CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Project work:

Application of dynamic meshes to potentialFreeSurfaceFoam to solve for 6DOF floating body motions

Developed for OpenFOAM-2.1.x

Author:
Guilherme Moura Paredes

Peer reviewed by:
Tian Tang
Jelena Andric

November 18, 2012

Contents

Chapter 1

Introduction

1.1 Report description

This report describes the applications of dynamic mesh capabilities to an OpenFOAM solver, **potentialFreeSurfaceFoam**, that is only able to compute simulations with static meshes.

The report is divided in nine chapters: Introduction, Theory of moving meshes, The solver, Including the dynamic mesh, Case set up, Results, Modifications to the work presented, Conclusions and Future work. The work described was performed on OpenFOAM 2.1.x. The operating system used was Ubuntu 12.04 and, as such, all commands described are based on Ubuntu Linux.

In this report, the following conventions are used:

- **Bold** font is used when referring to executable applications;
- `Verbatim` text is used when referring to command line instructions, written code, or code variables and fields;
- *Italic* text is used when quoting text from references.
- SMALL CAPS is used when referring to OpenFOAM boundary condition types.

List of variables

- `p` - field variable describing the total kinematic pressure in the fluid.
- `p_gh` - field variable describing the component of the total kinematic pressure cause by dynamic effects.
- `rho` - field variable describing the fluid density.
- `U` - field variable describing the fluid velocity.
- `zeta` - field variable describing the free surface profile.
- ρ - fluid density.
- ζ - free surface profile.
- Ω - control volume volume.
- $\vec{n}$ - outward pointing unit normal vector.
- $\vec{r}$ - position vector.
- t - time.
- S - control volume surface.

- $\vec{v}$ - fluid velocity.
- $\vec{x}$ - coordinate vector.
- $\vec{v_b}$ - velocity of the control volume boundary.

1.2 Objectives

The main objective of the present work is to modify the original **potentialFreeSurfaceFoam** solver, distributed with OpenFOAM from version 2.1.0, in order to be able to use it with dynamic meshes. The validation of the results of the solver when computing the time history of 6 DOF body motions will not be a part of the work.

1.3 Motivation

The idea behind the application of dynamic mesh capabilities to **potentialFreeSurfaceFoam** is to get a solver that allows the solution of the interaction of six degree of freedom bodies with fluids with a free surface. Solvers like **interDyMFoam** already make this possible. However, such solvers also solve for the dynamics of the air above the free surface, vastly increasing the computational domain and time. In typical engineering calculations involving surface gravity waves, the air pressure is approximated as constant and, therefore, can be disregarded. This is very close to the solution process used by **potentialFreeSurfaceFoam**, with the free surface being modelled as a boundary condition, where the the dynamic pressure boundary condition is the special type waveSurfacePressure. Applying dynamic meshes to **potentialFreeSurfaceFoam** will allow the interaction of six degree of freedom bodies with free surface fluids to be computed in a much faster way. This is specially important in design optimization, when several different designs are tested and the computational time difference between a simplified and complete solver gets more noticeable. These characteristics are closely related to part of the work to be developed in my PhD research: studying of mooring systems for floating point absorber wave energy converters. Moorings or mooring systems are the means by which floating objects are anchored to the sea bottom. The best known example is the simple anchor line of a small boat. Point absorbers are relatively small floating devices (when compared to the typical wave length of sea waves), whose working principles are very diverse, but all aim at converting energy from waves into some usable form, generally electricity. These floating bodies must be moored to the sea bottom to prevent them from drifting in the sea.

Chapter 2

Theory of moving meshes

This section will give a brief overview of the theory of moving meshes. The derivations presented will follow the work of [1].

Mass conservation on a typical engineering flow is expressed by the continuity equation, eq. 2.1:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho \vec{v})}{\partial \vec{x}} = 0 \tag{2.1}$$

where $\vec{v}$ is the flow velocity, $\vec{x}$ is the position vector, ρ is the fluid density and t is time. If eq. 2.1 is integrated in space over a control volume with moving boundaries, the following equation is obtained:

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_{\Omega} \rho \, \mathrm{d}\Omega - \int_{S} \frac{\mathrm{d}\vec{r}}{\mathrm{d}t} \cdot \vec{n} \, \mathrm{d}S + \int_{S} \rho \vec{v} \cdot \vec{n} \, \mathrm{d}S = 0 \tag{2.2}$$

where Ω and S represent, respectively, the volume and the surface of the control volume and $\vec{r}$ the position of the boundaries. The term

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_{\Omega} \rho \, \mathrm{d}\Omega$$

represents the variation of the mass contained in the control volume. The term

$$\int_{S} \rho \vec{v} \cdot \vec{n} \, \mathrm{d}S$$

represents the amount of mass flowing out of the control volume due to the proper flow velocity. Finally, the term

$$\int_{S} \frac{\mathrm{d}\vec{r}}{\mathrm{d}t} \cdot \vec{n} \, \mathrm{d}S$$

represents the mass variation within the control volume due to changes in the control volume limits, both shape and position.

Setting

$$\frac{\mathrm{d}\vec{r}}{\mathrm{d}t} = \vec{v_{\mathrm{b}}} \tag{2.3}$$

where $\vec{v_{\mathrm{b}}}$ is the control volume boundary velocity, we get

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_{\Omega} \rho \, \mathrm{d}\Omega + \int_{S} \rho \left(\vec{v} - \vec{v_{\mathrm{b}}} \right) \cdot \vec{n} \, \mathrm{d}S = 0 \tag{2.4}$$

From equation 2.4 its clear that, when dealing with moving meshes, the velocity of the mesh points, or the relative velocity between the mesh points and the flow, must be taken into account. This equation explains the need to, in 4.3.3, sometimes make the flux relative and sometimes make it absolute.

