Figure 3.14 and one more as a rectangular cylinder around the cylinder patch. The mesh is depicted in Figure 3.16, as can be seen when comparing with the mesh in Figure 3.14 the cells split up faster and this is due to the fact that the background mesh has cells of four times the volume. The background mesh can be seen as an indicator of the final quality of the mesh. Also here the drag coefficient is about 2.5 times higher than what is expected and found in earlier simulations and experiments in [10], [9], [8] and [11]. Also here, Figure 3.13 shows that the distance between the votices is as before and the Strouhal number is also here the same.

3.4 Conclusions

The Table 3.4 shows the result from earlier experiments compared to the present result.

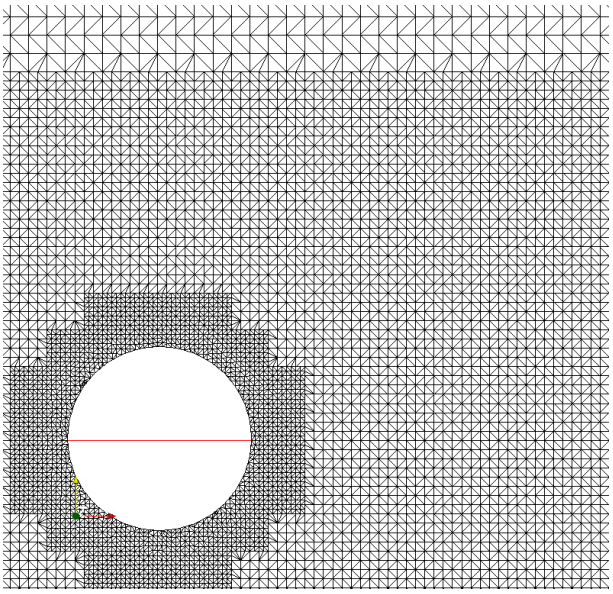

FIGURE 3.16: Mesh C: close to the cylinder.

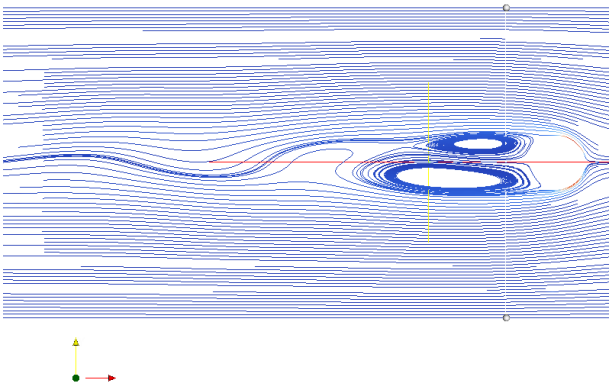

FIGURE 3.17: Mesh C: Streamlines for the time 112 seconds when the flow starts shedding vortices

Source	C_d	St
[9]	1.56	0.168
[8]	1.45	0.162
[11]	1.40	0.2
[10]	1.675	0.188
Mesh A	3.47	0.2
Mesh B	3.43	0.2
Mesh C	3.23	0.2

As it seems, the flow is behaving as expected for all three meshes with respect to the Reynolds number and the visual effect. The Strouhal number and the drag coefficient are higher compared to the numbers in Table 3.4 when comparing to existing data. The streamlines show that over time the behaviour is almost identical to what is found in [10] and concerning the fact that the simulation is performed in a three dimensional domain rather than in a two dimensional as is more common seems to pose no problem and as long as staying away from the boundaries the flow is not affected by the boundaries for a Reynolds number of 100 [8].

As could be seen in the plots, Figure 3.5 - Figure 3.10, the flow in the y-direction is close to zero when getting within one meter from the walls. This is something that need further investigation and analysis of the numerical implementation of the slip boundary condition as implemented in OpenFOAM. As can be seen the gradient is very shap from something that looks undisturbed to a value very close to zero and it is assumed that the flow close to the cylinder is not affected.

The implementation of the utility calculating the drag coeffcient does not seem to work and more time has to be spent on finding the problem. It has been stated that the original version only gives satisfactory results for cell sizes of extremely small scale and that would here mean that some 5 million cells are needed to obtain a good result.

It is interesting to notice the fact that the meshes need a different amount of time to reach the stage where they start shedding vortices. It seems that this is due to how well the boundary layer on the cylinder is resolved. Considering that the more cells that can be fit on the surface the more circular the cylinder will be. The bad result is because the resolution normal to the cylinder is not fine enough.

Chapter 4

Fluid Solid Interaction

In many engineering problems the study the fluid-solid interaction can give useful information on the behavior of a system. This report is mainly focusing on marine applications where a ship will disturb the water surface. The ship will, given a velocity cause the surface to break and spray close to the hull. Adding mesh motion to this not only requires more computer power but also adds more complexity to the surface reconstructions. Resolving these issues give way to studies of ship induced waves and more.

This chapter covers the implementation of a six-DoF[1] solver to couple with a VoF[2] solver. Here a floating box is considered as a simplification of more complex structures. The focus of this first study of a Volume of Fluid solver coupled with mesh motion in six degrees of freedom is to present three simple basic cases.

Case A : the water surface is completely undisturbed and the box floats submerged to 50%. Here the water surface and the horizontal plane cutting through the center of gravity are aligned.

Case B : same set up as above with the addition of a wave that is used to set the box moving.

Case C : same set up as in case B with a more resolved mesh.

The cases are used to describe how it is possible to couple a VoF-solver with a moving mesh in OpenFOAM.

