CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

ship hull response in cylBumpInterIbFoam tutorial

Developed for FOAM-extend-4.0
Requires: swak4Foam Library
bison with version of 2.4.1 or older
OpenFOAM

Author:
Mohsen Irannezhad
Chalmers University of Technology
imohsen@student.chalmers.se

Peer reviewed by:
Varun Venkatesh
Håkan Nilsson

January 20, 2017

Contents

Learning outcomes

The main requirements of a tutorial is that it should teach the four points: How to use it, The theory of it, How it is implemented, and How to modify it. Therefore the list of learning outcomes is organized with those headers.

The reader will learn:

How to use it:

- How to use the Immersed Boundary Method in order to simulate the flow around the ship hull in FOAM-extend-4.0

The theory of it:

- The theory of Immersed Boundary Method and its applications
- Different IBMs and their pros and cons

How it is implemented:

- How to setup an IBM test case in FOAM-extend-4.0, in this case a ship hull responding to inlet flow
- IBM implementation in FOAM-extend-4.0 for both laminar and turbulent flows

How to modify it:

- How to propagate waves at the inlet of the computational domain in FOAM-extend-4.0
- How to refine the mesh around the immersed body for high Reynolds number flow simulations

Prerequisites

The reader is expected to know the following in order to get maximum benefit out of this report:

- Run standard document tutorials like `damBreak` tutorial
- The least square curve fitting method and the least square weighting method
- The basic principles of ship hydrodynamics

Introduction

The fluid flow around an immersed body, e.g. a ship hull, and the body response to the forces exerted by the flow , e.g. sea-keeping responses, can be studied using Computational Fluid Dynamics (CFD). The most obvious practise is to adapt a body-fitted computational grid to the hull in which the hull geometry is sharply represented by the grid points, see Figure 1. In this method, the boundary points are located on the geometry and application of the boundary conditions is straight-forward.

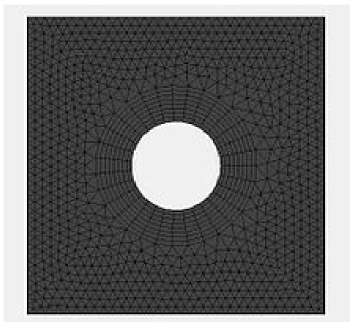

Figure 1: Boundary fitted mesh, mesh is conformal to the boundary

However, in many practical applications the geometry of the immersed body is very complicated and adapting a conformal good quality mesh to it is very difficult and time consuming if even possible. Moreover, the immersed body might move or its shape might deform during time, which implies that the body fitted mesh needs to be re-calculated at each time step. These will add even more complexity to the meshing process and increase the cost of such simulations. There are several methods to address these problems, such as general grid interface, over-set mesh, mesh morphing, Immersed Boundary Method (IBM), etc., which are used for different purposes and are suitable for different problems. The present work describes and uses the IBM method for sea-keeping simulations.

Chapter 1

Immersed Boundary Method

IBM is a method which was originally proposed and developed in the 70s by Peskin [1] in order to analyze biomedical flows. Today, IBM is used for several Fluid-Solid Interaction (FSI) problems and is a suitable solution for analyzing the sea-keeping responses of the ship hulls.

There are two meshes involved in a CFD simulation involving IBM. The so called background mesh is a well-defined and usually simple Cartesian mesh which contains the whole computational domain and ignores the existence of the immersed body. The immersed body is then represented by a second mesh, usually a surface mesh adapted to its exterior. This surface mesh is water tight, meaning that there are no holes in it hence completely separating the body interior from the fluid.

The inclusion of the surface mesh into the background mesh divides the background mesh cells into three distinctive cell types, see Figure 1.1. live cells (or fluid cells) are the ones which are completely in the fluid domain, dead cells (or solid cells) are completely inside the body and Immersed Boundary cells (IB cells) are intersecting the surface mesh and hence the immersed body. The nodes on the surface mesh are called the IB nodes.

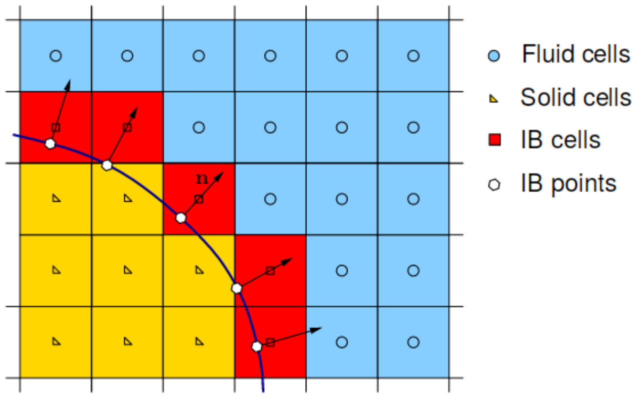

Figure 1.1: Cell types in IBM. Adapted from "Immersed Boundary Method in FOAM" by H. Jasak, 2015 [2]. Reproduced with permission of [2]

The governing equations of the flow are solved in the live cells of the Eulerian field of the background mesh and the position and shape of the surface are tracked by a Lagrangian representation of the body. In this way, a motion of the body is easily addressed by re-positioning of the body in the background mesh, and possible deformations of the body surface are addressed by re-doing the surface mesh. This is far easier than the re-meshing needed for a boundary fitted method. After each surface re-meshing and/or re-positioning of the body the cell types should be re-evaluated.

1.1 Boundary Conditions in IBM

One major problem in IBM is that the fluid equations are solved on the background mesh but there are no background mesh nodes on the immersed boundary to directly apply the boundary conditions to them. The method of the boundary condition implementation distinguishes different IBMs from each other. Different IBM methods are briefly explained here and was discussed in more detail detail by R. V. Meulen [3].

1.1.1 Continuous Forcing IBM

The original IBM suggested in the 70s was of the Continuous Forcing IBM type. In continuous forcing IBM (CFIBM) the effects of the boundary are directly applied through force/source terms in the governing equations, for instance f in the momentum equation 1.1, prior to discretization.

$$\rho\frac{\partial u}{\partial t} + \rho u \nabla . u = -\nabla p + \mu \Delta u + f \tag{1.1}$$

This greatly simplifies the implementation of the boundary conditions and gives a continuous force across the immersed body boundary. This method is proven very efficient when dealing with bodies which deform easily by forces applied to them. However, in case of rigid bodies such as a ship hull this method results in a stiff set of equations which are difficult to solve. Moreover, this method has been reported to be numerically unstable and inaccurate.

1.1.2 Discrete Forcing IBM

Discrete Forcing IBMs (DFIBMs) apply the boundary conditions by modifying the already discretized set of equation. In contrast to CFIBMs, DFIBMs are scheme-dependant which gives more control over accuracy and stability of the simulations. There are two major categories of DFIBMs: Direct Forcing DFIBMs and Indirect Forcing DFIBMs.

Indirect Forcing DFIBM

In the Indirect Forcing category of DFIBMs, the boundary conditions are not directly included in the discretized equations. Instead, they are added as discretized force/source terms. Unlike the source terms in CFIBMs in which the force term has a mechanical/physical nature, the source term in these methods are calculated from the desired boundary conditions. The force term is also continuous across the boundary in this family of methods and the immersed body boundaries are not sharply represented.

Direct Forcing DFIBM

In the Direct Forcing category of DFIBMs, the boundary conditions are directly implemented into the discretized equations through the IB cells and the immersed body geometry is sharply represented. Therefore, the forces are not continuous across the boundary.

1.2 Dead-to-Live Cells

Another major problem with IBM occurs when the deformation or movement of the boundary results in dead cells becoming live in the next time step. Addressing this problem in CFIBMs and indirect forcing DFIBMs is easier due to the continuous nature of the applied forces/sources across the geometry. One way to solve this problem in direct forcing DFIBMs is by simply assuming that this cell has the conditions of the closet live cell for the first time step after it comes to life.

IBMs comparison

The pros and cons of the above mentioned IBMs can be summarized as below:

- CFIBMs
  - Pros
    - Easy to implement
    - Good when bodies deform or move easily with force
    - Good when the mechanical force is well defined
    - Good in handling dead to live cells
  - Cons
    - Accuracy and stability issues
    - Stiff equations when the body is rigid
    - Bad at high Reynolds numbers due to lack of sharp representation of the body surface
- Indirect Forcing DFIBMs
  - Pros
    - Good for rigid bodies
    - Good in handling dead to live cells
    - Discretization scheme-dependent, hence more control over accuracy and stability
  - Cons
    - More difficult to implement
    - Bad at high Reynolds numbers due to lack of sharp representation of the body surface
- Direct Forcing DFIBMs
  - Pros
    - Good for rigid bodies
    - Discretization scheme-dependent, hence more control over accuracy and stability
    - Good at high Reynolds number flows
    - Sharpness of the boundary is preserved
  - Cons
    - More difficult to implement
    - Bad at handling dead to live cells

1.3 IBM in FOAM-extend

The IBM implemented in FOAM-extend is of the Indirect Forcing DFIBM category and is described by J. Favier et al. [4] and H. Jasak [2]. The idea is to skip solving the fluid governing equations in the IB cells and instead approximating the values in the IB cells use boundary values and values in neighboring live cells. This can be seen as moving the boundary from the actual body geometry to the IB cell centers surrounding the body surface. The governing equation are solved as usual after this process. The approximation process is the same for all equations except for the pressure and are described in the following.

1.3.1 IB Cell Value Approximation

Approximating the values in the IB cell centers for all equations in FOAM-extend, except the pressure equation, is based on the assumption that all quantities follow a quadratic behaviour near the boundary. This assumption, together with a least square weighting function, is used to approximate the values in the IB cell centers for both Dirichlet and Neumann boundary conditions.

Dirichlet Boundary Conditions

In case of a Dirichlet boundary condition, such as no-slip condition on the walls, the value in the IB cell center is approximated by

$$\phi_P = \phi_{ib} + C_0(x_P - x_{ib}) + C_1(y_P - y_{ib}) + C_2(x_P - x_{ib})(y_P - y_{ib}) + C_3(x_P - x_{ib})^2 + C_4(y_P - y_{ib})^2 \quad (1.2)$$

The unknown coefficients in the equation above are found through least square curve fitting to the neighboring stencil cells, see Figure 1.2.

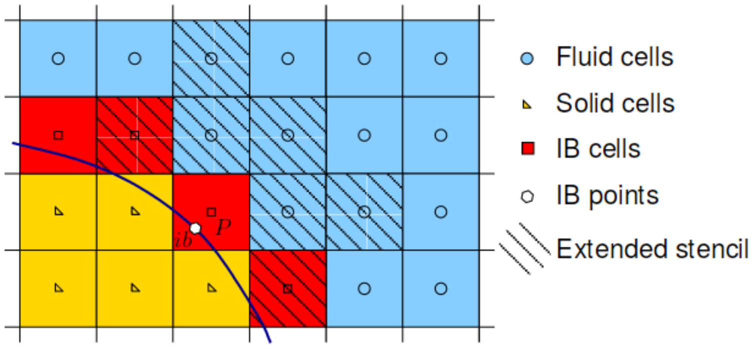

Figure 1.2: IB cell center approximation in Dirichlet BC. Adapted from "Immersed Boundary Method in FOAM" by H. Jasak, 2015 [2]. Reproduced with permission of [2]

Neumann Boundary Conditions

In case of a Neumann boundary condition, such as adiabatic or specified heat flux walls, the value in the IB cell is approximated through introduction of a local coordinate system, see Figure 1.3.