Chapter 3

The solver

3.1 potentialFreeSurfaceFoam

According to the description provided at the OpenFOAM website,[2], **potentialFreeSurfaceFoam** is "*a single phase, incompressible, Navier-Stokes solver that approximates waves through a wave height field that evolves in time. The solver can reliably predict the behaviour of a free surface where the effects of the low density phase, e.g. air, can be neglected and where waves do not break. Its computational costs is significantly lower than interface-capturing solvers.*". Even though **potentialFreeSurfaceFoam** solves for a fluid having free surface, in the solution algorithm there is no actual free surface. The effect of free surface is simulated through boundary conditions and not through the mapping of different fluid characteristics to the physical domain and solving for the interactions between them. The boundary condition simulating the free surface is WAVESURFACEPRESSURE, described in section 3.2. The problem is solved in a static grid and the free surface profile is only known by the values of the vectorField `zeta` at the free surface patch.

The reference to "potential" in the name "potentialFreeSurfaceFoam" is due to the waves at the free surface being approximated by a wave height potential. In the interior domain, the solver is capable of handling different types of turbulence models.

A typical result of the **potentialFreeSurfaceFoam** solver is displayed in figures 3.1 and 3.2, taken from the "oscillatingBox" tutorial provided in OpenFOAM 2.1.x for **potentialFreeSurfaceFoam**.

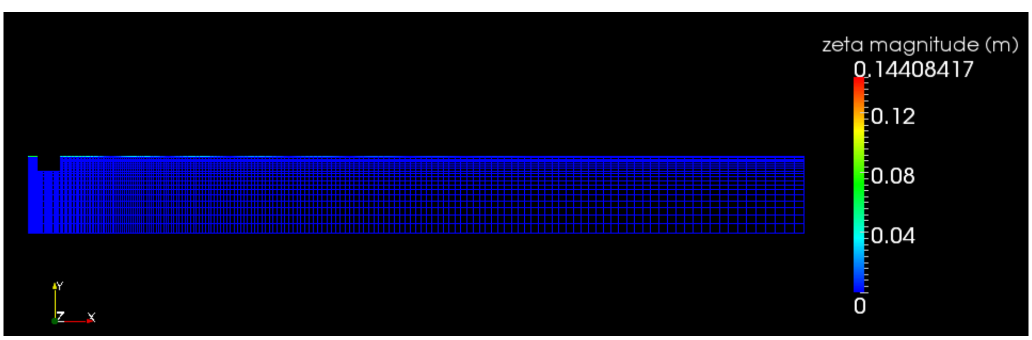

Figure 3.1: Visualization of the results of a modified version of the oscillatingBox tutorial. The mesh is not deformed, even though there are wave propagating at the free surface.

The free surface profile is represented by the `zeta` field. As can be seen, the grid shows no deformation even though there are waves at the free surface with an amplitude of about 10% of the domain height, 1 m.

A geometric representation of the free surface can be obtained in **paraView** using the filter

Figure 3.2: Visualization of the results of a modified version of the oscillatingBox tutorial, with the camera view from a 45° angle with the horizontal. The waves can be seen as property of the freeSurface patch.

"warp by vector", to deform the free surface patch. Only the points belonging to the free surface patch will be displaced.

3.2 waveSurfacePressure boundary condition

The waveSurfacePressure boundary condition is applied to boundaries corresponding to the free surface. It computes the wave height and the pressure change due the free surface profile change. The change in the free surface profile is calculated by integrating the fluid velocity at the free surface patch over time. The velocity is determined by equations 3.1 and 3.2:

$$\vec{u} = \frac{\phi}{\mathrm{d}A} \times \vec{n} \tag{3.1}$$

when the flux is the flow velocity (incompressible flow)

$$\vec{u} = \frac{\phi}{\mathrm{d}A \times \rho} \times \vec{n} \tag{3.2}$$

when the flux is flow mass flux (compressible and/or multiphase flow), where ρ is the density of the fluid, $\vec{u}$ is the velocity, $\mathrm{d}A$ is the cell face area and ϕ is the flux. For single phase incompressible flow, ϕ is equivalent to $\vec{u}$ and for compressible flow, ϕ is $\rho\vec{u}$.

Even though **potentialFreeSurfaceFoam** is an incompressible, single phase flow solver, waveSurfacePressure can handle both velocity fluxes (incompressible flow) and mass fluxes (generally used in compressible and/or multiphase flow). Since **potentialFreeSurfaceFoam** is an incompressible flow solver and waveSurfacePressure can handle both compressible and incompressible flows, there may be some confusion in the usage of the the terms dynamic pressure and kinematic pressure. Kinematic pressure is always the value obtained by dividing pressure by the fluid density. Dynamic pressure, however, may refer to either the pressure NOT divided the fluid density or to the part of the total pressure that is caused by dynamic effects. It is, therefore, possible to talk about "kinematic dynamic pressure" and "kinematic hydrostatic pressure" and "dynamic dynamic pressure" and "dynamic hydrostatic pressure". In this report, since **potentialFreeSurfaceFoam** only handles kinematic pressure, all references to dynamic pressure are to the dynamic part of the total pressure.

The pressure change due to the variation of the free surface position relative to the still water level, ζ, is computed by

$$\Delta p_\zeta = g\zeta \tag{3.3}$$

velocity fluxes or

$$\Delta p_\zeta = g\rho\zeta \tag{3.4}$$

for mass fluxes.

The waveSurfacePressure boundary condition is applied in the dynamic pressure field boundary condition.

Chapter 4

Including the dynamic mesh

4.1 Procedure

The modification of the **potentialFreeSurfaceFoam** solver was based on the codes of the **interFoam**, **interDyMFoam** (variation of **interFoam** for dynamic mesh handling), **pimpleFoam** and **pimpleDyMFoam** (variation of **pimpleFoam** for dynamic mesh handling) solvers.

The **interFoam** solver has a structure similar to **potentialFreeSurfaceFoam** in the way the PISO/PIMPLE loop is executed, making it convenient in the adaptations of the code related to the pressure equation.

pimpleFoam has an overall structure similar to **potentialFreeSurfaceFoam**, since both solvers only solve for single phase fluid domains. Thus it is ideal for the general adaptation of the **potentialFreeSurfaceFoam** code.