4.1 OpenFOAM

OpenFOAM is, as mentioned in the introduction, a C++ library. It includes a large number of applications and utilities to use out-of-the-box. Applications here refer to a code that is used to solve problems in continuum mechanics. Utilities, are used for pre and post processing. The

[1] Six Degrees of Freedom
[2] Volume of Fluid method

purpose of the library is that it should be used as a base to build more complex and more specific applications and utilities. [3]

The applications and utilities always works in three dimensions. For cases in two dimensions this is solved by only having one cell thickness and not solve equations in the third direction. As the library is designed for continuum mechanics it is convenient to work with tensors to describe physical entities, often with the addition of a dimensional specification to make sure the dimensions (7 of them: kg, m, s, K, kgmol,ampere, candela) work out as desired. [4]

Here an existing solver has been modified to suit marine applications. Additional code has been written to allow for extraction for viscous and buoyancy forces and transform these forces to displacement which has been assigned the mesh. Though the solver does not involve any damping system it represents the structure of a 6-DoF coupled VoF-solver and extending this basic structure to account for turbulence forces and damping is possible within the scope of OpenFOAM.

4.2 Theoretical Introduction

4.2.1 Volume of Fluid Method

The Volume of Fluid method (VoF) is a two-phase surface compression method that solves the Navier-Stokes equations. In the VoF-method the two phases are considered as a single phase with a volume fraction between 0 and 1, in Section 2.1.2 in (4.2.1).

$$\gamma = \begin{cases} 1 & \text{if in the water} \\ 0 < \gamma < 1 & \text{in the region around the surface} \\ 0 & \text{if in the air} \end{cases}$$

The interface is not a sharp surface but rather a region where further refinements are performed. The velocity field is estimated from the previous values and a PISO-algorithm is used to solve an extra transport equation for the volume fraction after which the velocity field is updated.[7]

4.2.2 Under-relaxation

Under-relaxation is an efficient way to stabilize numerical iterative schemes by controlling to what extent the solver uses the values previously calculated to determine the new ones. Let α be the under-relaxation factor, then

$$\phi := \alpha\phi_{new} + (1-\alpha)\phi_{old}, \qquad \alpha \in [0,1] \tag{4.1}$$

where ϕ_{old} is the old value and ϕ_{new} is the new value of a parameter in the solver. [14]

4.2.3 Degrees of Freedom, Forces and angular momentums

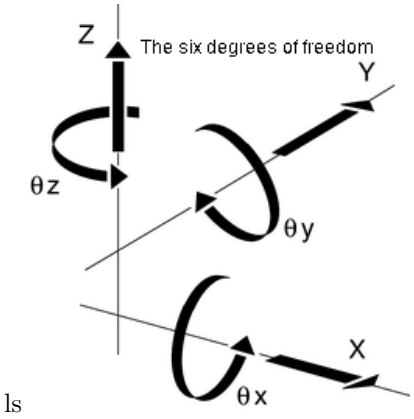

FIGURE 4.1: Axes describing the 6 degrees of freedom

The term six degrees of freedom (6-DoF) refers to how the movement of a body is limited. Having all six degrees of freedom means that the body can translate and rotate along three axes in a three dimensional system. Figure 4.1 shows the possible translations and rotations in three dimensional space. Solving the Navier-Stokes equations for flow it results in forces and these can be calculated by integrating over the body, as in equations (4.2) and (4.3)

$$\boldsymbol{F} = \int_S \boldsymbol{F}_{ext} + \boldsymbol{F}_{flow} dS \tag{4.2}$$

$$\boldsymbol{M} = \int_S \boldsymbol{M}_{ext} + \boldsymbol{M}_{flow} dS \tag{4.3}$$

where $\boldsymbol{F}$ is the total force, the buoyancy force or pressure force and the viscous forces, F_{flow} and $\boldsymbol{M}$ is the total moment around the mass center due to pressure and viscous forces, M_{flow}. $\boldsymbol{F}_{ext}$ is here only the gravity since no other external forces are considered [15].

4.2.4 Rotation and Translation

From the forces extracted as above, the velocity and the displacement for one time step are calculated. It is here necessary to use the accumulated distances because each time step has a mesh updating function that resets the mesh. The calculations of flux and other quantities are done according to the moved mesh. Two objects, quaternion and septernion can store a translation or rotation and when called return a tensor for rotation or a vector for translation. An object of the class septernion consists of a vector and a quaternion. The theoretical focus here will be on the quaternion which is the mathematical object handling the rotation.

A quaternion a vector in four-dimensional vector space, written as a linear combination of the basis elements $\{1, i, j, k\}$ as $a1 + bi + cj + dk$, where a, b,c and d are real numbers and $i^2 = j^2 = k^2 = ijk = -1$ as found by the Irish mathematician Hamilton. Here a is called the scalar part and $bi + cj + dk$ is called the vector part. The vector part of the quaternion can be identified to be the same as an element of R^3 vector space.

In OpenFOAM an object of the *quaternion* class consists of a vector and a scalar - the angle of rotation. A *quaternion* can be constructed in different ways, in this work is has been done by specifying the three Euler angles. This means explicitly that one creates three *quaternions*, one for each angle in the three directions and multiply them together as $q_x * q_y * q_z$. The conversion between the Euler angles and the *quaternion* is described mathematically as follows

$$\begin{aligned} w = c_1c_2c_3 - s_1s_2s_3 \quad v_x = s_1s_2c_3 + c_1c_2s_3 \\ v_y = s_1c_2c_3 + c_1s_2s_3 \quad v_z = c_1s_2c_3 - s_1c_2s_3 \\ c_1 = \cos(rot_y) \quad c_2 = \cos(rot_z) \quad c_3 = \cos(rot_x) \\ s_1 = \sin(rot_y) \quad s_2 = \sin(rot_z) \quad s_3 = \sin(rot_x) \end{aligned}$$

where the w is the resulting angle for the *quaternion* and v_i the vector and rot_i is the rotation around that axis, $i = x, y, z$. The conversion from a *quaternion* back to a rotation vector in R^3 is described below.