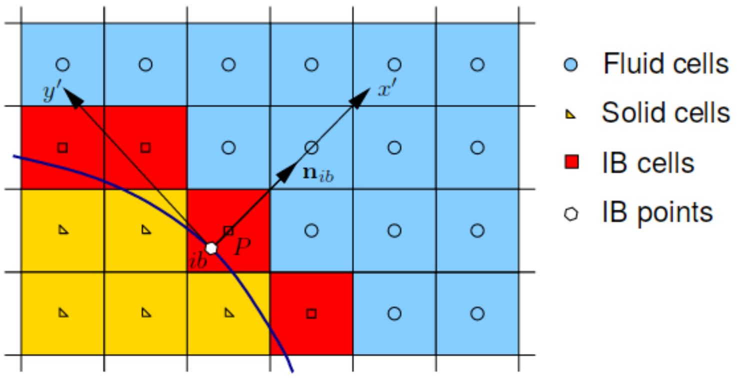

Figure 1.3: IB cell center approximation in Neumann BC. Adapted from "Immersed Boundary Method in FOAM" by H. Jasak, 2015 [2]. Reproduced with permission of [2]

The values in the IB cell is given by

$$\phi_P = C_0 + [n_{ib}.(\nabla\phi)_{ib}]x'_P + C_1 y'_P + C_2 x'_P y'_P + C_3 {x'_P}^2 + C_4 {y'_P}^2 \tag{1.3}$$

The coefficients are found as in the Dirichlet boundary treatment.

Least Square Weighting functions

There are two different least square weighting functions available in FOAM-extend for the purpose of finding the C coefficients discussed above and more details are given by H. Jasak [2].

1.3.2 Pressure Equation

The implementation of the pressure boundary condition is more complicated as the value of pressure is neither known nor needed on the immersed body. The theoretical details of this implementation are out of scope of this work and is described by J. Favier et al. [4].

1.4 IBM class implementation in FOAM-extend

IBM is implemented in FOAM-extend through three classes:

- immersedBoundaryPolyPatch
  - takes care of basic mesh support functions for IBM meshes
- immmersedBoundaryFvPatch
  - supports basic and derived Fv properties of IBM
  - calculates the background mesh and surface mesh intersection points
  - recognizes live and dead cells, normals and distances
  - calculates the interpolation matrices used in imposition of boundary conditions
  - contains information concerning parallel communications framework and layout.
- immersedBoundaryFvPatchField
  - field support
  - calculates and evaluates the boundary conditions using the interpolation matrices
  - evaluates patch fields for IB patches
  - calculates and interpolates field data on mesh intersections
  - takes care of any boundary updates due to movement or deformation

1.5 High Reynolds Number Flows

The assumption of quadratic behaviour of velocity near the body walls is not valid at high Reynolds number flows and leads to inaccuracy. Wall functions are used in body fitted grid simulations to address high Reynolds number flows. In high Reynolds number flows, the velocity profile near the wall follows the log-law and can be analytically evaluated. Therefore, if the mesh is fine enough and the first cell near the wall is located in the log layer, there is no need to solve the momentum equations on these cells and wall functions will give us the velocity.

Implementation of the immersed wall functions in the IBM method in FOAM-extend is however not straight-forward as it cannot be applied to the IB cells. Instead, for each IB cell a sampling point is introduced at a distance equal to 1.5 times the grid resolution at the IB cell in the normal direction to the body. This guaranties that this cell lies in a live cell. The flow properties in this sampling point are then calculated using the neighboring live cells and this sampling point is used to evaluate the adequacy of mesh resolution for resolving the log layer.

In case this sampling point does not lie in the log layer the usual quadratic approximations are used, otherwise the wall functions are applied to the IB cells. It is worth to mention that only the the tangential velocity is calculated using the wall functions and the normal velocity component is always approximated quadratically.

The main drawback of immersed body wall functions, apart from implementation complexity, is that the mesh resolution should be finer than the boundary fitted approach to be able to resolve high Reynolds number flows. The refineImmersedBoundaryMesh utility can be used to refine the background mesh only close to the immersed body hence reducing the total number of cells when higher Reynolds number flows are simulated.

Chapter 2

Test Case

2.1 Introduction

The goal of this part of the project is to simulate the flow around a ship hull using the Immersed Boundary Method in FOAM-extend. In order to achieve this, it is decided to use one similar available case and modify it according to the desired purpose. The closest case is the `cylBumpInterIbFoam` tutorial available in $FOAM_TUTORIALS in foam-extend-4.0. This tutorial is modified and a new tutorial has been made which simulates the flow around a ship hull in two different inlet conditions that can be applied to any ship hull.

cylBumpInterIbFoam tutorial

The `cylBumpInterIbFoam` tutorial is available in FOAM-extend-4.0 tutorials in the `$FOAM_TUTORIALS/immersedBoundary` which is a case concerning the interaction of two fluids and a rigid body. The solver `interIbFoam` is used in this tutorial which uses the Volume Of Fluid (VOF) approach to simulate the fluid-fluid interaction. The original `cylBumpInterIbFoam` tutorial is called "Dam Break Over a Bump", see Figure 2.1. The bump here is a half cylinder at the bottom of the domain, the water and air are represented by red and blue respectively.

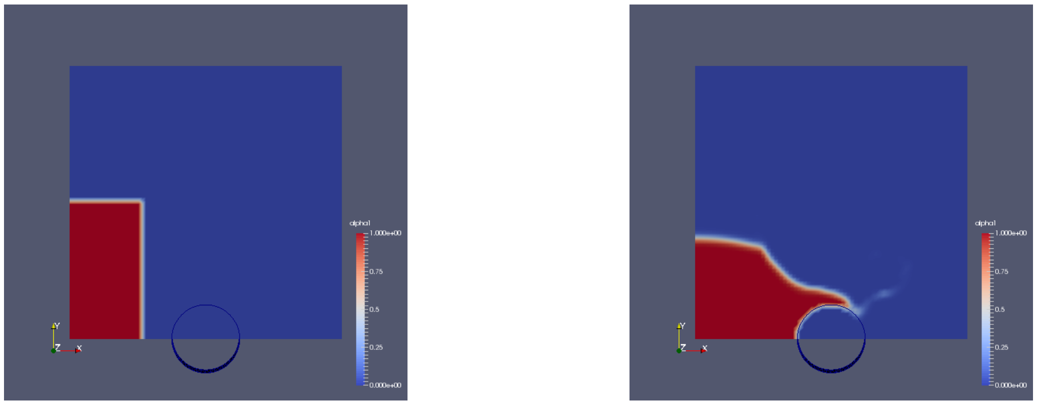

Figure 2.1: Original cylBumpInterIbFoam tutorial fluid field, T=0 (Left) and T=t (Right)

New Tutorial Idea

In order to simulate the flow around a ship hull, the idea is to use the ship hull as a bump and place it in a 3D tank of water and implement the related boundary conditions to the new case using the `cylBumpInterIbFoam` tutorial implementation. So the cylinder obstacle is replaced by the hull geometry, available in OpenFOAM as DTC-scaled, and it is placed in a tank with a specific water level depth, see Figure 2.2. Different boundary conditions are then applied to the inlet. In all the cases there is an inlet, an outlet, a bottom, an atmosphere and two sides which are representing the boundaries of the computational domain in 3D.

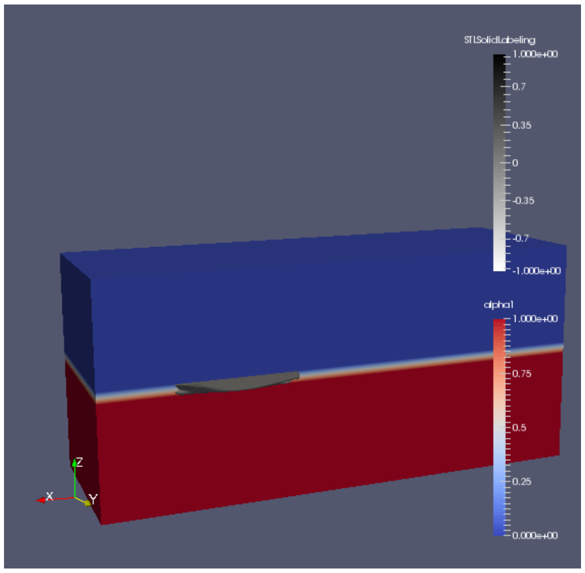

Figure 2.2: DTC-hull in a partly filled water tank (half domain)

The ship hull geometry, DTC-scaled.stl.gz, is available in OpenFOAM: $FOAM_TUTORIALS/resources/geometry

2.2 Workflow

The simulation procedure is presented in a more general way here while the details of each step are explained in the following sections. It should be noted that some general steps are already available in the `cylBumpInterIbFoam` tutorial and only the modifications are explained here. Two different inlet velocities are presented which results in two cases which are referred as Constant Velocity and Wave Propagating Wave cases. The necessary steps are summarized as below:

- Copy the `cylBumpInterIbFoam` tutorial to the run directory
- Replace the `ibCylinder` geometry in the `constant/triSurface/` directory with the hull geometry
- Define a new volume mesh using the `blockMeshDict` dictionary in `constant/polyMesh/`
- Create `polyMesh/boundary` using `blockMesh` utility
- Include the `immersedBoundaryPolyPatch` into the `polyMesh/boundary`
- Refine the volume mesh around the immersed boundary using the `refineImmersedBoundaryMesh` utility and copy the new boundary and patches to the `constant/polyMesh` directory (optional, used for high Reynolds flow simulations)
- Modify properties `constant/g`
- Modify the solution control in `system/fvSolution` dictionary

Constant Velocity Case

- Modify `system/setFieldsDict` dictionary to specify the water field
- Modify the time step size in the `system/controlDict` dictionary
- Modify the boundary fields of alpha1, pressure and velocity by adding the new `immersedBoundaryFvPatchField`
- Modify the `./Allrun` and `./Allclean` scripts and run the case with `./Allrun`
- Post process the case using `paraview`

Wave Propagating Case

- Download, install and utilize the `swak4Foam` library
- Include the new library to the `system/controlDict` dictionary
- Implement boundary conditions for alpha1, pressure and velocity with the `groovyBC` library
- Make a `funkySetFieldsDict` dictionary in order to use the `funkySetFields` utility
- Modify the `./Allrun` and `./Allclean` scripts and run the case with `./Allrun`
- Post process the case using `paraview`

2.3 Case Setup

Two different case setups are implemented in the new tutorial and the results are presented. The geometry is the same in both cases, which is the DTC-scaled.stl in the `constant/trisurface` directory of the tutorial. Another potential geometry could be the `wigley` hull; however, as this geometry is not watertight, it is not used in this project. The solver requires a ".ftr" format file for the immersed geometry which can be created using the `surfaceConvert` utility in OpenFOAM terminal window from its ".stl" format.