The specific modifications to be made to **potentialFreeSurfaceFoam** were determined by comparing the static and dynamic versions of the code of **interFoam** and **pimpleFoam** and finding the differences between them. Since the only difference between the static and dynamic versions is the possibility to handle dynamic meshes, all the differences in the source codes must be the implementation of dynamic meshes.

The modifications to **potentialFreeSurfaceFoam** can be divided in three different types:

- adding/removing files;
- changing code within specific files;
- changing/setting up cases to comply with the new solver.

The modifications that must be introduced in the original code are not related to solution algorithms, physics or other fundamental aspects. All the changes that must be applied are directly or indirectly associated with the frame of reference of the flux ϕ. In a static mesh solver, the mesh points do not change position in time. On the other hand, in a dynamic mesh, the points may move and there is the need to write the flux relative to the cell points. As a first approach, only files and/or code involving the flux ϕ (`phi` in the code) need to be adapted when coding dynamic meshes.

4.2 potentialFreeSurfaceFoam

In OpenFOAM 2.1.x, the **potentialFreeSurfaceFoam** solver source code is found in

```
$FOAM_SOLVERS/incompressible/potentialFreeSurfaceFoam
```

This folder has the following structure:

```
potentialFreeSurfaceFoam/
├── potentialFreeSurfaceFoam.C
├── potentialFreeSurfaceFoam.dep
├── createFields.H
├── UEqn.H
├── pEqn.H
└── Make/
    ├── options
    └── files
```

The files `potentialFreeSurfaceFoam.C`, `createFields.H`, `UEqn.H` and `pEqn.H` contain the source code of **potentialFreeSurfaceFoam**. The files in the `Make/` directory contain information for the compiler.

`createFields.H` contains the code which initializes the fields that will be used within the solver as, for example, pressure, `p`, dynamic pressure, `p_gh`, free surface elevation, `zeta`, gravitational acceleration, `g`, etc. `pEqn.H` contains the code that will solve the pressure equation in the PIMPLE algorithm. `Ueqn.H` contains the code that will solve the velocity equation in the PIMPLE algorithm. Finally, `potentialFreeSurfaceFoam.C` contains the code that calls the main OpenFOAM libraries and structures the sequence in which the different files, libraries, codes and algorithms are used, for example, the execution of the PIMPLE algorithm that will make use of `pEqn.H` and `Ueqn.H`.

On the compiler files, `Make/options` contains instructions to the compiler on where to look for libraries and files called in the solver source code and `Make/files` on where to write the compiled file of the solver.

The file `potentialFreeSurfaceFoam.dep` and files contained within other folders in `Make/` are generated when the solver is compiled and are not needed in the development of the new solver.

Both the main solver files and the compiler files will have to be adapted.

4.3 Modifying the solver

In the present section, the actions taken to modify the solver will be described. In some instances, suggestions of how to execute these actions will be given in the form of terminal commands. As far as possible, the commands required to preform a specific action will be mentioned in its subsection. However, there will be occasions when a history of commands from previous sections must have been executed beforehand, in order for the specific command mentioned to work correctly. Therefore, to properly follow the description of the modification of the solver, this section should be read sequentially. In the description of the modifications to the source code, when it is referred that a specific instruction or code was added to a line, it is assumed that all the code in that line and the lines below are moved down as much as necessary, except in the cases where the line in question is empty.

Since the new solver will be able to handle dynamic meshes, from now on it will be named **potentialFreeSurfaceDyMFoam**.

4.3.1 File structure

The modifications applied to **potentialFreeSurfaceFoam** were not executed directly in the code provided with OpenFOAM 2.1.x, but in a copy stored in a convenient working folder. In what follows, this folder is assumed to be:

```
$WM_PROJECT_USER_DIR/applications/solvers/incompressible/potentialFreeSurfaceDyMFoam
```

This folder was set-up executing the following commands in the terminal window:

```
cd $WM_PROJECT_DIR
cp -r --parents applications/solvers/incompressible/\
  potentialFreeSurfaceFoam $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR/applications/solvers/incompressible
mv potentialFreeSurfaceFoam potentialFreeSurfaceDyMFoam
cd potentialFreeSurfaceDyMFoam
```

4.3.2 Files

The first task in modifying **potentialFreeSurfaceFoam** is setting up the file structure. Since the new solver is named **potentialFreeSurfaceDyMFoam**, the file `potentialFreeSurfaceFoam.C` was renamed to `potentialFreeSurfaceDyMFoam.C`. This was executed with the following command:

```
mv potentialFreeSurfaceFoam.C potentialFreeSurfaceDyMFoam.C
```

As explained in 4.2, `potentialFreeSurfaceFoam.dep` and the any files and folders within `Make/` other than `files` and `options` are not useful. Therefore, they were deleted. This was performed with the following command:

```
wclean
```

As mentioned in 4.1, **pimpleFoam** and **pimpleDyMFoam** were the basis for the general modification of potentialFreeSurfaceFoam. The source code for **pimpleFoam** can be found in

```
$FOAM_SOLVERS/incompressible/pimpleFoam
```

and the source code for **pimpleDyMFoam** can be found in

```
$FOAM_SOLVERS/incompressible/pimpleFoam/pimpleDymFoam
```

A comparison between the file structure of **pimpleFoam** and **pimpleDyMFoam** (disregarding the `Make/` and `SRFPimpleFoam/` directories, as well as the compilation files `Allmake` and `pimpleFoam.dep`) reveals that **pimpleDyMFoam** has two extra files: `correctPhi.H` and `readControls.H`. These files were copied to the new solver folder:

```
cp -r $FOAM_SOLVERS/incompressible/pimpleFoam/pimpleDyMFoam/correctPhi.H .
cp -r $FOAM_SOLVERS/incompressible/pimpleFoam/pimpleDyMFoam/readControls.H .
```

4.3.3 Code

Since there is no reference to the flux ϕ in the `UEqn.H` file (in the code the flux is represented by the variable `phi`), it didn't require any change. The file `createFields.H` has a reference to `phi`. However, it is only to create the field and not to manipulate it. Therefore, this file didn't require any changes related to the way the flux is treated. It will required some other changes though, to be shown in detail in 4.3.4, but these are not related to the dynamic meshes.