$$\begin{aligned} w_2 = \sqrt{w} \quad x_2 = \sqrt{v}_x \\ y_2 = \sqrt{v}_y \quad z_2 = \sqrt{v}_z \\ t_{xy} = 2 * v_x * v_y \quad t_{wz} = 2 * w * v_z \\ t_{xz} = 2 * v_x * v_z \quad t_{wy} = 2 * w * v_y \\ t_{yz} = 2 * v_y * v_z \quad t_{wx} = 2 * w * v_x \end{aligned}$$

$$\begin{pmatrix} w_2 + x_2 - y_2 - z_2 & t_{xy} - t_{wz} & t_{xz} + t_{wy} \\ t_{xy} + t_{wz} & w_2 - x_2 + y_2 - z_2 & t_{yz} - t_{wx} \\ t_{xz} - t_{wy} & t_{yz} + t_{wx} & w_2 - x_2 - y_2 + z_2 \end{pmatrix}$$

These two mathematical maps, first from R^3 to the four dimensional vector space of quaternions and then in the opposite direction describe how the implementation is made in OpenFOAM. This is used in practice once the displacement has been calculated from the forces and the rotation is applied to the patch. It is possible to explicitly write out the rotation matrix but usning these built in functions makes the code easier to read and understand.

4.3 Dynamic Mesh

In many of applications it is interesting to examine the interaction between a solid and one or more fluids. To accomplish this it is necessary to move parts of the mesh and for that purpose there are a number of methods, see [16]. The one here considered is *mesh deformation*, where the cells in the mesh are deformed (stretched or compressed) due to the motion of a part of the mesh.

4.3.1 Mesh Deformation

Mesh deformation can mathematically be considered as a map between the domain D, which represents the configuration at a certain time t with a boundary B, and D' Δt time later. It is only interesting to consider a very small subset of all the possible maps that fulfill this requirement, namely the ones that will result in domain D' where the meshes fulfil the requirements given in Section 3.2 [16]. Consider Figure 4.2 which shows the mesh in the xy-plane at the initial set up. Already here the mesh shows some signs of low quality. Compare this to the mesh in Figure 4.3 which shows that many cells have been skewed severely.

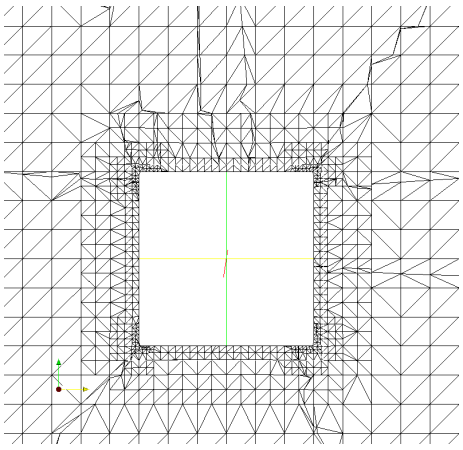

FIGURE 4.2: the xy-plane at time 0, the mesh is showing some signs of being of low quality

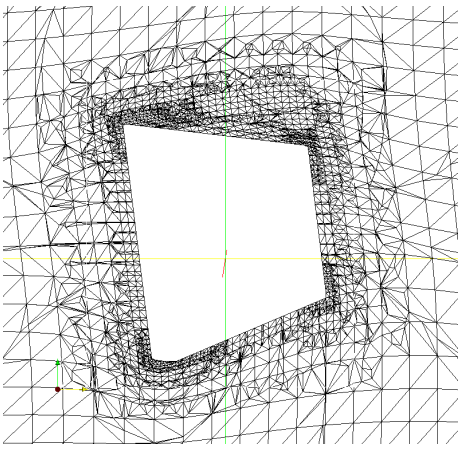

FIGURE 4.3: the xy-plane after deformation. The shape of the box is not changed only translated and rotated.

4.4 Programming

The aim is to move the points on the box and the cells neighboring it by describing them explicitly to the *pointDisplacement* and *cellDisplacement* fields, using the ==-operator at each time step of the solver. This is implemented in the file *krafter.H*. The ==-operator assigns a field of vectors describing the accumulated displacement up to that time step. This means that the diffusion algorithms described in [17] can be used in the mesh. This is important as it is not possible to move only a few points in the mesh without destroying the shape of the cells. The neighboring points need to be moved as well and this is done by the *motionFvSolvers* by a diffusion algorithm.

4.4.1 Algorithm

The following algorithm describes the extraction of the forces and the implementation of the mesh moving scheme. More in-depth explanation will be given in the following section.

while Time is running **do**

Extract forces and angular momentums from pressure, buoyancy.

Extract forces and angular momentums from viscous effects.

Divide by mass for forces and inertia for momentum.
Extract the acceleration by multiplication of the real time interval.
Reset and move center of gravity.
Translate patch to origin.
Rotate in accordance to momentum.
Translate back to origin.
Translate according to forces.
end while

4.4.2 Implementation

Four files are added to the original version of *interDyMFoam* in OpenFOAM-1.5: *initializeForceBalance.H*, which initializes the variables and reads the motion fields, *readForceBalanceControls.H*, which reads from the *forceFoamDictionary*, *krafter.H*, which calculates the forces and angular momentums and moves the mesh and finally *pEqnGravity.H*, which reconstructs the pressure. In the file *krafter.H* one finds the implementation of the forces and angular momentums and the extraction of the displacement by translation and rotation and finally the assignment of these displacements to the mesh.