- Copy the `cylBumpInterIbFoam` tutorial to the run directory

```
f40NR
mkdir -p $FOAM_RUN
run
cp -r $FOAM_TUTORIALS/immersedBoundary/cylBumpInterIbFoam .
cd cylBumpInterIbFoam/constant/triSurface
rm ibCylinder.ftr ibCylinder.stl
```

- Replace the `ibCylinder` geometry in the `constant/triSurface/` directory with the hull geometry

Then open a new terminal window and continue with OpenFOAM terminal:

```
 OF4x
 cp $FOAM_TUTORIALS/resources/geometry/DTC-scaled.stl.gz $FOAM_RUN
 run
 gunzip DTC-scaled.stl.gz
 surfaceConvert DTC-scaled.stl DTC-scaled.ftr
 mv DTC-scaled.stl hull.stl
 mv DTC-scaled.ftr hull.ftr
```

Table 2.1: The DTC-scaled hull dimensions

Length	Width	Height
6.28m	0.859m	0.572m

Then copy the ".stl" and the ".ftr" format files to the `cylBumpInterIbFoam/constant/triSurface` directory in the FOAM-extend run-directory. The hull dimensions are presented in Table 2.1.

2.3.1 Mesh Generation

In order to simulate the case, the computational domain is meshed with the following utilities. The `blockMesh` utility creates the `polyMesh` with a Cartesian grid (background mesh in IBM) and the `refineImmersedBoundaryMesh` utility is used for refining the mesh around the immersed boundary (for High Reynolds number flows).

blockMesh Utility

The computational domain which is presented by the `blockMeshDict` is modified according to the following. The `blockMesh` utility would then create a `polyMesh` in which the domain is divided into two blocks. One block with finer mesh containing the inlet and the hull to study the ship response and the wave pattern around the ship, and a second block which includes the outlet to the first block in order to study the propagating wave pattern behind the ship (More details are given inside the code).

- Define a new volume mesh using the `blockMeshDict` dictionary in `constant/polyMesh/`

```
f40NR
run
cd cylBumpInterIbFoam
```

Open the constant/polyMesh/blockMeshDict file and change it to the following.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
    (-16   -9     -6)   // the domain generated according to the size of the hull.
    ( -5   -9     -6)   // vertices are places in a way to include enough distance
    ( -5    9     -6)   // from the inlet and to the sides and bottom to apply proper
    (-16    9     -6)   // boundary conditions in 0_org files.
    (-16   -9      6)
    ( -5   -9      6)
    ( -5    9      6)
    (-16    9      6)
    ( 10   -9     -6)
    ( 10    9     -6)
    ( 10   -9      6)
    ( 10    9      6)
);

blocks          // the Domain is divided into two blocks which contain same number of
(               // divisions in y and z directions and different in x direction.
    hex (0 1 2 3 4 5 6 7) (50 50 40) simpleGrading (1 1 1)
    hex (1 8 9 2 5 10 11 6) (80 50 40) simpleGrading (1 1 1)
);

//          4 ---------- 7
//         /|           /|         z
//        / |          / |         ^
//       5 ---------- 6  |         |
//       |  |         |  |         |
//       |  0 --------|- 3         |-------->y
//       | /          | /         /
//       |/           |/         /
//       1 ---------- 2         x

//          5 ---------- 6
//         /|           /| <-- this block contains the inlet and hull while the upper block
//        / |          / |     contains the outlet. Hence, the generated mesh in this block
//       10 ----------11 |     is finer than the other one.
//       |  |         |  |
//       |  1 --------|- 2
//       | /          | /
//       |/           |/
//       8 ---------- 9

edges
(
);

boundary
(
    inlet
    {
        type patch;
        faces
        (
        (9 11 10 8)
        );
    }

    outlet
    {
        type patch;
        faces
        (
        (0 4 7 3)
        );
    }

    atmosphere

    {
        type patch;
        faces
        (
        (4 5 6 7)
        (5 10 11 6)
        );
    }

    bottom
    {
        type patch;
        faces
        (
        (0 3 2 1)
        (1 2 9 8)
        );
    }

    sides
    {
        type patch;
        faces
        (
        (8 10 5 1)
        (1  5 4 0)

        (3  7 6 2)
        (2 6 11 9)
        );
    }
);

mergePatchPairs
(
);
// ************************************************************************* //
```

It should be noted that after running the `blockMesh` utility the following `boundary` file is created in the `constant/polyMesh` directory.

- Create `polyMesh/boundary` using `blockMesh` utility

```
blockMesh
```

Then open the constant/polyMesh/boundary file and look at its contents.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

```
5   // without immersed boundary geometry
(
    inlet
    {
        type            patch;
        nFaces          2000;
        startFace       766300;  // this number will be used in the following
    }
    outlet
    {
        type            patch;
        nFaces          2000;
        startFace       768300;
    }
    atmosphere
    {
        type            patch;
        nFaces          6500;
        startFace       770300;
    }
    bottom
    {
        type            patch;
        nFaces          6500;
        startFace       776800;
    }
    sides
    {
        type            patch;
        nFaces          10400;
        startFace       783300;
    }
)


// ************************************************************************* //
```

The generated boundary file has all the boundary conditions for the domain except the immersed geometry. The next step is to add the `immersedBoundaryPolyPatch` boundary to the created boundary file and make a new boundary file. There is an extra directory in this tutorial named `save` which contains a `boundary` file and a `blockMeshDict` file. The `blockMeshDict` file will be explained further in the report but for now just copy the new created `blockMeshDict` there and replace it with the old one from the `cylBumpInterIbFoam` tutorial.

- Include the `immersedBoundaryPolyPatch` into the `polyMesh/boundary`

As it is discussed in the section 1.4, the `immersedBoundaryPolyPatch` takes care of basic mesh support functions for IBM meshes. The member functions of the ”immersedBoundaryPolyPatch.H” is given by

```
// Member Functions

        // Access
```

```
            //- Return immersed boundary surface mesh
            const triSurfaceMesh& ibMesh() const
            {
                return ibMesh_;
            }

            //- Return true if solving for flow inside the immersed boundary
            bool internalFlow() const
            {
                return internalFlow_;
            }

            //- Return triSurface search object
            const triSurfaceSearch& triSurfSearch() const;
```

The next step is to add `immersedBoundaryPolyPatch` to the created `boundary` file and make a new `boundary` file and replace it with the one from the original `cylBumpInterIbFoam` tutorial in the `save` directory. This is because of the fact that the new `blocMeshDict` creates a new total number of cells which changes the `startFace` of the immersed boundary.

The `startFace` for the geometry (hull) should be modified to the `startFace` in the new `boundary` according to the new total number of cells, while its `nFaces` in this method is always considered as zero irrespective of the geometry. Thereafter, the other boundaries would be attached to the hull boundary. The `boundary` file for the above `blockMeshDict` should be changed to the following in the `save` directory:

```
rm save/boundary
cp constant/polyMesh/boundary save/
```

Open the `save/boundary` file and add the hull as type immersedBoundary with zero faces and the same startFace as the first boundary startFace (inlet in this example). Then since a boundary was added, change the number of boundaries to six.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

6    // Notice this number, includes the hull
(

    hull   // change the name to the .stl file name
    {
        type            immersedBoundary;
        nFaces          0;        // Always considered as 0 for the immersed boundary geometry.
        startFace       766300;   // <-- copy the created startFace in boundary here!
        internalFlow    no;      // There is no internal flow in the immersed body.
    }

    inlet                        // copy other parts of the boundary file here...
    {
        type            patch;   //          |
        nFaces          2000;    //          |
        startFace       766300;  //          |
    }                            //          V
    outlet
    {
        type            patch;
        nFaces          2000;
        startFace       768300;
    }

    atmosphere
    {
        type            patch;
        nFaces          6500;
        startFace       770300;
    }

    bottom
    {
        type            patch;
        nFaces          6500;
        startFace       776800;
    }

    sides
    {
        type            patch;
        nFaces          10400;
        startFace       783300;
    }

)


// ************************************************************************* //
```

- Refine the volume mesh around the immersed boundary using the `refineImmersedBoundaryMesh` utility and copy the new boundary and patches to the `constant/polyMesh` directory (optional, used for high Reynolds flow simulations)

refineImmersedBoundaryMesh Utility

In order to reduce the near-wall y+ for simulating high-Reynolds number flows, the background mesh next to the immersed boundary can be refined. This is one of the techniques that can be used to implement wall functions into the simulation without exra refinement in unnecessary regions. For refining the background mesh around the immersed surface the `refineImmersedBoundaryMesh` utility could be used. There are three different levels of mesh refinement available through this utility. The one level mesh refinement is called `ibCells`, the two level mesh refinement is called `ibCellCells` and the three level mesh refinement is called `ibCellCellFaces`. Figures 2.3 and 2.4 show the results of mesh refinement around the immersed boundary in the free surface for one and three levels refinement. They show that the one level refinement did not give a good mesh at the boundary while the three level refinement results in a nice Cartesian mesh near the boundary.

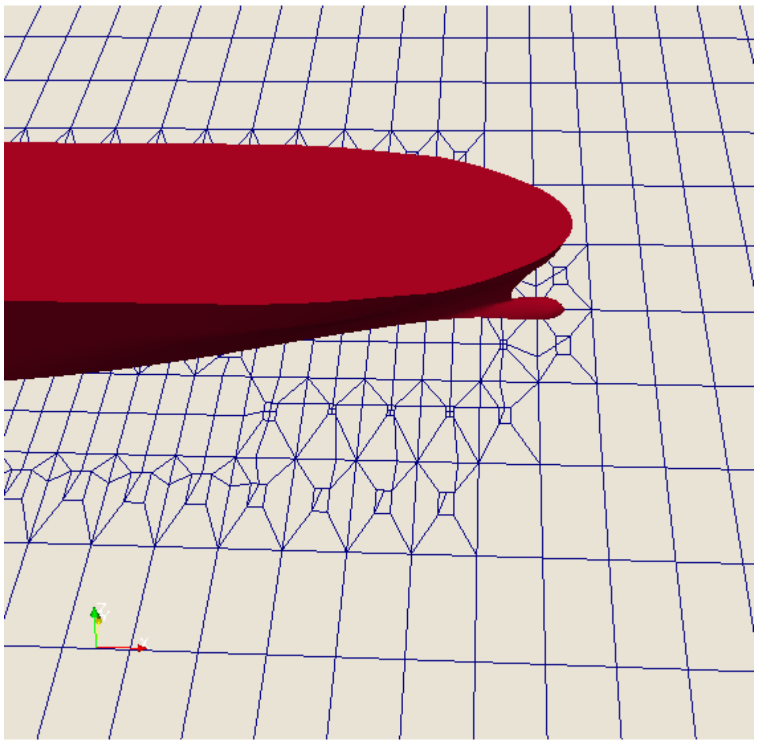

Figure 2.3: One-level Immersed Mesh Refinement (ibCells)

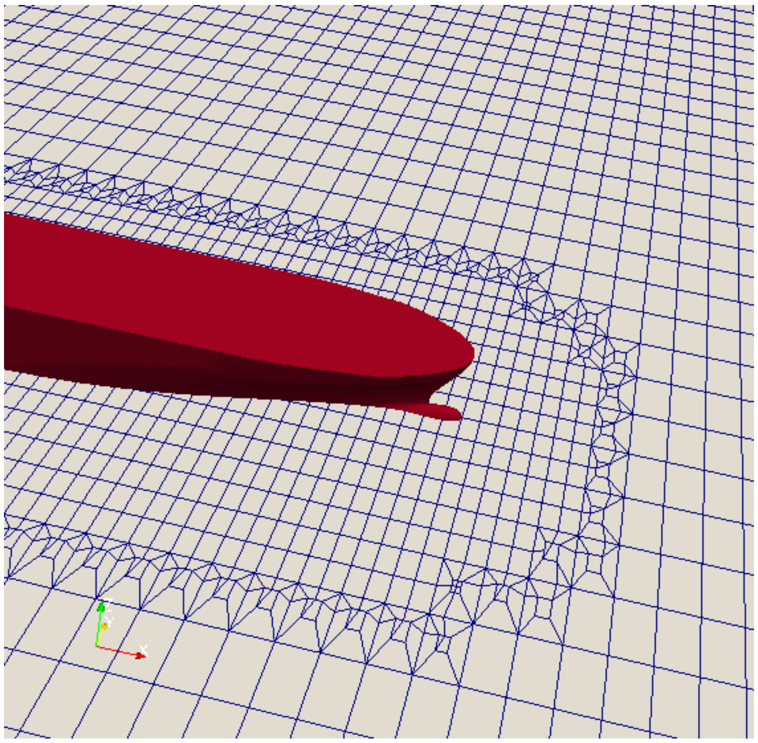

Figure 2.4: Three-level Immersed Mesh Refinement (ibCellCellFaces)

In order to use one of the available options of mesh refinements the `refineImmersedBoundaryMesh` should be run after the `blockMesh` utility with a flag showing the level of refinement. It uses the copied `boundary` file from the `save` directory and forms a new directory `0/polyMesh` in the tutorial folder which contains a new `boundary` file and `neighbour`, `owner`, `points`, and other produced packages. These files should be copied to the `constant/polyMesh` directory just after the mesh refinement process. Then the created directory `0` should be removed and another empty `0` should be built. These steps are available in the `./Allrun` script and will be described again.