The modifications to the `potentialFreeSurfaceDyMFoam.C` were mostly the declaration of header files and pieces of code that handle dynamic meshes in OpenFOAM.

Comparing `potentialFreeSurfaceDyMFoam.C` with `pimpleFoam.C`, the only differences between the two codes are just after the definition of the main function, in the order in which the different header files are declared and the position of `pimpleControl pimple(mesh)`:

pimpleFoam.C

```
int main(int argc, char *argv[])
{
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createFields.H"
    #include "initContinuityErrs.H"

    pimpleControl pimple(mesh);
```

potentialFreeSurfaceFoam.C

```
int main(int argc, char *argv[])
{
    #include "setRootCase.H"

    #include "createTime.H"
    #include "createMesh.H"

    pimpleControl pimple(mesh);

    #include "createFields.H"
    #include "initContinuityErrs.H"
```

Since both `potentialFreeSurfaceDyMFoam.C` and `pimpleFoam.C` have a very similar structure, the only changes required to `potentialFreeSurfaceDyMFoam.C` were the ones that occur between `pimpleFoam.C` and `pimpleDyMFoam.C`:

- in line 40 it was added

  ```
  #include "dynamicFvMesh.H"
  ```

- in line 51, the code

  ```
  #include "createMesh.H"
  ```

  was replaced by

  ```
  #include "createDynamicFvMesh.H"
  ```

- in line 64, the code

  ```
  #include "readTimeControls.H"
  ```

  was moved to line 52, after

  ```
  #include "createDynamicFvMesh.H"
  ```

  declared in the step above.

- in line 64 it was added

  ```
  #include "readControls.H"
  ```

- in lines 66 and 67 it was added

  ```
  // Make the fluxes absolute
  fvc::makeAbsolute(phi, U);
  ```

- in line 73 it was added

  ```
  mesh.update();

  if (mesh.changing() && correctPhi)
    {
      #include "correctPhi.H"
    }
  ```

```
    // Make the fluxes relative to the mesh motion
    fvc::makeRelative(phi, U);

    if (mesh.changing() && checkMeshCourantNo)
      {
        #include "meshCourantNo.H"
      }
```

The final code of the `potentialFreeSurfaceDyMFoam.C` file is the following:

potentialFreeSurfaceDyMFoam.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Application
    potentialFreeSurfaceFoam

Description
    Incompressible Navier-Stokes solver with inclusion of a wave height field
    to enable single-phase free-surface approximations

    Wave height field, zeta, used by pressure boundary conditions

    Turbulence modelling is generic, i.e. laminar, RAS or LES may be selected.

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

#include "fvCFD.H"
#include "singlePhaseTransportModel.H"
#include "turbulenceModel.H"
#include "dynamicFvMesh.H"
#include "pimpleControl.H"
#include "IObasicSourceList.H"

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

int main(int argc, char *argv[])
```

```
{
    #include "setRootCase.H"

    #include "createTime.H"
    #include "createDynamicFvMesh.H"
    #include "readTimeControls.H"
    pimpleControl pimple(mesh);

    #include "createFields.H"
    #include "initContinuityErrs.H"

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

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

    while (runTime.run())
    {
        #include "readControls.H"
        #include "CourantNo.H"
        // Make the fluxes absolute
        fvc::makeAbsolute(phi, U);
        #include "setDeltaT.H"

        runTime++;

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

        mesh.update();

        if (mesh.changing() && correctPhi)
        {
            #include "correctPhi.H"
        }

        // Make the fluxes relative to the mesh motion
        fvc::makeRelative(phi, U);

        if (mesh.changing() && checkMeshCourantNo)
        {
            #include "meshCourantNo.H"
        }

        // --- Pressure-velocity PIMPLE corrector loop
        while (pimple.loop())
        {
            #include "UEqn.H"

            // --- Pressure corrector loop
            while (pimple.correct())
            {
                #include "pEqn.H"
            }

            if (pimple.turbCorr())
            {
                turbulence->correct();
            }
        }

```

```
        runTime.write();

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

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

    return 0;
}


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

In the pEqn.H file, the modifications made were setting the flux phi to be referenced to an absolute initial mesh or relative to a moving, deforming mesh.

The changes were the following:

- in lines 11 to 14, the code

```
phi = (fvc::interpolate(U) & mesh.Sf())
    + fvc::ddtPhiCorr(rAU, U, phi);

adjustPhi(phi, U, p_gh);
```

  was replaced by

```
phi = (fvc::interpolate(U) & mesh.Sf());

if (ddtPhiCorr)
{
    phi += fvc::ddtPhiCorr(rAU, U, phi);
}

if (p.needReference())
{
    fvc::makeRelative(phi, U);
    adjustPhi(phi, U, p);
    fvc::makeAbsolute(phi, U);
}
```

- in line 48, the following code was added:

```
// Make the fluxes relative to the mesh motion
fvc::makeRelative(phi, U);
```

The final code is the following:

pEqn.H

```
1 volScalarField rAU(1.0/UEqn().A());
2 surfaceScalarField rAUf(rAU.name() + 'f', fvc::interpolate(rAU));
3
4 U = rAU*(UEqn() == sources(U))().H();
```

```

if (pimple.nCorrPISO() <= 1)
{
    UEqn.clear();
}

phi = (fvc::interpolate(U) & mesh.Sf());

if (ddtPhiCorr)
{
    phi += fvc::ddtPhiCorr(rAU, U, phi);
}

if (p.needReference())
{
    fvc::makeRelative(phi, U);
    adjustPhi(phi, U, p);
    fvc::makeAbsolute(phi, U);
}

// Non-orthogonal pressure corrector loop
while (pimple.correctNonOrthogonal())
{
    fvScalarMatrix p_ghEqn
    (
        fvm::laplacian(rAUf, p_gh) == fvc::div(phi)
    );

    p_ghEqn.setReference(p_ghRefCell, p_ghRefValue);

    p_ghEqn.solve(mesh.solver(p_gh.select(pimple.finalInnerIter())));

    if (pimple.finalNonOrthogonalIter())
    {
        phi -= p_ghEqn.flux();
    }
}

#include "continuityErrs.H"

// Explicitly relax pressure for momentum corrector
p_gh.relax();

// Make the fluxes relative to the mesh motion
fvc::makeRelative(phi, U);

p = p_gh + (g & (mesh.C() + zeta - refLevel));

U -= rAU*fvc::grad(p_gh);
U.correctBoundaryConditions();
sources.correct(U);
```

The last source code file that needed to be modified was `correctPhi.H`. The basic structure is well suited for **potentialFreeSurfaceDyMFoam**, but **pimpleDyMFoam** computes total pressure, `p` and **potentialFreeSurfaceFoam** computes dynamic pressure, `p_gh`. The instances of variable `p` in `correctPhi.H` had to be modified to `p_gh`. However, since there are several commands that use the letter p in the `correctPhi.H` file, a simple "find and replace all" command was not appropriate.