The forces are calculated in *krafter.H* as follows: the buoyancy force from the pressure is summed up for the patch and then multiplied by the area vectors of the patch, resulting a pressure force with a given direction. For the angular momentums the center of gravity *CgCenter*, is subtracted from the position of the points on the patch so that the moment is about the center of gravity rather then just around the origin. The gravity force is the product of the mass and the vector representing the magnitude and direction of the gravity acceleration. Note that the patch is built from faces which are built from points, the cells concerned are the ones that are neighboring the patch and have one face that is a boundary face of the same. The *Sf()*-function in the extraction of the pressure forces returns the area vectors for the faces. $Cf()$ returns the face centers. The viscous force and moment are extracted from the velocity field where *magSf()* is used to bring out cell face area. These three functions are called from fvMesh, which is the base class for the dynamicFvMesh which is the type of mesh used in simulations with *interDyMFoam*. The forces and angular momentums are added together and later called *mometStatic* and *forceStatic* in krafter.H. From these the accumulated displacement and angle are calculated.

At initialization, the original positions of the points and cells are extracted and assigned to the fields *tempPoint* and *tempCell*. First copying these to temporary fields on which the rotation and transformation is performed. The temporary field is then translated to the origin by subtraction from the center of gravity and by using the conversion from quaternion to a rotational matrix given in (4.4). This produces a new temporary field. The field is then translated back to its original position. This field is then translated by adding a vector to its position and then the original field is subtracted from the one obtained after translation to extract the displacement. Finally the center of gravity is translated to its new position. See appendix A for the Dictionaries involved in the simulations with this solver.

4.4.3 Possible Alterations

There are a few parameters that are known to have an impact on the performance of the system. Firstly, different settings with the initial configuration of the water can be examined. In *system/setFieldsDict* there are two *boxToCell* regions and the latter of the two can be changed by changing the size of its bounding box. This will directly influence how the box moves due to external forces and angular momentums. In *constant/forceFoamDict*, the gravity will not influence the rotation. Changing the weight of the box will define how much the box is submerged. Secondly, the *accelLim* can be set to limit the displacement. The third parameter which can be changed is the under-relaxation factor, see (4.1) which is found in the *system/fvSolvers* file at the very end. Adding to these, a number of possible alterations the mesh can be performed. The mesh can be refined or altered in different ways. Using a finer mesh will make it the simulation run more slowly and it will also be more sensitive to displacement. In the mesh folder in *system/snappyHexMeshDict* the parameters for the *snappyHexMesh* are set. Setting the level for the surface refinement differently will change the mesh structure close to the box.

4.5 Results

4.5.1 Mesh

A mesh was generated in *snappyHexMesh*, first using *blockMesh* to create a background mesh with size 40x40x40 meters and *snappyHexMesh* placed a box as a searchable surface with the size 6x6x6 meters in the center of the domain, depicted in Figure 4.4. The *snappyHexMesh* utility created the box without any refinement box in order to keep the cell size fairly large. Since a small cell will suffer more from skewness given a certain displacement, a larger cells are preferred. Cells used in case A, B:

```
Number of cells of each type:
    hexahedra:     85552
    polyhedra:     5679
```

Case C:

```
Number of cells of each type:
    hexahedra:     510272
    polyhedra:     0
```

There are some 6 per cent cells that are not hexagonal in the coarser grid. This will affect the quality of the solution close to the box. The polyhedral cells are not regular and are used in the transition between the triangular elements on the surface to connects with the hexagonal cells in the initial mesh. It can be seen on figures with the box that the iso-surface, for $\gamma = 0.5$, close to the box has a rough texture. Using a VOF-method it is important to have hexagonal cells for the quality of the solution. The polyhedral cells are boundary cells and thus closer to the box,

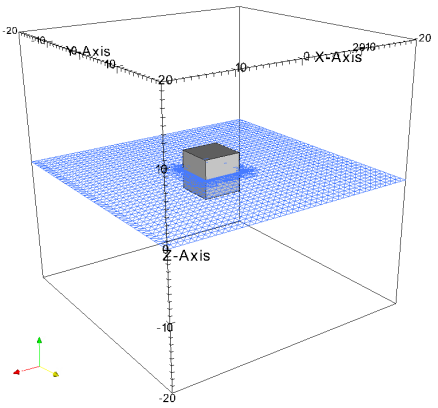

FIGURE 4.4: The domain and the surface of the water.The water fills the domain from -20m to 0m along the z-axis

which causes the iso-surface to look strange in a region close to the box. How much this actually affects the quality of the solution is not clear.

4.5.2 Boundary Conditions

For all the patches, the pressure p, the dynamic pressure p_d and γ are assigned the *zeroGradient* or the Neumann boundary condition. This is because all the patches are considered solid walls. For the velocity U, the box-patch was assigned the *movingWallVelocity*-Boundary Condition, which accounts for the movement of the fluid as well as for the movement of the box-patch, to make sure that these there was no flux into the box. Without using this boundary condition, the solver will interpret this as if the wall is not moving and let the flux pass to positions where there actually is a box. All the other patches in $\boldsymbol{U}$ are given the boundary condition slip, which when the base patch is *patch* works as free slip with no flow in the direction normal to the patch. This will also reflect incoming waves. It can be added that if the *wall* base patch is used, the cells next to the wall will also be considered as wall as well, which can have effects on the flow. The patches in the two new files added to use the solver, cellDisplacement and pointDisplacement are all assigned *fixedValue* and *uniform (0 0 0)*, this is also called the Dirichlet boundary condition.