2.3.2 Properties

The `constant/g` file should be modified in order to introduce a gravity of 9.81 in the -Z direction for the discussed coordinate system. The coordinate system in the new tutorial is not the same used in the `cylBumpInterIbFoam`. Open the `constant/g` file and modify it as follows.

- Modify properties `constant/g`

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

2.3.3 Solver

The solver which is used for this tutorial is `interIbFoam`. It is a solver for 2 incompressible, isothermal immiscible fluids using the VOF approach with immersed boundary support. The momentum and other fluid properties are of the "mixture" and a single momentum equation is solved.

Volume Of Fluid (VOF)

In computational fluid dynamics, the volume of fluid method is a free-surface (fluid-fluid interface) modelling technique. In this method the phase fraction of each fluid is calculated in each cell to track and locate the cells in which the two fluids are interacting. Then the located cells are creating the free surface between the two fluids. It takes care of fluid shape in a local domain and reconstructs the interface from volume fraction of one fluid, maintaining sharp interfaces. The numerical calculation procedure of this method is out of the scope of this project, however the method is used in the `interIbFoam` solver to model the fluid-fluid interface.

turbulence modelling

In the `interIbFoam` solver the turbulence modelling is generic, i.e. laminar, RAS or LES may be selected. However, all the efforts made in this work to introduce turbulence in the flow failed and this tutorial will only concern the simulation of the laminar flow. It could be as a future work to change the conditions to turbulent and resolve the problems. One such problem might be the insufficient mesh resolution near the immersed body despite all the refinements presented above. As mentioned earlier, the mesh resolution requirements for use of wall functions are much stricter for IBM which might be the major source of lack of turbulence. Therefore, `constant/turbulenceProperties` remains as laminar while the turbulent modelling simulation files are attached to the end of this report. Open the `constant/turbulenceProperties` file and check the following.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType  laminar;
```

```
// ************************************************************************* //
```

- Modify the solution control in `system/fvSolution` dictionary

The next step is to modify the `system/fvSolution` file. The only modification is to remove the reference points of the pressure as the sampling point from `cylBumpInterIbFoam` is located inside the hull. However, by removing the reference points the PIMPLE algorithm uses an arbitrary point, i.e first cell of the domain, as the reference point. Open the `system/fvSolution` file and check the following.

```
 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    pcorr
    {
        solver          CG;
        preconditioner   Cholesky;

        minIter         0;
        maxIter         1000;
        tolerance       1e-08;
        relTol          0.01;
    }

    pd
    {
        solver          CG;
        preconditioner   Cholesky;

        minIter         0;
        maxIter         1000;
        tolerance       1e-08;
        relTol          0.01;
    }

    pdFinal
    {
        solver          CG;
        preconditioner   Cholesky;

        minIter         0;
        maxIter         1000;
        tolerance       1e-08;
        relTol          0.0;
    }

    U
    {
        solver          BiCGStab;
        preconditioner   ILU0;

        minIter         0;
        maxIter          100;
        tolerance        1e-08;
        relTol           0;
    }

    alpha1
    {
        solver           BiCGStab;
        preconditioner   ILU0;

        minIter          0;
        maxIter          100;
        tolerance        1e-08;
        relTol           0;
    }
}

PISO
{
    cAlpha          1;
}

PIMPLE
{
    nOuterCorrectors 2;
    nCorrectors     4;
    nNonOrthogonalCorrectors 0;


    limitMagU      20;
}

relaxationFactors
{
    equations
    {
        U         0.9;
    }
    fields
    {
        pd        0.8;
    }
}
```

In the `interIbFoam` solver, four header files are included which are related to the immersed boundary method. In the following you can find the `interIbFoam.C` file contents.

```
 \*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "interfaceProperties.H"
#include "twoPhaseMixture.H"
#include "turbulenceModel.H"
```

```
#include "immersedBoundaryFvPatch.H"
#include "immersedBoundaryAdjustPhi.H"

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

int main(int argc, char *argv[])
{
#   include "setRootCase.H"
#   include "createTime.H"
#   include "createMesh.H"
#   include "readGravitationalAcceleration.H"
#   include "readPIMPLEControls.H"
#   include "immersedBoundaryInitContinuityErrs.H"
#   include "createFields.H"
#   include "readTimeControls.H"
#   include "correctPhi.H"
#   include "CourantNo.H"
#   include "setInitialDeltaT.H"

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

    Info<< "\nStarting time loop\n" << endl;

    while (runTime.run())
    {
#       include "readPIMPLEControls.H"
#       include "readTimeControls.H"
#       include "immersedBoundaryCourantNo.H"
#       include "setDeltaT.H"

        runTime++;

        Info<< "Time = " << runTime.timeName() << nl << endl;

        // Pressure-velocity corrector
        int oCorr = 0;
        do
        {
            twoPhaseProperties.correct();

#           include "alphaEqn.H"

#           include "UEqn.H"

            // --- PISO loop
            for (int corr = 0; corr < nCorr; corr++)
            {
#               include "pEqn.H"
            }

#           include "immersedBoundaryContinuityErrs.H"

#           include "limitU.H"
```

```
            // Recalculate the mass fluxes
            rhoPhi = phi*fvc::interpolate(rho);

            p = pd + cellIbMask*rho*gh;

            if (pd.needReference())
            {
                p += dimensionedScalar
                (
                    "p",
                    p.dimensions(),
                    pRefValue - getRefCellValue(p, pdRefCell)
                );
            }

            turbulence->correct();
        } while (++oCorr < nOuterCorr);

        runTime.write();

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }

    Info<< "End\n" << endl;

    return 0;
}
```

The included headers are the ”immersedBoundaryFvPatch.H”, the ”immersedBoundaryAdjust-Phi.H”, the ”immersedBoundaryInitContinuityErrs.H” and the ”immersedBoundaryCourantNo.H”. The ”immersedBoundaryFvPatch.H” has the following contents and it is already discussed in sections 1.3 and 1.4. It should be noted that the returned values at the last part of this code will be used for discussed calculation procedure in section 1.3.

```
 class immersedBoundaryFvPatch
:
    public fvPatch
{
    // Private data

        //- Reference to processor patch
        const immersedBoundaryPolyPatch& ibPolyPatch_;

        //- Finite volume mesh reference
        const fvMesh& mesh_;
        .
        .
        .
        // Member Functions
```

```
.
.
.
// Immersed boundary data access

    //- Get fluid cells indicator, marking only live fluid cells
    const volScalarField& gamma() const;
    .
    .
    .
     //- Return list of fluid cells next to immersed boundary (IB cells)
    const labelList& ibCells() const;
    //- Return list of faces for which one neighbour is an IB cell
    //  and another neighbour is a live fluid cell (IB faces)
    const labelList& ibFaces() const;
    .
    .
    .
    //- Return IB points
    const vectorField& ibPoints() const;
    .
    .
    .
    //- Return IB cell extended stencil
    const labelListList& ibCellCells() const;
    .
    .
    .
    //- Return dead cells
    const labelList& deadCells() const;

    //- Return extended dead cells
    const labelList& deadCellsExt() const;

    //- Return dead faces
    const labelList& deadFaces() const;

    //- Return live cells
    const labelList& liveCells() const;

    //- Return immersed boundary cell sizes
    const scalarField& ibCellSizes() const;

    //- Get inverse Dirichlet interpolation matrix
    const PtrList<scalarRectangularMatrix>&
    invDirichletMatrices() const;

    //- Get inverse Neumann interpolation matrix
    const PtrList<scalarRectangularMatrix>&
    invNeumannMatrices() const;
```

The ”immersedBoundaryInitContinuityErrs.H” is for declaration and initialisation of cumulative continuity error. The ”immersedBoundaryCourantNo.H” calculates and outputs the mean and maximum Courant numbers in a IB-sensitive manner.

```
scalar CoNum = 0.0;
scalar meanCoNum = 0.0;
scalar velMag = 0.0;

if (mesh.nInternalFaces())
{
    surfaceScalarField magPhi = mag(faceIbMask*phi);

    surfaceScalarField SfUfbyDelta =
        mesh.surfaceInterpolation::deltaCoeffs()*magPhi;

    CoNum = max(SfUfbyDelta/mesh.magSf())
        .value()*runTime.deltaT().value();

    meanCoNum = (sum(SfUfbyDelta)/sum(mesh.magSf()))
        .value()*runTime.deltaT().value();

    velMag = max(magPhi/mesh.magSf()).value();
}
```

The ”immersedBoundaryAdjustPhi.H” adjust the immersed boundary fluxes to obey continuity. If the mesh is moving, adjustment needs to be calculated on relative fluxes. The ”immersedBoundaryAdjustPhi.C” is given by

```
    ...
    if (mesh.moving())
    {
        fvc::makeRelative(phi, U);
    }

    forAll (phi.boundaryField(), patchI)
    {
        const fvPatchVectorField& Up = U.boundaryField()[patchI];

        if (isA<immersedBoundaryFvPatchVectorField>(Up))
        {
            if (Up.fixesValue())
            {
                // Found immersed boundary path which fixes value.
                // Correction is necessary
                ...
```

If the `fixesValue` set to ”yes” is each variable, then in each iteration the fluxes of that variable are adjusted by calling the ”immersedBoundaryAdjustPhi.C”. So the values on the boundary are fixed.

The `system/fvSchemes` file remains untouched since the aim in this tutorial is to use the available solver and solution procedure for the new case. Till now the implementations were same for both test cases. The next step is the introduction of different boundary conditions at the inlet which requires different case setup work flows.

2.4 Constant Velocity Case

The first case is to introduce a uniform velocity at the inlet facing the hull while the ship is fixed in the domain. This condition represents a condition where the ship is moving with a constant speed in the calm water. It should be noted that the air above the water surface is also is influenced by introducing the uniform inlet. However, uniform inlet for air has negligible effects on the results. In the system directory the `system/setFieldsDict` should change to the following which means that the ship with the height of 0.572m experience a ship water depth of 0.2m in the initial condition. Open the `system/setFieldsDict` file and change as follows.

- Modify `system/setFieldsDict` dictionary to specify the water field

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
    volScalarFieldValue alpha1 0
);

regions
(
    boxToCell
    {
        box (-100 -100 -100) (100 100 0.2); //water depth
        fieldValues
        (
            volScalarFieldValue alpha1 1
        );
    }
);

// ************************************************************************* //
```

Due to the very fine mesh the `deltaT` is considered as low as possible to prevent divergence of the simulation. However, for each simulation there are some limitations that constraints the user. Time is one of them which can be interpreted as cost. The finer the mesh the smaller the time steps and the simulation time would be longer though results become more accurate. In this tutorial the `system/controlDict` is modified to the following to have reasonable results with an efficient simulation for the aim of this project. However, for more accurate results finer mesh and/or smaller time steps can be investigated. Open the `system/controlDict` file and modify it as follows.