The change was performed manually, case by case. `p` was replaced by `p_gh` in

line 27 `p.boundaryField()...`	for `p_gh.boundaryField()...`
line 31 `forAll(p.boundaryField()...`	for `forAll(p_gh.boundaryField()...`
line 33 `if (p.boundaryField()...`	for `if (p_gh.boundaryField()...`
line 50 `..."pcorr", p.dimensions()...`	for `..."pcorr", p_gh.dimensions()...`
line 61 `...setReference(pRefCell, pRefValue)`	for `...setReference(p_ghRefCell, p_ghRefValue);`

There was a total of six occurrences in five lines that had be changed.

In **pimpleFoam** and **pimpleDyMFoam**, in the PIMPLE loop, the variable `rAU` is computed at the cell centres. In the case of **potentialFreeSurfaceFoam**, however, after determining the variable `rAU` at the cell centres, this variable is interpolated and computed at the cell faces, `rAUf`. It is this new variable that is used in the PIMPLE loop. The name of the variable `rAU` at the `correctPhi.H` file was, therefore, changed to `rAUf`:

- in line 58:

  ```
  fvm::laplacian(rAU, pcorr) == fvc::div(phi)
  ```

  was changed to

  ```
  fvm::laplacian(rAUf, pcorr) == fvc::div(phi)
  ```

The variable `rAUf` is not used in either **pimpleFoam** or **pimpleDyMFoam**, so it is not declared in these solvers. Also, the declaration of `rAUf` in **potentialFreeSurfaceDyMFoam** occurs after the call to `correctPhi.H`. This means that there will be a compilation error if the variable is not declared before the call to `correctPhi.H` or in `correctPhi.H` itself. The `correctPhi.H` file from **interDyMFoam**, has the same use of `rAUf` as potentialFreeSurfaceFoam and can be found in:

```
$WM_PROJECT_DIR/applications/solvers/multiphase/interFoam/interDyMFoam
```

Reviewing this file, it can be seen that after the creation of the `pcorr` volScalarField, the variable `rAUf` is declared in line 33:

```
dimensionedScalar rAUf("(1|A(U))", dimTime/rho.dimensions(), 1.0);
```

Unlike **interFoam** or **interDyMFoam**, **potentialFreeSurfaceFoam** works with kinematic pressure, so the variable `rAUf` created for **potentialFreeSurfaceDyMFoam** should not have the dimensions of time/ρ, but only of time. So, in line 53, the following code was added:

```
dimensionedScalar rAUf("(1|A(U))", dimTime, 1.0);
```

The final code of `correctPhi.H` is the following:

correctPhi.H

```
1  {
2      if (mesh.changing())
3      {
4          forAll(U.boundaryField(), patchI)
5          {
6              if (U.boundaryField()[patchI].fixesValue())
7              {
8                  U.boundaryField()[patchI].initEvaluate();
9              }
10         }
11
```

```
            forAll(U.boundaryField(), patchI)
            {
                if (U.boundaryField()[patchI].fixesValue())
                {
                    U.boundaryField()[patchI].evaluate();

                    phi.boundaryField()[patchI] =
                        U.boundaryField()[patchI]
                      & mesh.Sf().boundaryField()[patchI];
                }
            }
        }

        wordList pcorrTypes
        (
            p_gh.boundaryField().size(),
            zeroGradientFvPatchScalarField::typeName
        );

        forAll(p_gh.boundaryField(), patchI)
        {
            if (p_gh.boundaryField()[patchI].fixesValue())
            {
                pcorrTypes[patchI] = fixedValueFvPatchScalarField::typeName;
            }
        }

        volScalarField pcorr
        (
            IOobject
            (
                "pcorr",
                runTime.timeName(),
                mesh,
                IOobject::NO_READ,
                IOobject::NO_WRITE
            ),
            mesh,
            dimensionedScalar("pcorr", p_gh.dimensions(), 0.0),
            pcorrTypes
        );
        dimensionedScalar rAUf("(1|A(U))", dimTime, 1.0);
        while (pimple.correctNonOrthogonal())
        {
            fvScalarMatrix pcorrEqn
            (
                fvm::laplacian(rAUf, pcorr) == fvc::div(phi)
            );

            pcorrEqn.setReference(p_ghRefCell, p_ghRefValue);
            pcorrEqn.solve();

            if (pimple.finalNonOrthogonalIter())
            {
                phi -= pcorrEqn.flux();
            }
        }
    }

```

```
#include "continuityErrs.H"
```

The modifications presented above are the ones that must be applied to the source code of the solver to get dynamic meshes implemented. Before compilation, the ancillary files for the compiler also required changes.