4.5.3 Initial Values

In order to create some movements in the fluids an extra region of water was placed on top of the one covering half the domain, see Figure 4.10. This was done by setting the second region in *system/setFiledsDict* a bit above the first region. The height of this region will determine how the box moves due to the forces from the water on the box. The box is at time zero submerged to fifty percent in the water with its vertical sides orthogonal to the water surface. The under-relaxation factors was set to 0.7 for U and 0.5 for p and γ so to weight in the old results and stop from to sharp jumps in the solution.

4.5.4 Case A: a calm surface

If a box is floating on perfectly still water there should be no reason for the box to flip over. In this case it is a rather unstable system where the center of gravity is aligned with the water surface. The only force applied to the box is the gravity force which is homogeneous over the box. The pressure forces and viscous forces will keep the box floating. In the simulation the box was placed on the water surface and the simulation ran for 16 seconds until the solver diverged due to cell deformation. The box had flipped over. In Figure 4.5 the initial set up is shown and as can be seen the surface is perfectly calm. The solver has a tendency to, due to numerical discrepancies and possibly a low mesh quality, introduce disturbances. In Figure 4.6 the solver starts moving the box and in Figure 4.7 it diverges. The reason for this behavior could be linked to the mesh quality and the stability of the solver. This motivates some sort of damping further than the one constructed for the acceleration or using another approach than mesh deformation to handle the mesh movement.

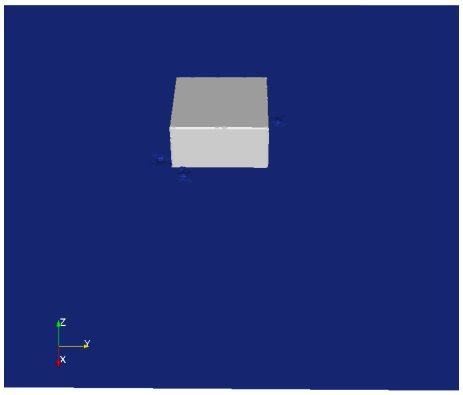

FIGURE 4.5: Case A: initial condition, undisturbed surface.

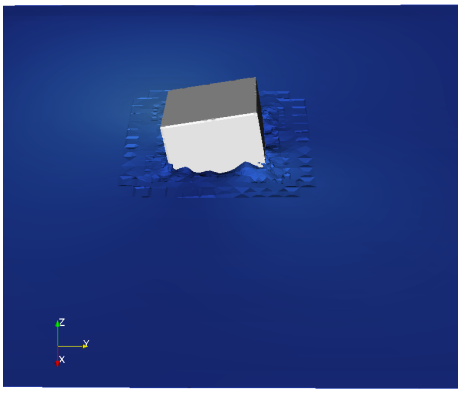

FIGURE 4.6: Case A: after 8 seconds, the surface is disturbed from numerical discrepancies.

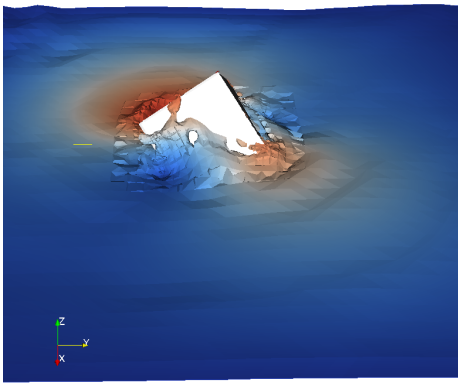

FIGURE 4.7: Case A: the box and surface just before divergence, the numerical scheme and the mesh is not accurate enough.

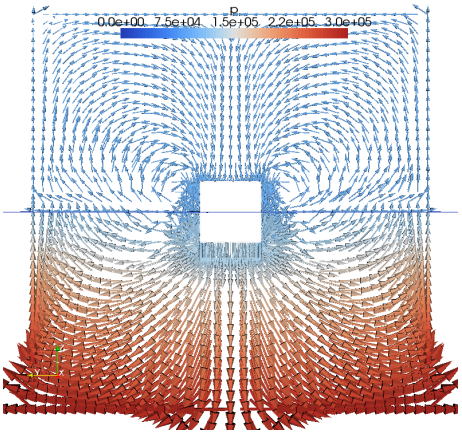

FIGURE 4.8: Case A: Slice in the yz-plane with pressure and velocity glyphs at the first time step

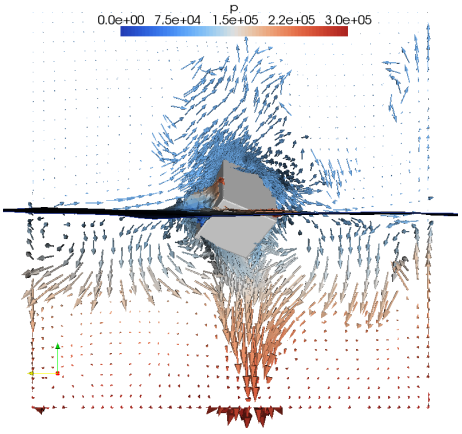

FIGURE 4.9: Case A: Slice in the yz-plane with pressure and velocity glyphs at the last time step.

The pressure is visualized at the first, Figure 4.8 and at the last timestep, Figure 4.9. Here a yz-plane through the cube with pressure and velocity vectors shows the pressure distribution. The line through the cube at $y = 0$ is the surface of the water.

4.5.5 Case B: a coarse mesh

When setting the initial value to be a static wave or rather three meter high region covering part of the domain as can be seen in Figure 4.10 the box is behaving as one could expect. It remains symmetric when the wave passes, Figure 4.11. In Figure 4.12 one can see the box tilt due to the forces from the water and start to tip over on the side in Figure 4.13. Further in Figure 4.14 and in Figure 4.15 the box starts rotating around the Z-axis and tilts that much that the solver diverges due to skewed cells, Figure 4.16.