- Modify the time step size in the `system/controlDict` dictionary

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

libs
(
    "liblduSolvers.so"
)

application     interIbFoam;

startFrom       startTime;
```

```
startTime       0;

stopAt          endTime;

endTime         10;

deltaT          0.001; //could be smaller to get more accurate results

writeControl    adjustableRunTime;
writeInterval   0.05;   //0.05;

// writeControl    timeStep;
// writeInterval   1;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression compressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable yes;

adjustTimeStep  yes;

maxCo           0.5; //could be smaller

// libs ( "libimmersedBoundary.so");
```

Now it's time to specify the boundary conditions. The following modifications have been done for alpha1, U and pd files. The description of the settings for the immersed boundary is mentioned in the code for the alpha1 and it applies also for U and pd.

- Modify the boundary fields of alpha1, pressure and velocity by adding the new `immersedBoundaryFvPatchField`

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions [0 0 0 0 0 0 0];

internalField uniform 0;

boundaryField
{
    hull
    {
        type immersedBoundary;   //The type should be considered as immersed boundary
        refValue uniform 0;    //The reference value is considered as zero.
        refGradient  uniform 0;
        fixesValue no;     // If this set to "yes" for moving mesh, then in each
                 //iteration the fluxes of this variable are adjusted by calling
                 //the the "immersedBoundaryAdjustPhi.C" to fix the values on the boundary.

        setDeadCellValue   yes;  //Should the dead cells have a value or not?
        deadCellValue      0;   //The value of the dead cells if they suppose to get values.

        value uniform 0;
    }

    inlet
    {
        type            fixedValue; //the value of alpha at inlet is assumed as zero
        value           $internalField;
    }

    outlet
    {
        type zeroGradient;
    }

    atmosphere
    {
        type inletOutlet; //to put a boundary condition for open atmosphere
        inletValue uniform 0;
        value uniform 0;
    }

    bottom
    {
        type zeroGradient;
    }

    sides
    {
        type zeroGradient;
    }
}


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

gedit 0_org/U

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

internalField   uniform (-2 0 0);

boundaryField
{
    hull
    {
        type immersedBoundary;
        refValue uniform (0 0 0);
        refGradient  uniform (0 0 0);
        fixesValue yes;

        setDeadCellValue   yes;
        deadCellValue      (0 0 0);
    }
    inlet
    {
        type fixedValue;
        value uniform (-2 0 0); //speed of 2m/s towards the ship according to the coordinate system
    }
    outlet
    {
        type zeroGradient;
    }
    atmosphere
    {
        type            pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }
    bottom
    {
        type fixedValue;
        value uniform (0 0 0);
    }
    sides
    {
        type zeroGradient;
    }
}


// ************************************************************************* //
```

```
gedit 0_org/pd
```

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

internalField uniform 0;

boundaryField
{
    hull
    {
        type immersedBoundary;
        refValue uniform 0;
        refGradient  uniform 0;
        fixesValue no;

        setDeadCellValue   yes;
        deadCellValue      0;

        value uniform 0;
    }
    inlet
    {
        type zeroGradient;
    }
    outlet
    {
        type zeroGradient;
    }
    atmosphere
    {
        type            totalPressure;
        p0              uniform 0;
        U               U;
        phi             phi;
        rho             rho;
        psi             none;
        gamma           1;
        value           uniform 0;
    }
    bottom
    {
        type zeroGradient;
    }
    sides
    {
        type zeroGradient;
    }
}
```

The last step is to run the case by running the following `./Allrun` script. It can be seen that the procedure of running is a bit more tricky in this tutorial. It means that the `./Allrun` script would change according to the usage of immersed boundary mesh refinement.

- Modify the `./Allrun` and `./Allclean` scripts and run the case with `./Allrun`

```
#!/bin/sh
# Source tutorial run functions
. $WM_PROJECT_DIR/bin/tools/RunFunctions

# Get application name
application="interIbFoam"


# mkdir constant/polyMesh   #<--- Uncomment in case of using refineImmersedBoundaryMesh utility
                            # it is used in case of running for the second or third,... time due
                            # to the remove command in the following...
# cp -f save/blockMeshDict constant/polyMesh  #<--- Uncomment in case of using
                                              # refineImmersedBoundaryMesh utility


runApplication blockMesh
\cp -f save/boundary constant/polyMesh/    #<--- As it can be seen the saved boundary
                                           # file here replaced with the new one
# runApplication refineImmersedBoundaryMesh -ibCellCellFaces #(-ibCells, -ibCellCells also could be
                                                          # used dependind on the level of refinement)
                          #^--- Uncomment in case of using refineImmersedBoundaryMesh utility


#\rm -rf constant/polyMesh  #<--- Uncomment in case of using refineImmersedBoundaryMesh utility
#\mv 0/polyMesh constant/polyMesh/#<--- Uncomment in case of using refineImmersedBoundaryMesh utility
#\rm -rf 0                  #<--- Uncomment in case of using refineImmersedBoundaryMesh utility

\mkdir 0
\cp 0_org/* 0/

runApplication setFields

runApplication $application
```

In case of cleaning the case the `./Allclean` file is made as:

```
#!/bin/sh

# Source tutorial clean functions
. $WM_PROJECT_DIR/bin/tools/CleanFunctions

\rm -f constant/polyMesh/boundary

cleanCase
\rm -rf 0
```

Finally, running the case by typing the following command in terminal window.

```
./Allrun
```

- Post process the case using `paraview`

```
paraFoam -nativeReader
```

The results will be shortly discussed in the post-processing section 3.1. It should be noted that till now the constant velocity case is implemented. The next step is implementation of the wave propagated case.

2.5 Wave Propagated Case

Notice: In order to implement the wave propagated case, the initial steps should be done again, since the initial steps are common between both cases! Hence, please start by chapter 2.3 and skip the chapter 2.4 and continue with chapter 2.5 for the wave propagated case setup.

In this case set-up the major difference is the introduction of a wave at the inlet. This represents a moving ship towards waves. Most of the case setup steps are similar as before while the inlet boundary condition should change. In order to generate a wave at boundary there are some utilities available in OpenFOAM, e.g. `waves2Foam`, `groovyBC` etc. However, not all of them are applicable to every case and condition. For instance, the `waves2Foam` utility is not applicable to FOAM-extend-4.0. Hence, for wave generation at the inlet, the `swak4Foam` library is used in which the functionality of two utilities `groovyBC` and `funkySetFields` are combined.

2.5.1 swak4Foam Library

The `swak4Foam` library offers the user the possibility of introducing different expressions for boundaries, containing the fields and evaluates them without programming. It means that for the case under study the boundary at the inlet can be modified with introducing wave expressions to simulate the wave in it. First, the installation procedure will be explained, then the required preparation will be discussed and finally the `groovyBC` and `funkySetFields` utilities will be explained.

- Download, install and utilize the `swak4Foam` library

Installation Procedure

In order to install the `swak4Foam` library in FOAM-extend-4.0, its development version can be downloaded according to the following commands. In case of any problem during downloading or for downloading other versions of it please check www.openfoamwiki.net website [5].

Open a new terminal window and download the `swak4Foam` from the main Mercurial development repository, using the hg command. Then checkout the right branch.

```
f40NR
run
hg clone http://hg.code.sf.net/p/openfoam-extend/swak4Foam swak4Foam
cd swak4Foam && hg update develop
```

Then install the `swak4Foam` according to the following. The first `./Allwmake` command takes around 40 mins while the second one is for getting a summary of the installation and i runs much quicker. More detail is given in the www.openfoamwiki.net website[6].

```
./maintainanceScripts/compileRequirements.sh
./Allwmake > log.make 2>&1
./Allwmake > log.make 2>&1
```

Then you can check the version by typing the command `funkySetFields`. This results in an error message, but just after the usual OpenFOAM-banner the version of swak4Foam and the release date would be published.

```
funkySetFields
```

ControlDict Preparation

Please add these lines to the case system directory and inside the `controlDict` near the top.

- Include the new library to the `system/controlDict` dictionary