In `Make/files` in the first line, `potentialFreeSurfaceFoam` was replaced with the name of the new solver, `potentialFreeSurfaceDyMFoam`. The destination of the compiled file in line 3 was replaced by:

```
EXE = $(FOAM_USER_APPBIN)/potentialFreeSurfaceDyMFoam
```

The final file is the following:

files

```
1  potentialFreeSurfaceDyMFoam.C
2
3  EXE = $(FOAM_USER_APPBIN)/potentialFreeSurfaceDyMFoam
```

In `Make/options` it was included the locations of where to look for the header and other files called in the new solver source code:

- in line 2 and 3, under `EXE_INC = \`, it was added

```
-I$(LIB_SRC)/dynamicMesh/lnInclude \
-I$(LIB_SRC)/dynamicFvMesh/lnInclude \
```

- in lines 11 and 12, under `EXE_LIBS = \` it was added

```
-ldynamicFvMesh \
-ltopoChangerFvMesh \
```

The final file is the following:

options

```
1
2   EXE_INC = \
3       -I$(LIB_SRC)/dynamicMesh/lnInclude \
4       -I$(LIB_SRC)/turbulenceModels/incompressible/turbulenceModel \
5       -I$(LIB_SRC)/transportModels \
6       -I$(LIB_SRC)/transportModels/incompressible/singlePhaseTransportModel \
7       -I$(LIB_SRC)/finiteVolume/lnInclude \
8       -I$(LIB_SRC)/meshTools/lnInclude
9
10 EXE_LIBS = \
11      -ldynamicFvMesh \
12      -ltopoChangerFvMesh \
13      -lincompressibleTransportModels \
14      -lincompressibleTurbulenceModel \
15      -lincompressibleRASModels \
16      -lincompressibleLESModels \
17      -lfiniteVolume \
18      -lmeshTools
```

4.3.4 A bug

During the executing of this project, a probable bug was found in **potentialFreeSurfaceFoam**. The computed total pressure values along a vertical line crossing the computational domain have a non-smooth and non-physical variation, as can be seen in figure 4.1

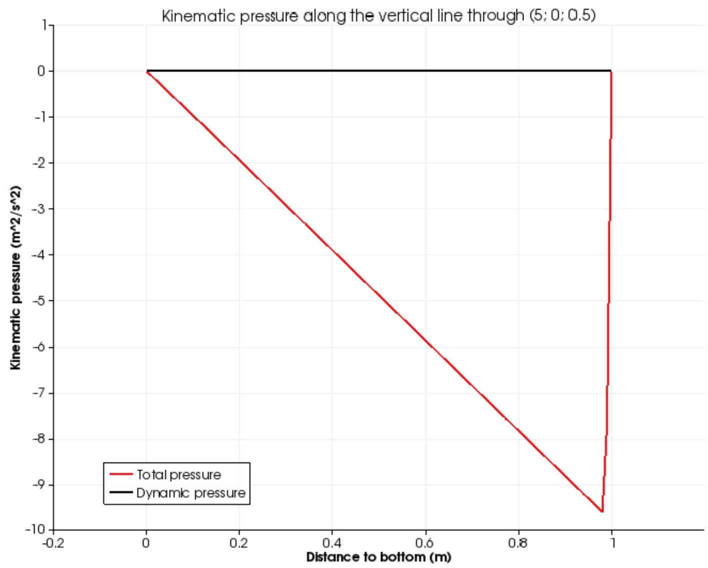

Figure 4.1: Total pressure along a vertical line crossing the domain in the tutorial case oscillatingBox of potentialFreeSurfaceFoam. The water surface is 1 m above the bottom. Even with values of dynamic pressure practically zero, the total pressure below the water surface is negative, indicating negative hydrostatic pressure.

In figure 4.1, it can be seen that, even though the dynamic pressure is practically zero, the total pressure below the water surface is negative and jumps from 0 m^2/s^2 at the free surface to -10 m^2/s^2 just below, which is not possible in incompressible flow. However, the pressure at the water surface is zero and the pressure in the interior domain increases linearly with depth, as expected, at the correct rate. In result, the pressure below the water surface is offset. This points to a problem with the pressure reference level within the solver, computing the correct value at the free surface and an offset value below. **potentialFreeSurfaceFoam** and **potentialFreeSurfaceDyMFoam** work with dynamic pressure throughout the solving process and, at the end, compute the total pressure by adding a reference hydrostatic pressure. Because of this, all the dynamic effects are actually computed correctly and only the total pressure is wrong. In the case of the dynamics of bodies interacting with the fluids, it is the total pressure that governs the time evolution and, therefore, this problem cannot be ignored. The way to correct this was to hard code the correct reference level in the **potentialFreeSurfaceDyMFoam** source code, since a dynamic way to correctly get the reference level was not found. Because of this, the reference level has to be set up and the solver recompiled every time a new reference is needed. The changes to the reference level were made in the file `createFields.H`:

- in line 76, the code

  ```
  dimensionedVector("zero", dimLength, vector::zero)
  ```

  was changed to

  ```
  dimensionedVector("one", dimLength, vector::one)
  ```

- lines 79 and 80 were commented out:

```
/*refLevel.boundaryField()[freeSurfacePatchI]
      == mesh.C().boundaryField()[freeSurfacePatchI];*/
```

In line 76, the vector is set to have the value one because the pressure should be referenced to the initial free surface pressure, which, in the tutorial "oscillatingBox" and in the test case presented in this report in chapter 5 is situated one meter above the bottom. Lines 79 and 80 should be the part of the code where the reference level is set for each part of the domain. However, these lines only set reference levels for the `freeSurface` patch, explaining why the free surface has the correct value computed.

In figure 4.2 is represented the pressure profile along a vertical line, with the changes to `createFields.H` described. As can be seen, the pressure has the correct variation with depth.

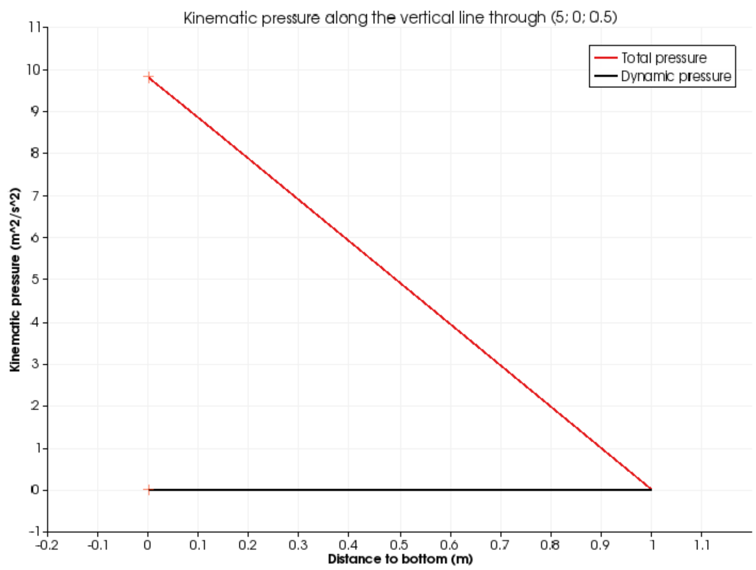

Figure 4.2: Total pressure along a vertical line crossing the domain in the tutorial case oscillatingBox of potentialFreeSurfaceFoam. With negligible values of the dynamic pressure, the total pressure now has the correct variation with depth.