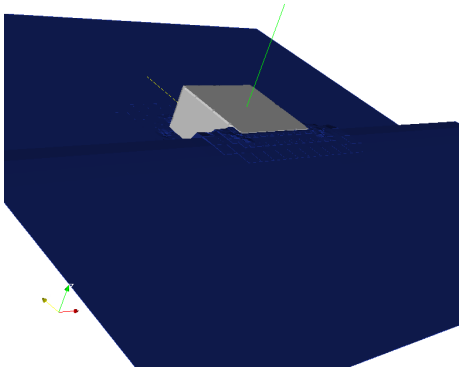

FIGURE 4.10: Case B: initial condition where part of the surface one is elevated.

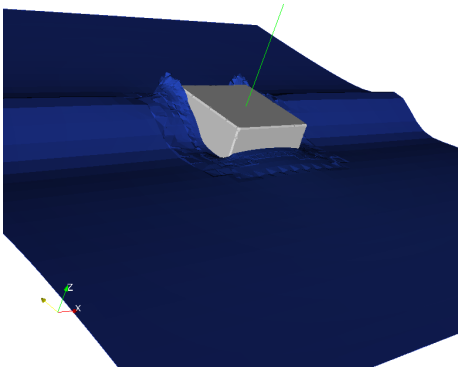

FIGURE 4.11: Case B: the wave just passes the box which is still stable

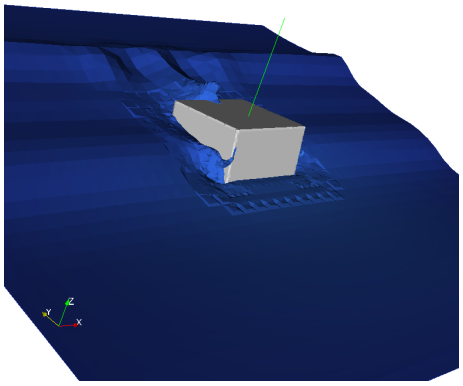

FIGURE 4.12: Case B: the box starts tilting a bit due to the forces from the wave

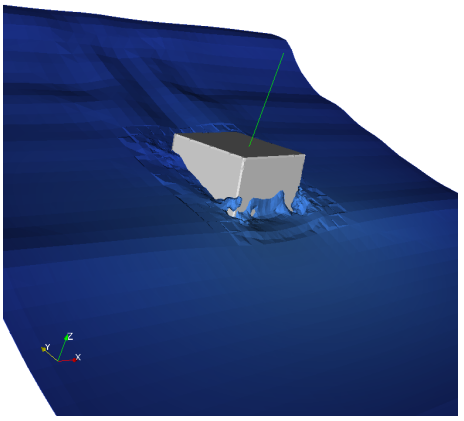

FIGURE 4.13: Case B: the box leans over to the left

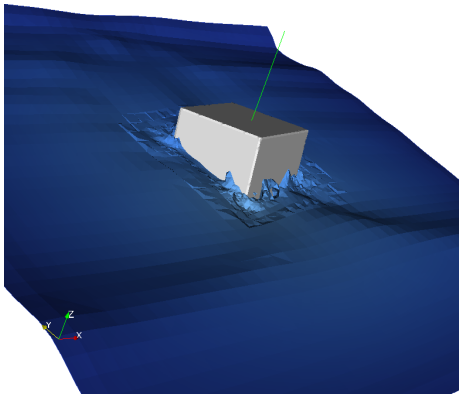

FIGURE 4.14: Case B: the box starts to rotate

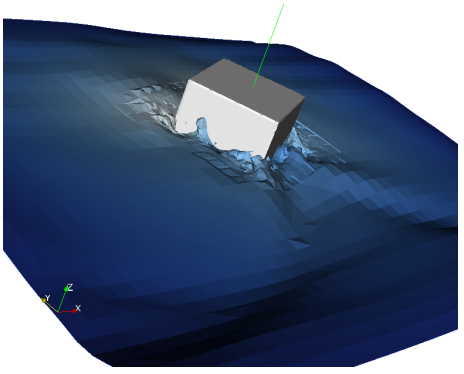

FIGURE 4.15: Case B: the box has now rotated to a critical stage

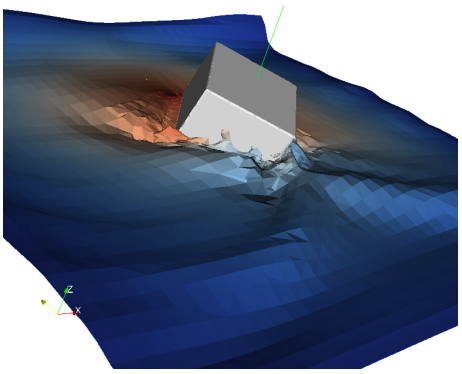

FIGURE 4.16: Case B: The box just before divergence.

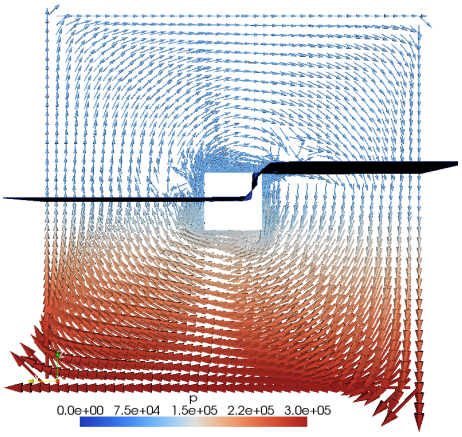

FIGURE 4.17: Case A: Slice in the yz-plane with pressure and velocity glyphs, velocity vectors at the first time step

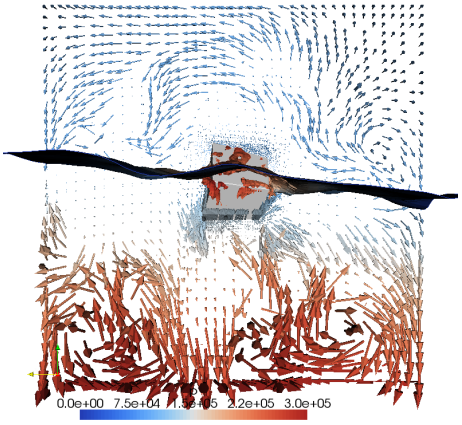

FIGURE 4.18: Case A: Slice in the yz-plane with pressure and velocity glyphs, velocity vectors at the last time step.