```
libs (
      "liblduSolvers.so"
      //"libOpenFOAM.so"
      "libsimpleSwakFunctionObjects.so"
      "libswakFunctionObjects.so"
      "libgroovyBC.so"
     );
```

Contents

The `swak4Foam` library consist of the following libraries and utilities given in www.openfoamwiki.net website [7]. For more information take a look at README file of the installed `swak4Foam` library.

Libraries

- `swak4FoamParsers`: to access the OpenFOAM data-structures
- `groovyBC`: A boundary condition that allows arbitrary expressions in the field-file
- `swakFunctionObjects`: for manipulating and creating fields with expressions
- `simpleSwakFunctionObjects`: Evaluate expressions and output the results
- `swakSourceFields`: used as source-term or coefficient in some solver
- `swakTopoSources`: `topoSources` for `cellSet` and `faceSet`
- `pythonIntegration`: allows inclusion of Python code directly into an OpenFOAM Simulation run.

Utilities

- `funkySetFields`: Utility that allows creation and manipulation of files with expressions
- `funkySetBoundaryField`: Sets any field on a boundary to a non-uniform value based on an expression
- `replayTransientBC`: Utility to quickly test whether a `groovyBC` gives the expected results

In order to propagate a wave at the inlet the `groovyBC` library and the `funkySetFields` utility are introduced to the case.

2.5.2 groovyBC

This library allows the user to introduce a mixed-BC with values, gradients and valueFractions as expressions instead of fields. The prerequisite of using `groovyBC` is installed `bison` with version of 2.4.1 or older! Check your `bison` version with "bison -V" command. The addition of this library is already done in the "controlDict Preparation" paragraph of section 2.5.1. The parameters are defined as bellow:

- `valueExpression`: String with the value to be used if a Dirichlet-condition is needed. Defaults to zero
- `value`: is used if no "`valueExpression`" is given. value is also used for the first timestep/iteration if "`valueExpression`" is specified. If "`valueExpression`" is specified without setting "value", 0 is taken for the first timestep/iteration. (might cause a Floating Point Exception)
- `gradientExpression`: String with the gradient to be used if a Neumann condition is needed. Defaults to zero
- `fractionExpression` Determines whether the face is Dirichlet (1) or Neumann (0). Defaults to 1
- `variables`: List with temporary variables separated by a semicolon. May make the writing of expressions shorter. Defaults to empty. Names defined here "shadow" fields of the same name

- `timelines` List with sub-dictionaries that specify interpolation tables over time. See the original timeVaryingUniform-condition. Currently only scalars are allowed. The parameter name specifies the name under which this may be accessed. The name "shadows" fields of the same name

The expression syntax including the C++ operators, pseudo-variables and pseudo-functions are from a source file for bison. The basic ones can be found here[8]. An available example of the `groovyBC` usage is called "`groovyWaveTank` which is a 2D case for generating "2nd-order Stokes waves" in a tank designed for "`interFoam`" solver in OpenFOAM. The test case is available in www.openfoamwiki.net website [9]. As an illustration one screenshot of the case is presented, see Figure 2.5.

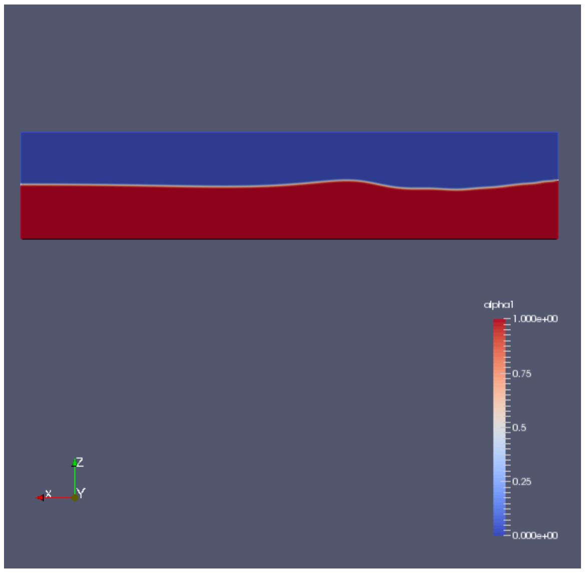

Figure 2.5: groovyWaveTank User Case

2nd-order stokes waves

The same wave is used in the case setup and due to that the `0_org/alpha1` is modified to the following.

- Implement boundary conditions of alpha1, pressure and velocity with the `groovyBC` library

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions [0 0 0 0 0 0 0];

internalField uniform 0;

boundaryField
{
    hull
    {
        type immersedBoundary;
        refValue uniform 0;
        refGradient  uniform 0;
        fixesValue no;
        setDeadCellValue   yes;
        deadCellValue      0;

        value uniform 0;
    }

    inlet
    {
       type            groovyBC;
       valueExpression "(pos().z<=A*cos(-w*time())+0.5*k*A*A*cos(2*(-w*time()))) ? 1 : 0";
       //The wave formula is implemented here with wave length of 5 and amplitude of 0.3

       variables       "l=5;A=0.3;G=vector(0,0,-9.81);k=2*pi/l;w=sqrt(k*mag(G));";
       timelines       ();
    }

    outlet
    {
        type zeroGradient;
    }

    atmosphere
    {
        type inletOutlet;
        inletValue uniform 0;
        value uniform 0;
    }

    bottom
    {
        type zeroGradient;
    }

    sides
    {
        type zeroGradient;
    }
}


// ************************************************************************* //
```

Open the 0_org/pd file and modify as follows.

```
 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField uniform 0;

boundaryField
{
    hull
    {
        type immersedBoundary;
        refValue uniform 0;
        refGradient  uniform 0;
        fixesValue no;

        setDeadCellValue   yes;
        deadCellValue      0;

        value uniform 0;
    }
    inlet
    {
        type            buoyantPressure;
        value           uniform 0;
    }
    outlet
    {
        type            buoyantPressure;
        value           uniform 0;
    }
    atmosphere
    {
        type            totalPressure;
        p0              uniform 0;
        U               U;
        phi             phi;
        rho             rho;
        psi             none;
        gamma           1;
        value           uniform 0;
    }
    bottom
    {
        type            buoyantPressure;
        value           uniform 0;
    }
    sides
    {
        type            buoyantPressure;
        value           uniform 0;
    }
}


// ************************************************************************* //
```

Open the `0_org/U` file and modify it as follows.

```
 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform (0 0 0);
```

```
boundaryField
{
    hull
    {
        type immersedBoundary;
        refValue uniform (0 0 0);
        refGradient  uniform (0 0 0);
        fixesValue yes;

        setDeadCellValue   yes;
        deadCellValue      (0 0 0);
    }
    inlet
    {
       type            groovyBC; //2nd order stokes wave with length of 5 and amplitude of 0.3
       valueExpression "(pos().z<=A*cos(-w*time())+0.5*k*A*A*cos(2*(-w*time()))) ?\
       vector( A*w*exp(k*pos().z)*cos(-w*time()), 0,\
       A*w*exp(k*pos().z)*sin(-w*time())) : vector(0,0,0)";
       variables       "l=5;A=0.3;G=vector(0,0,-9.81);k=2*pi/l;w=sqrt(k*mag(G));";
       timelines       ();
    }
    outlet
    {
        type zeroGradient;
    }
    atmosphere
    {
        type            pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }

    {
        type            slip;
        value           uniform (0 0 0);
    }
    sides
    {
        type zeroGradient;
    }
}


// ************************************************************************* //
```

2.5.3 funkySetFields

The funkySetFields utility sets the value of a scalar or a vector field depending on an expression and gives the possibility to set the value of fields on selected patches. It is similar to the `setFields` utility.

In the dictionary `funkySetFieldsDict` a list of dictionaries named expressions is read and one dictionary is evaluated after another. More information can be found in www.openfoamwiki.net website [10]. In each dictionary there can be the following entries:

- field: the target field

- expression: the expression to write to the field
- condition: select a subset of the cells (this is optional)
- keepPatches: see command line options (optional)
- create: see command line options (optional)
- valuePatches: see command line options (optional)
- dimension: see command line options (optional)

In the case of wave generation in the ongoing tutorial the file `system/funkySetFieldsDict` should be added to the system directory.

- Make a `funkySetFieldsDict` dictionary in order to use the `funkySetFields` utility

Open a new file and name it `funkySetFieldsDict` in the `system` directory and add the following in that file.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

expressions
 (
setWave
{
 field alpha1; //field to initialise
 expression "1";
 condition  "(-100<=pos().z) && (pos().z<=0.2) && (-100<=pos().y) &&\
 (pos().y<=100) && (-100<=pos().x) && (pos().x<=100)";
        //keepPatches true; //keep the boundary conditions that were set before
}
 );

// ************************************************************************* //
```

The last step is to modify the `./Allrun` script. Remember to have mesh refinement around the immersed boundary (hull) the related modifications discussed in the constant velocity case should be attached to the following one.

- Modify the `./Allrun` and `./Allclean` scripts and run the case with `./Allrun`

```
gedit Allrun

#!/bin/sh
# Source tutorial run functions
. $WM_PROJECT_DIR/bin/tools/RunFunctions

# Get application name
application="interIbFoam"


# mkdir constant/polyMesh   #<--- Uncomment in case of using refineImmersedBoundaryMesh utility
                            # it is used in case of running for the second or third,... time due
                            # to the remove command in the following...
# cp -f save/blockMeshDict constant/polyMesh  #<--- Uncomment in case of using
```

```
                                                # refineImmersedBoundaryMesh utility


runApplication blockMesh
\cp -f save/boundary constant/polyMesh/    #<--- As it can be seen the saved boundary
                                           # file here replaced with the new one

# runApplication refineImmersedBoundaryMesh -ibCellCellFaces #(-ibCells, -ibCellCells also could be
                                                         # used depends on the level of refinement)
                          #^--- Uncomment in case of using refineImmersedBoundaryMesh utility


#\rm -rf constant/polyMesh  #<--- Uncomment in case of using refineImmersedBoundaryMesh utility
#\mv 0/polyMesh constant/polyMesh/#<--- Uncomment in case of using refineImmersedBoundaryMesh utility
#\rm -rf 0                  #<--- Uncomment in case of using refineImmersedBoundaryMesh utility

\mkdir 0
\cp 0_org/* 0/

runApplication funkySetFields -time 0 #it should be specified

runApplication $application
```

Run the case by

```
./Allrun
```

During the simulations if you get an error regarding the ”libOpenFOAM.so”, please comment this in the `controlDict` added new lines.

Same as constant velocity the next steps is completed and the results are presented in the section 3.2.

- Post process the case using `paraview`