The complete `createFields.H` file is the following:

options

```
1      Info<< "Reading field p (kinematic)\n" << endl;
2      volScalarField p
3      (
4          IOobject
5          (
6              "p",
7              runTime.timeName(),
8              mesh,
9              IOobject::MUST_READ,
10             IOobject::AUTO_WRITE
11         ),
12         mesh
13     );
```

```

    Info<< "Reading field U\n" << endl;
    volVectorField U
    (
        IOobject
        (
            "U",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );

    #include "createPhi.H"

    singlePhaseTransportModel laminarTransport(U, phi);

    autoPtr<incompressible::turbulenceModel> turbulence
    (
        incompressible::turbulenceModel::New(U, phi, laminarTransport)
    );

    #include "readGravitationalAcceleration.H"

    Info<< "\nReading freeSurfaceProperties\n" << endl;

    IOdictionary freeSurfaceProperties
    (
        IOobject
        (
            "freeSurfaceProperties",
            runTime.constant(),
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    );

    word freeSurfacePatch(freeSurfaceProperties.lookup("freeSurfacePatch"));
    label freeSurfacePatchI = mesh.boundaryMesh().findPatchID(freeSurfacePatch);
    if (freeSurfacePatchI < 0)
    {
        FatalErrorIn(args.executable())
            << "Patch " << freeSurfacePatch << " not found. "
            << "Available patches are:" << mesh.boundaryMesh().names()
            << exit(FatalError);
    }

    Info<< "Creating field refLevel\n" << endl;
    volVectorField refLevel
    (
        IOobject
```

```
            (
                "refLevel",
                runTime.timeName(),
                mesh,
                IOobject::NO_READ,
                IOobject::NO_WRITE
            ),
            mesh,
            dimensionedVector("ones", dimLength, vector::one)
        );

        /*refLevel.boundaryField()[freeSurfacePatchI]
            == mesh.C().boundaryField()[freeSurfacePatchI];*/

        Info<< "Creating field zeta\n" << endl;
        volVectorField zeta
        (
            IOobject
            (
                "zeta",
                runTime.timeName(),
                mesh,
                IOobject::READ_IF_PRESENT,
                IOobject::AUTO_WRITE
            ),
            mesh,
            dimensionedVector("zero", dimLength, vector::zero)
        );

        Info<< "Creating field p_gh\n" << endl;
        volScalarField p_gh
        (
            IOobject
            (
                "p_gh",
                runTime.timeName(),
                mesh,
                IOobject::MUST_READ,
                IOobject::AUTO_WRITE
            ),
            mesh
        );

        // Force p_gh to be consistent with p
        // Height is made relative to field 'refLevel'
        p_gh =  p - (g & (mesh.C() + zeta - refLevel));


        label p_ghRefCell = 0;
        scalar p_ghRefValue = 0.0;
        setRefCell(p_gh, pimple.dict(), p_ghRefCell, p_ghRefValue);


```

```
    IObasicSourceList sources(mesh);
```

This potential bug was reported.

After executing of the changes described, the new solver was compiled, executing the command `wmake`.

Chapter 5

Case set up

5.1 Introduction

This chapter describes the set up of a 2D simulation of a floating box, moving due to waves generated in a fluid. The case is based on the `oscillatingBox` tutorial of **potentialFreeSurfaceFoam**.

5.2 The basic case - oscillatingBox

To set up the case, first, the original oscillatingBox tutorial was copied to the USER working directory and renamed `oscillatingDyMBox`:

```
cd $WM_PROJECT_USER_DIR/run
cp -r $FOAM_TUTORIALS/incompressible/potentialFreeSurfaceFoam/\
  oscillatingBox oscillatingDyMBox
cd oscillatingDyMBox
```

5.3 Patches

A new patch, called `floatingBox` was created, empty, by adding to `constant/polyMesh/blockMeshDict` the following code:

```
floatingBox
{
    type wall;
    faces
    (
    );
}
```

below

```
floatingObject
{
    type wall;
    faces
    (
    );
}
```

The first patch simulates a box that moves with the waves; the second patch simulates a box that moves and generates waves. The patches are created without any geometric definition for two reasons. First, because it allows the fluid domain to be constructed with a single block, whose geometry is much simpler to specify and mesh; secondly, because it allows the floating objects to be defined (as will be shown later in 5.6) with a utility dictionary that will not require the geometry of the domain to be changed, making it simpler to change the floating body characteristics. The complete `blockMeshDict` file is the following:

blockMeshDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
    ( 0 0 0)
    (10 0 0)
    (10 1 0)
    ( 0 1 0)
    ( 0 0 0.1)
    (10 0 0.1)
    (10 1 0.1)
    ( 0 1 0.1)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (200 20 1) simpleGrading (10 0.1 1)
);

edges
(
);

boundary
(
    freeSurface
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
```

```
    walls
    {
        type wall;
        faces
        (
            (0 4 7 3)
            (2 6 5 1)
            (1 5 4 0)
        );
    }
    floatingObject
    {
        type wall;
        faces
        (
        );
    }
    floatingBox
    {
        type wall;
        faces
        (
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);

mergePatchPairs
(
);