4.5.6 Case C: a fine mesh

A mesh with cells, one fourth of the volume of the original ones, with 0.25 m length was used for this simulation. The result is similar but the resolution was better on the iso-surface. A region

with a height of three meters covering the part of the domain next to the box was set as the initial condition, see Figure 4.19. The box behaves in much the same way as in the case with fewer cells. In Figure 4.20 the wave has passed the box and just hits the opposite wall where it is reflected and in Figure 4.21 it returns from the left again and hits the box a third time. Here the box is already starting to tilt and rotate and the fourth time it hits, Figure 4.22 where water covers part of the side which is facing upwards, the solver again diverges.

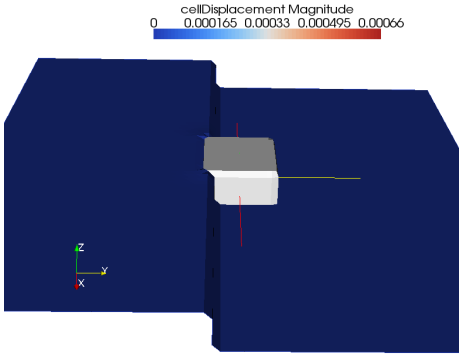

FIGURE 4.19: Case C: initial condition, the static wave has a height of 3 meters

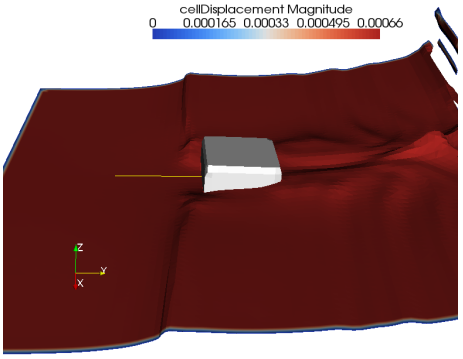

FIGURE 4.20: Case C: the wave hits the opposite side and the box is tilting

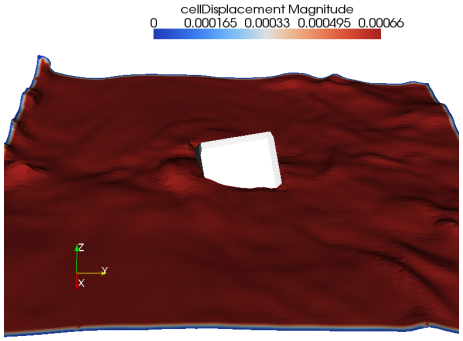

FIGURE 4.21: Case C: the wave hits again for the third time, coming from the left

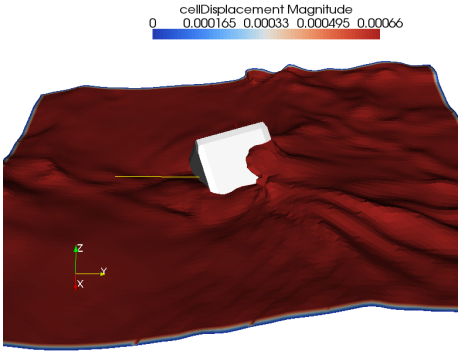

FIGURE 4.22: Case C: wave hits the fourth time and now sinks the box and the solver diverges

4.6 Discussion

This sort of set up, using a cube that floats on water would be an unstable system in any real world experimental set up as well, considering that half the cube is submerged in the water and the center of gravity is right at the level of the surface. It describes though, how the box moves according to the forces induced by the wave hitting it. If we consider a case where no wave is

induced and the water surface is perfectly still the box would be left balancing on the surface, Figure 4.5 shows the surface before the simulation starts.

4.6.1 Divergence

The solver diverges after about 10 seconds for the case with the coarse mesh. This is a result of a deterioration in mesh quality which is caused by the moving patch. In *paraFoam* there is a filter called mesh quality and when applying this with the setting hexagonal - skew it shows that the cells close to the patch are severely deformed in the last time step. This could probably be helped by adding some kind of damping or stabilization scheme to the solver.

4.6.2 Future Work and Improvements

A lot of work and further refinements can be done to make this code more accurate and more efficient, both when it comes to the mesh handling as well as the VoF-parameters, for example, the under-relaxation factors could be optimized. One of the solutions considered was to have some kind of spring damping system for both rotation and translation to stop the box from moving more than the mesh allows before its quality deteriorates. Another possibility would be to use a lower Courant number and thereby shorter time steps. This could improve the accuracy somewhat but at the same time it would require longer simulations. Further, here a mesh generated in *snappyHexMesh* was used without a refinement box. Using the simpler *blockMesh* utility would even allow for larger cells close to the surface and therefore provide a more forgiving mesh. On the other hand the resolution of the boundary layer close to the box would be too coarse to calculate the viscous forces. Since the displacement has to be related to the cell size, a move with small cells has a larger impact on the mesh quality than if the mesh is coarse.