```
paraFoam -nativeReader
```

Chapter 3

Results and Discussion

The results are post processed in paraview. Figure 3.1 shows the fluid domain at the initial condition. The `iso-surface` of alpha1=0.5 will show the free surface. It is necessary to use the `-nativeReader` flag for the `paraFoam` command in foam-extend.

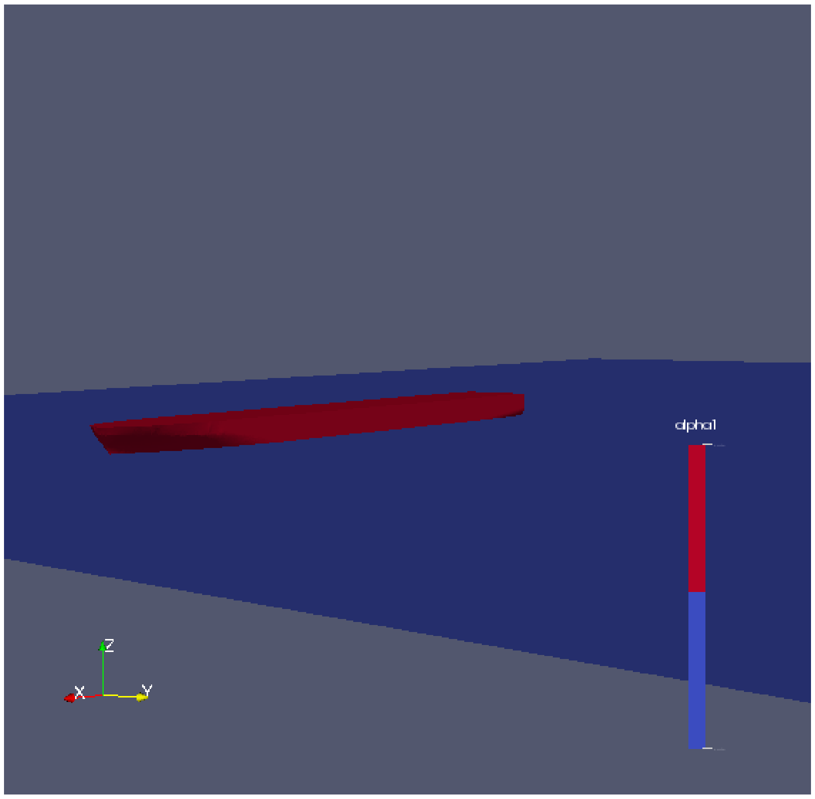

Figure 3.1: Free surface in initial condition

3.1 Constant Velocity Case

The inlet velocity of 2 m/s was applied at the inlet. Figures 3.2 and 3.3 show the generated waves in all of the domain after 90 and 120 time steps. Figure 3.4 shows half of the ship hull and illustrates the flow around the hull. For a ship with constant speed the pressure would be higher in fore and aft part of the ship and just after front part, the shoulder would experience a low pressure condition.

Figure 3.5 is captured from the lower position to show how big are the waves in different parts. For both cases if the mesh becomes finer the accuracy of the results would be higher.

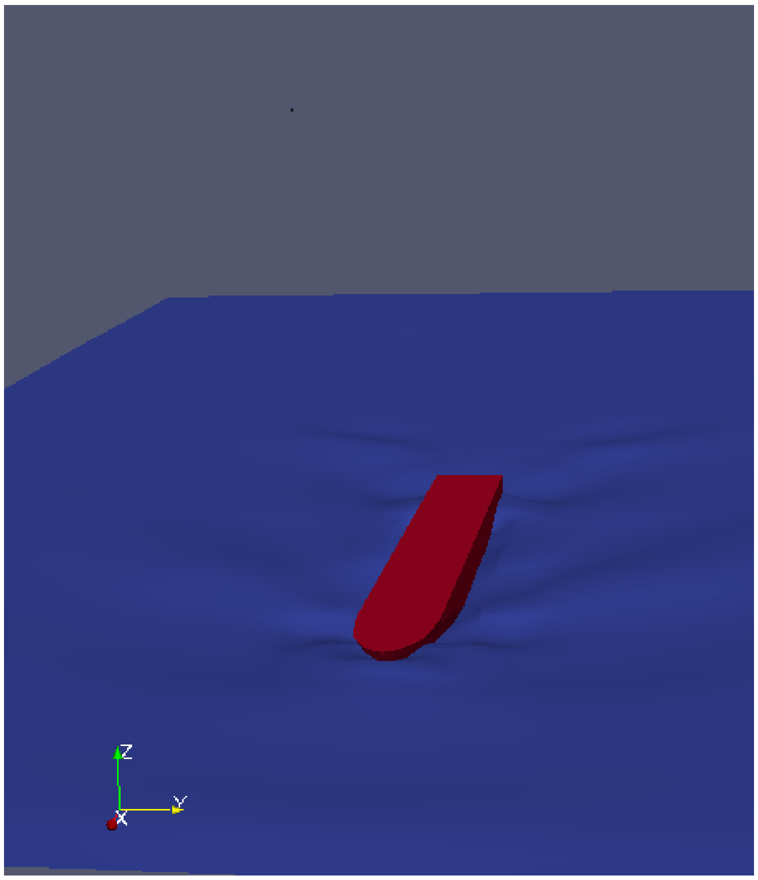

Figure 3.2: Time step 90

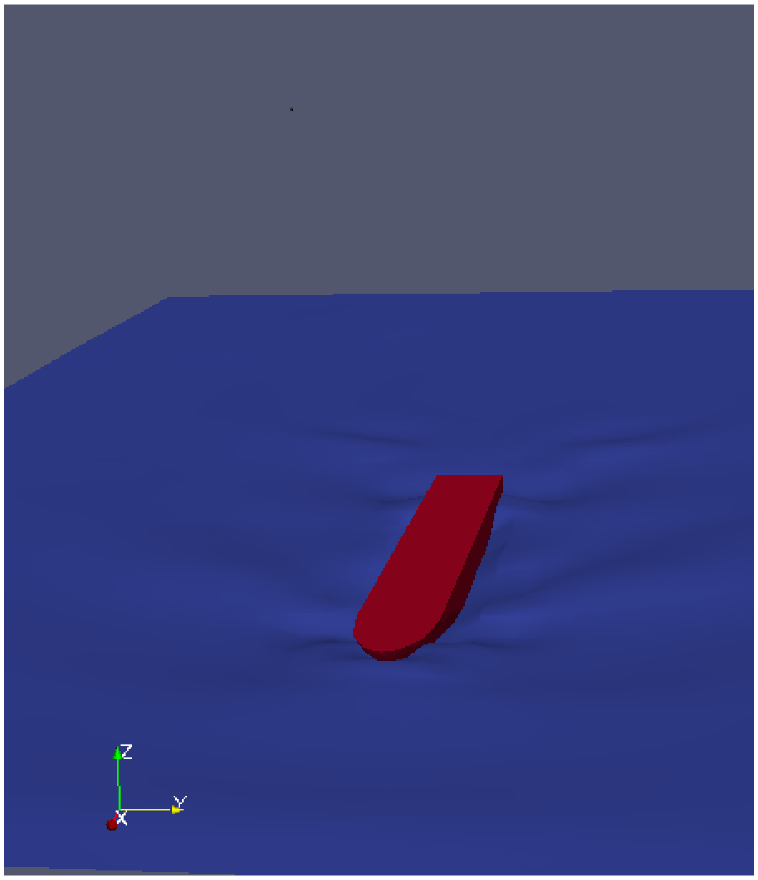

Figure 3.3: Time step 120

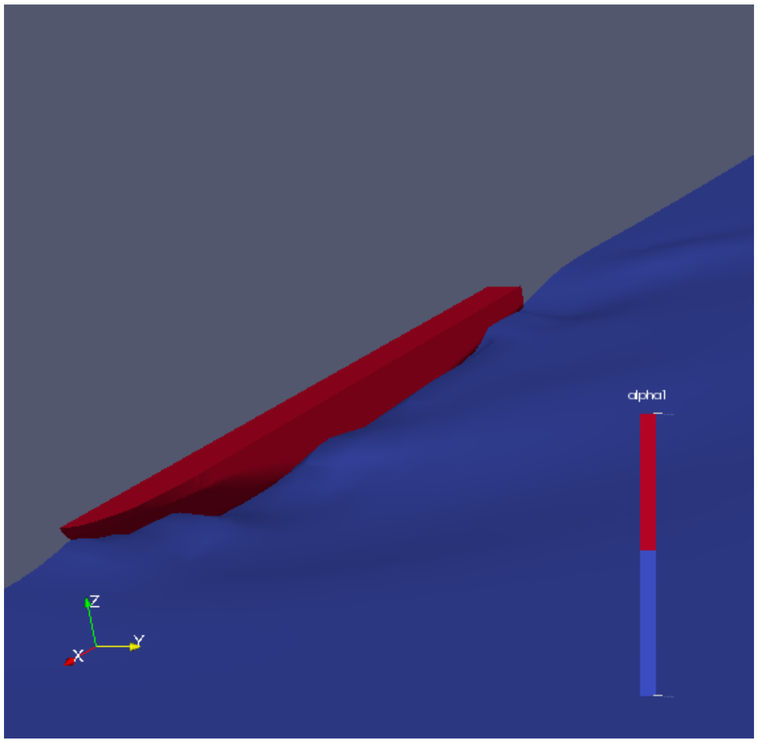

Figure 3.4: Flow around the half ship

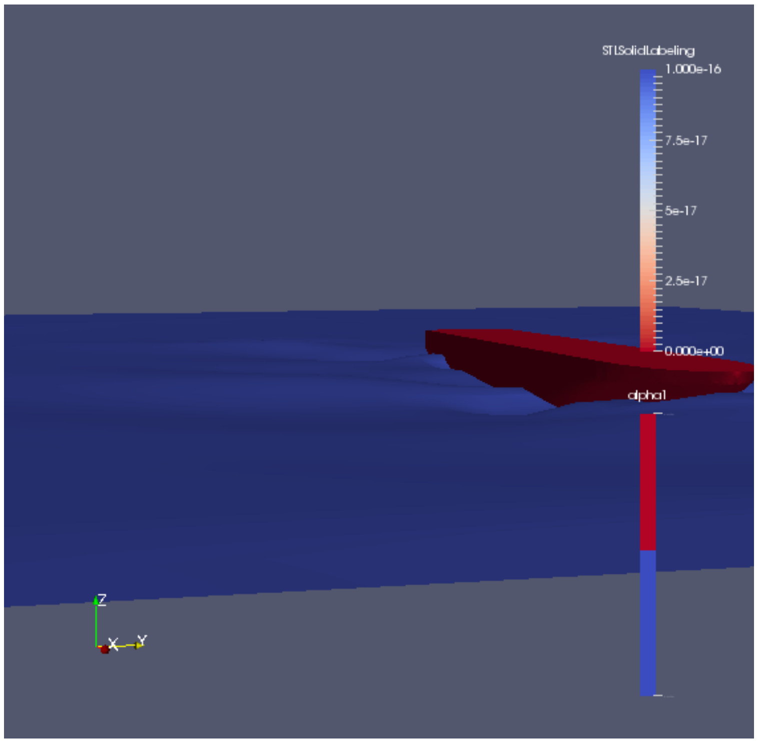

Figure 3.5: water surface from a lower view

3.2 Wave Propagated Case

The propagated wave of 2nd-order stokes with l=5 and A=0.3 at the inlet can be seen in Figures 3.6 to 3.8. Figure 3.6 shows how the wave hits the fixed body in a general way. Figure 3.7 is captured when the wave peak reaches the hull. Figure 3.8 shows the conditions when the wave trough reaches the front part of the ship. The wave pattern behind the ship is small for the introduced condition.

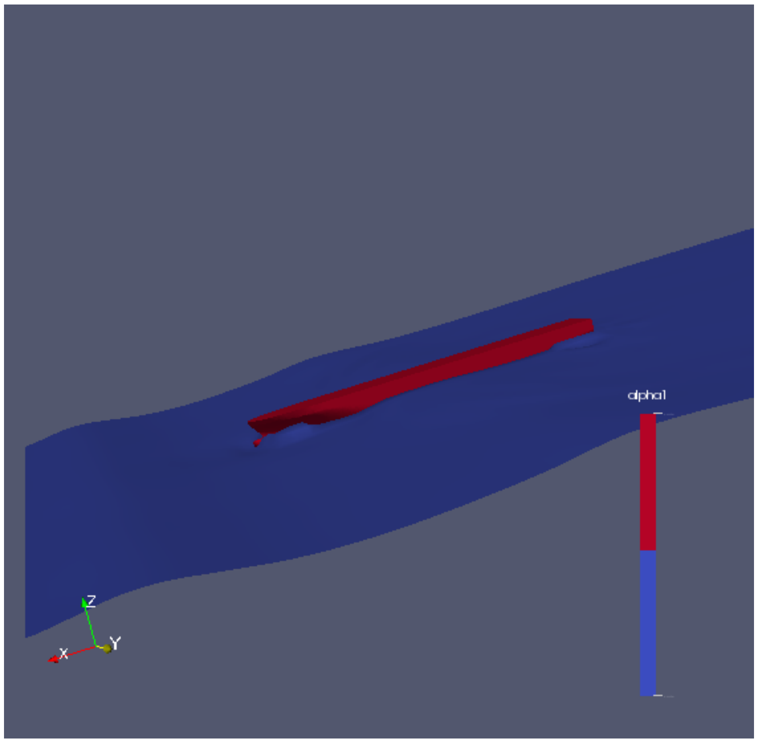

Figure 3.6: General wave introduced water surface condition

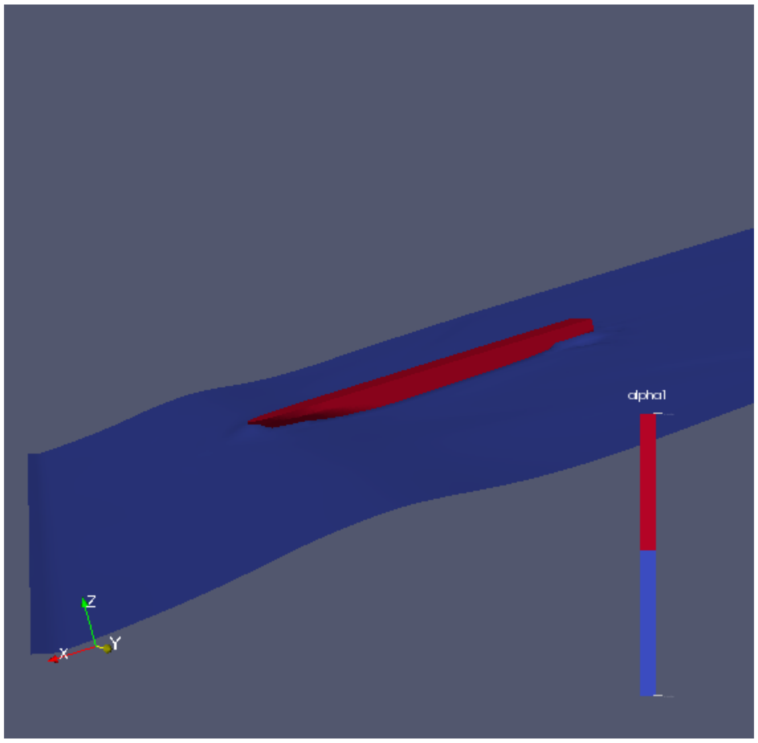

Figure 3.7: wave peak reaches the front part

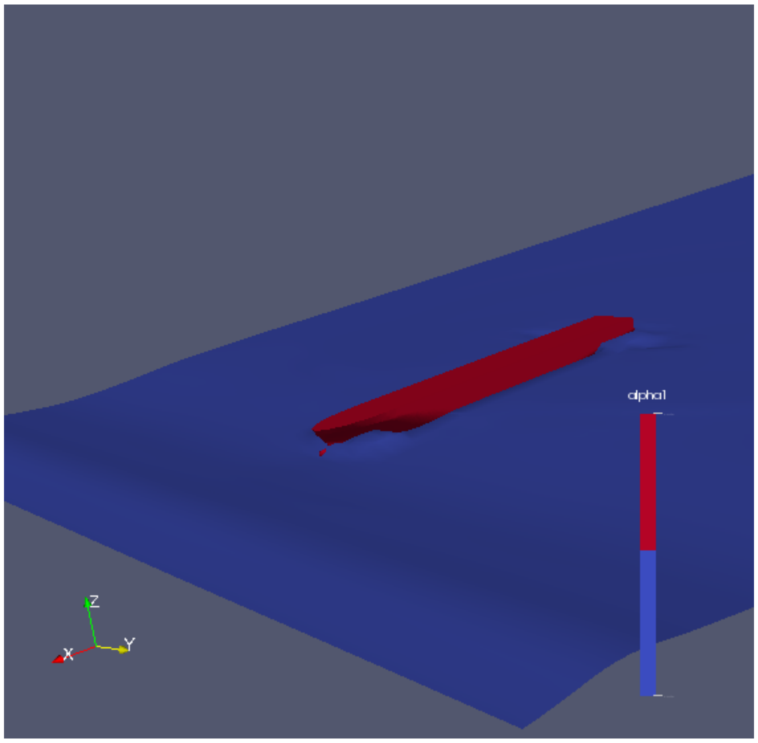

Figure 3.8: wave trough reaches the front part

3.3 Future Work

In order to further expand the presented project the following future works are suggested.

- As it was mentioned before, due to some practical difficulties, the tutorial is concerned with laminar flow simulation. One of the possible sources of the problem with turbulent flow simulation is the coarse mesh. The mesh refinement was applied and the error was not solved. Therefore, it can be as a future work to resolve the problem with turbulent flow simulation by further refining the mesh to see if this is the only source of problem. The applied files are attached as an appendix to the project.

- The ship is assumed as fixed during the simulation while it can be treated as a moving boundary. The `interDyMFoam` solver available in `pitchingPlate` tutorial can be used to simulate such conditions.

- The `swak4Foam` library contains a utility called "`forceSectional`" which can be used to sum up the forces on the body for further post processing.

- Finally, the new boundary conditions can be applied in the `groovyBC` and different conditions can be studied due to 6DOF of the hull in reality.

Study questions

1. Why IBM over boundary fitted grids?
2. How do you generate mesh in IBM?
3. What is the purpose of mesh refinement near the immersed body?

Bibliography

[1] C. S. Peskin. Numerical analysis of blood flow in the heart. *J. Comput Phys.*, 25:220–252, 1977.

[2] http://www.tfd.chalmers.se/∼ hani/kurser/OS_CFD_2015/HrvojeJasak/ImmersedBoundary.pdf.

[3] Reinout vander Meulen. The immersed boundary method for the (2d) incompressible navier-stokes equations. Master's thesis, Delft University, 2006.

[4] J. Favier M. Meldi P. Meliga E. Serre E. Constant, C. Li. Implementation of a discrete immersed boundary method in openfoam. *Computers anf Fluids*, 2016.

[5] http://openfoamwiki.net/index.php/Installation/swak4Foam/Downloading.

[6] http://openfoamwiki.net/index.php/Installation/swak4Foam/Installing_On/Ubuntu.

[7] http://openfoamwiki.net/index.php/Contrib/swak4Foam.

[8] https://openfoamwiki.net/index.php/Contrib_groovyBC.

[9] https://openfoamwiki.net/index.php/Contrib_groovyBC.

[10] http://openfoamwiki.net/index.php/Contrib_funkySetFields.

Appendices

The implementation of the turbulent modelling has been done with the following basic setups. Here are the modified ones and other files remain unchanged from the tutorial implementations.