Chapter 5

Conclusion

Here some last concluding notes are made to sum up the work. The result from the two previous parts could be used to assemble a tool for marine applications. With some script language to assemble the different parts into a package that can be used to solve continuous mechanics problems involving two phases and a moving mesh.

In Chapter 3 automatic meshing scheme *snappyHexMesh* was examined and it can be said to give reasonable results. Much of the meshing in CFD is made by hand and often the meshing skill of the person will affect the result. The work involves to a large extent visual examination of the mesh and adjustments are made where needed. Since *snappyHexMesh* only relies on text based input it is often more difficult to make adjustments. Though extensions to the code are planned which allow for specifications of refinements based on certain lines of points of the StL.

The calculation of the drag coefficient can be improved by addition of cell layers close to the cylinder. In the current set up no layers are used since it is not clear if these are correctly set up in the current implementation of *snappyHexMesh*. The idea of the layers would be to create a number of layers of very fine cells close to the cylinder before letting the cells expand to meet the background mesh.

The implementation of the VoF-solver, described in Chapter 4, was by far the more time consuming part of the work. It involved such difficulties as where in the code library to implement the extraction of the forces, move the mesh and update the pressure. After consulting the forum [2] on the subject, it seemed most promising to work directly in the solver code rather than rewriting any boundary conditions or constructing a new kind of mesh.

The set up used in this report has some disadvantages. The center of gravity is just aligned with the water surface and this results in a neutral but very sensitive system. All three simulations have been run with this set up and though it shows the basic idea of the solver it certainly would be interesting to examine a set up where the box is submerged more than 50 percent in the water. The instability of the system can clearly be seen in the experiment with the calm surface, Case A, where he box flips over after some time steps. Before flipping over the solver shows some signs of low quality when for no particular reason starts filling the cells above the surface with water. This is due to deformed cells. It makes it look as if water is shooting out of

the calm surface and onto the box, a strange behavior which seems to be present in commercial codes as well when using a non-uniform grid.

Some notes on what to think about when running the solver used in Chapter 4. It is important is to match the resolution of the mesh with the need for mesh movement. It became clear that if using a fine resolution in the grid near the moving object, the box in Chapter 4, then the solver will be more prone to diverge since the cells are skewed.

The difference between Case B which is the coarse mesh and Case C which has a fine mesh is that the resolution in C is fine enough to use only hexagonal elements. This improves the performance of the solver and as can be seen in the figures the water surface has a smoother texture close to the box.

In short, this free open source library offers the possibility to assemble a library of tools for continuous mechanics. It is unfortunatly not well explained and documented how to implement new tools nor is there much information on how to use the exising ones apart from some tutorials. But the code is free and there is plenty of information to be gathered if searching the forum and reading the source code. With a bit of time one can build tools to accommodate many needs by building libraries for specific purposes. If the problem one is facing as a CFD engineer often calls for a similar type of simulations it could be an economically efficient approach to use OpenFOAM.

Bibliography

[1] Henrik Rusche. *Computational Fluid Dynamics of Dispersed Two-Phase Flows at High Phase Fractions*. PhD thesis, Imperial College of Science, Technology & Medicine, London, 2002.

[2] OpenCFD. Openfoam forum. *OpenCFD WebSite*, 2009. URL http://openfoam.cfd-online.com/cgi-bin/forum/discus.cgi.

[3] *OpenFOAM-1.5 user's guide, OpenCFD, London*, 2008.

[4] *OpenFOAM-1.5 programmer's guide, OpenCFD, London*, 2008.

[5] *ANSYS Fluent 6.3 user's guide*, 2008.

[6] I. E. Barton. Comparison of simple- and piso-type algorithms for transient flows. *International Journal for Numerical Methods in Fluids*, 26:459–483, 1998.

[7] *Best Practice Guidelines for Marine Applicaion of Computational Fluid Dynamics, WS Atkins Consultants, 2002.*

[8] D.F.L Labb and P.A. Wilson. A numerical investigation of the effects of the spanvise length on the 3-d wake of a circular cylinder. *Journal of Fluids and Structures*, (23):1168–1188, 2007.

[9] K.C. Ng. A collocated finite volume embedding method for simulation of flow past stationary and moving body. *Computers & Fluids*, (38):347–357, May 2008.

[10] J.F. Ravoux, A. Nadim, and H. Haj-Hariri. An embedded method for bluff body flows: Interactions of two side-by-side cylinder wakes. *Theoretical Computational Fluid Dynamics*, (16):433–266, July 2003.

[11] S.Kang. Uniform-shear flow over a circular cylinder at low Reynolds numbers. *Journal of Fluids and Structures*, (22):541–555, 2006.

[12] Meng-Hsuan Chung. Cartesian cut cell approach for simulating incompressible flows with rigid bodies of arbitrary shape. *Computers & Fluids*, (35):607–623, April 2006.

[13] Frank M White. *Fluid Mechanics*. McGraw-Hill, 1994.

[14] R.M. Barron and Ali Asalehi Neyshabouri. Effects of under-relaxation factors on turbulent flow simulations. *International Journal For Numerical Methods in Fluids*, (42):923–928, 2003.

[15] Roozbeh Panahi, Ebrahim Jahanbakhsh, and Mohammad S. Seif. Development of a VOF-fractional step solver for floating body motion simulation. *Applied Ocean Research*, 28(28): 171–181, August 2006.

[16] Hrvoje Jasak and Zeljko Tukovic. Automatic mesh motion for the unstructured finite volume method. *Transactions of FAMENA*, 30(2):1–23, 2007.

[17] Pirooz Moradnia. A tutorial on how to use dynamic mesh solver icodymfoam. *PhD course in CFD with OpenSource software*, Fall, Gothenburg 2007.