```
.
    0_org
     --- alpha1
     ---epsilon
     ---k
     ---nut
     ---pd
     ---U
    Allclean
    Allrun
    constant
     --- g
     --- polyMesh
           --- blockMeshDict
     --- RASProperties
     --- transportProperties
     --- triSurface
           --- hull.ftr
           --- hull.stl
     --- turbulenceProperties
    save
     --- blockMeshDict
     --- boundary
    system
     --- controlDict
     --- decomposeParDict
     --- fvSchemes
     --- fvSolution
     --- mapFieldsDict
     --- setFieldsDict
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | foam-extend: Open Source CFD                    |
|  \\    /   O peration     | Version:     4.0                                |
|   \\  /    A nd           | Web:         http://www.foam-extend.org         |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      alpha1;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions [0 0 0 0 0 0 0];

internalField uniform 0;

boundaryField
```

```
{
    hull
    {
        type immersedBoundary;
        refValue uniform 0;
        refGradient  uniform 0;
        fixesValue no;

        setDeadCellValue   yes;
        deadCellValue      0;

        value uniform 0;
    }

    inlet
    {
        type zeroGradient;
    }

    outlet
    {
        type zeroGradient;
    }

    atmosphere
    {
        type inletOutlet;
        inletValue uniform 0;
        value uniform 0;
    }

    bottom
    {
        type zeroGradient;
    }

    sides
    {
        type zeroGradient;
    }
}


// ************************************************************************* //
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | foam-extend: Open Source CFD                    |
|  \\    /   O peration     | Version:     4.0                                |
|   \\  /    A nd           | Web:         http://www.foam-extend.org         |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      epsilon;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -3 0 0 0 0];

internalField   uniform 14.855;

boundaryField
{
    hull
    {
        type            immersedBoundaryEpsilonWallFunction;
        patchType       immersedBoundary;
        refValue        uniform 1e-10;
        refGradient     uniform 0;
        fixesValue      false;
        setDeadCellValue yes;
        deadCellValue   1e-10;
        Cmu             0.09;
        kappa           0.41;
        E               9.8;

        value           uniform 14.855;
    }
    atmosphere
    {
        type            zeroGradient;
    }
    inlet
    {
        type            fixedValue;
        value           uniform 14.855;
    }
    outlet
    {
        type            inletOutlet;
        inletValue      uniform 14.855;
        value           uniform 14.855;
    }
    bottom
    {
        type            zeroGradient;
    }
    sides
    {
        type            zeroGradient;
    }
}
```

```
// ************************************************************************* //
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | foam-extend: Open Source CFD                    |
|  \\    /   O peration     | Version:     4.0                                |
|   \\  /    A nd           | Web:         http://www.foam-extend.org         |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      k;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform 0.375;

boundaryField
{
    hull
    {
        type            immersedBoundaryWallFunction;
        patchType       immersedBoundary;
        refValue        uniform 1e-10;
        refGradient     uniform 0;
        fixesValue      false;
        setDeadCellValue yes;
        deadCellValue   1e-10;

        value           uniform 0.375;
    }
    atmosphere
    {
        type            zeroGradient;
    }
    inlet
    {
        type            fixedValue;
        value           uniform 0.375;
    }
    outlet
    {
        type            inletOutlet;
        inletValue      uniform 0.375;
        value           uniform 0.375;
    }
    bottom
    {
        type            zeroGradient;
    }
    sides
    {
        type            zeroGradient;
    }
}


// ************************************************************************* //
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | foam-extend: Open Source CFD                    |
|  \\    /   O peration     | Version:     4.0                                |
|   \\  /    A nd           | Web:         http://www.foam-extend.org         |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      nut;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform 0;

boundaryField
{
    hull
    {
        type            immersedBoundaryWallFunction;
        patchType       immersedBoundary;
        refValue        uniform 0;
        refGradient     uniform 0;
        fixesValue      false;
        setDeadCellValue false;
        deadCellValue   0;
        value           uniform 0;
    }
    atmosphere
    {
        type            calculated;
        value           uniform 0;
    }
    inlet
    {
        type            calculated;
        value           uniform 0;
    }
    outlet
    {
        type            calculated;
        value           uniform 0;
    }
    bottom
    {
        type            calculated;
        value           uniform 0;
    }
    sides
    {
        type            zeroGradient;
    }
}


// ************************************************************************* //
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | foam-extend: Open Source CFD                    |
|  \\    /   O peration     | Version:     4.0                                |
|   \\  /    A nd           | Web:         http://www.foam-extend.org         |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      pd;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField uniform 0;

boundaryField
{
    hull
    {
        type immersedBoundary;
        refValue uniform 0;
        refGradient  uniform 0;
        fixesValue no;

        setDeadCellValue   yes;
        deadCellValue      0;

        value uniform 0;
    }
    inlet
    {
        type zeroGradient;
    }
    outlet
    {
        type zeroGradient;
    }
    atmosphere
    {
        type            totalPressure;
        p0              uniform 0;
        U               U;
        phi             phi;
        rho             rho;
        psi             none;
        gamma           1;
        value           uniform 0;
    }
    bottom
    {
        type zeroGradient;
    }
    sides
    {
        type zeroGradient;
    }
}


// ************************************************************************* //
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | foam-extend: Open Source CFD                    |
|  \\    /   O peration     | Version:     4.0                                |
|   \\  /    A nd           | Web:         http://www.foam-extend.org         |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform (-2 0 0);

boundaryField
{
    hull
    {
        type immersedBoundaryVelocityWallFunction;
        patchType immersedBoundary;
        refValue uniform (0 0 0);
        refGradient  uniform (0 0 0);
        fixesValue yes;

        setDeadCellValue   yes;
        deadCellValue      (0 0 0);

        value uniform (0 0 0);
    }
    inlet
    {
        type fixedValue;
        value uniform (-2 0 0);
    }
    outlet
    {
        type zeroGradient;
    }
    atmosphere
    {
        type            pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }
    bottom
    {
        type fixedValue;
        value uniform (0 0 0);
    }
    sides
    {
        type zeroGradient;
    }
}


// ************************************************************************* //
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | foam-extend: Open Source CFD                    |
|  \\    /   O peration     | Version:     4.0                                |
|   \\  /    A nd           | Web:         http://www.foam-extend.org         |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
simulationType  RASModel;
```

```
// ************************************************************************* //
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | foam-extend: Open Source CFD                    |
|  \\    /   O peration     | Version:     4.0                                |
|   \\  /    A nd           | Web:         http://www.foam-extend.org         |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      RASProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

RASModel            kEpsilon;

turbulence          on;

printCoeffs         on;


// ************************************************************************* //
```