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

5.4 Boundary and initial conditions

The changes made to boundary and initial conditions were the definition of the different initial and boundary values of the variables for the new `floatingBox` patch. Additionally, it was created a `pointDisplacement` file, that defines the rigid body and moving characteristics of the mesh points. In `0.org/p`, below

```
floatingObject
{
    type calculated;
    value uniform 0;
}
```

it was added

```
floatingBox
{
    type calculated;
    value uniform 0;
}
```

The final `p` file is the following:

p

```

/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform 0;

boundaryField
{
    freeSurface
    {
        type            calculated;
        value           uniform 0;
    }
    walls
    {
        type            calculated;
        value           uniform 0;
    }
    floatingObject
    {
        type            calculated;
        value           uniform 0;
    }
    floatingBox
    {
        type            calculated;
        value           uniform 0;
    }
    frontAndBack
    {
        type            empty;
    }
```

```
}


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

In `0.org/p_gh`, below

```
floatingObject
{
    type zeroGradient;
}
```

it was added

```
floatingBox
{
    type zeroGradient;
}
```

The complete `p_gh` file is the following:

p_gh

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      p_gh;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform 0;

boundaryField
{
    freeSurface
    {
        type            waveSurfacePressure;
        value           uniform 0;
    }
    walls
    {
        type            zeroGradient;
        value           uniform 0;
    }
    floatingObject
    {
```

```
        type            zeroGradient;
        value                 uniform 0;
    }
    floatingBox
    {
        type            zeroGradient;
        value           uniform 0;
    }
    frontAndBack
    {
        type            empty;
    }
}


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

In 0.org/U, below

```
floatingObject
{
    type oscillatingFixedValue;
    refValue uniform (0 1 0);
    offset (0 -1 0);
    amplitude table
    (
        (   0 0)
        (  10 0.025)
        (1000 0.025)
    );
    frequency       constant 1;
    value           uniform (0 0 0);
}
```

it was added

```
floatingBox
{
    type movingWallVelocity;
    value uniform (0 0 0);
}
```

The final U file is the following:

U

```
 /*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
```

```
    class       volVectorField;
    location    "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform (0 0 0);

boundaryField
{
    freeSurface
    {
        type            pressureInletOutletParSlipVelocity;
        value           uniform (0 0 0);
    }
    walls
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    floatingObject
    {
        type            oscillatingFixedValue;
        refValue        uniform (0 1 0);
        offset          (0 -1 0);
        amplitude       table
        (
            (   0     0)
            (  10 0.025)
            (1000 0.025)
        );
        frequency       constant 1;
        value           uniform (0 0 0);
    }
    floatingBox
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
    frontAndBack
    {
        type            empty;
    }
}


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

Since the `floatingBox` will be moving, it is required to specify how the movement of the box should be determined. This was done by setting up a file called `pointDisplacement`, that informs the solver how the mesh points should move throughout the computation. A file similar to the one required exists in the floatingObject tutorial of **interDyMFoam**. It was copied to the `0.org` folder of `oscillatingDyMBox` and modified as needed:

```
cp $FOAM_TUTORIALS/multiphase/interDyMFoam/\
  ras/floatingObject/0.org/pointDisplacement 0.org
```

In the new `pointDisplacement` file, the portion of code

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

was deleted. The name of the patch

```
stationaryWalls
```

was changed to

```
walls
```

(note the small w instead of capital w). The code

```
floatingObject
{
    type            sixDoFRigidBodyDisplacement;
    centreOfMass    (0.5 0.5 0.5);
    momentOfInertia (0.08622222 0.08622222 0.144);
    mass            9.6;
    rhoInf          1;  // needed only for solvers solving for kinematic pressure
    report          on;
    value           uniform (0 0 0);
}
```

was changed to

```
floatingBox
{
    type sixDoFRigidBodyDisplacement;
    centreOfMass (1.2 .9 0.05);
    momentOfInertia (0.08622222 0.144 0.08622222);
    mass 10;
    rhoInf 1000;  // needed only for solvers solving for kinematic pressure
    report on;
    value uniform (0 0 0);
}
```

This last part is where the six degree of freedom body characteristics of the floating box are attributed. Since the floating box in this tutorial has the name `floatingBox`, the name of the rigid body had to be changed from `floatingObject` to `floatingBox`. Then, the mass and inertia characteristics had to be defined. A special note to the value `rhoInf`. In the floatingObject tutorial of **interDyMFoam**, since **interDyMFoam** solves for dynamic pressure, the value of the fluid density is already included in all the calculations. **potentialFreeSurfaceFoam** and **potentialFreeSurfaceDyMFoam**, however, solve for kinematic pressure. This means that the forces calculated in **interDyMFoam**, as the integration of pressure over the body surface, are correctly determined, but, in **potentialFreeSurfaceDyMFoam** and **potentialFreeSurfaceFoam**, they are divided by the value of the fluid density. Therefore, in **potentialFreeSurfaceDyMFoam**, the fluid density, `rhoInf` has to be specified in this dictionary in order for the correct dynamics to be computed.

After the last patch, the information for the box that generates waves, `floatingObject`, was added. The box generating waves doesn't actually move, it just simulates an equivalent motion by applying, at its impermeable boundaries, an oscillatory velocity field on the fluid. To have this, the following code was added after the `floatingBox` patch:

```
floatingObject
{
    type fixedValue;
    value uniform (0 0 0);
}
```

The characteristics of the `freeSurface` patch were added, after the `floatingObject` patch:

```
freeSurface
{
    type fixedValue;
    value uniform (0 0 0);
}
```

and the `frontAndBack` patch was added at the end of the file

```
frontAndBack
{
    type fixedValue;
    value uniform (0 0 0);
}
```

Unlike what would be expected, the free surface patch doesn't move or deform with the waves, as was explained in 3.1.

Due to an effect called drift force, the floating box tends to move steadily forward with the waves. To prevent the mesh from deforming too much and collapsing, some restrictions were imposed on its motions. The box was only allowed to move vertically up and down and to rotate around the z axis (transverse to the domain). These restrictions were imposed by applying a `fixedLine` constraint and defining the moment of inertia around the x and y axes to be very large, say 1000000.

The fixedLine constrain was added to the `floatingBox` patch, after `value uniform (0 0 0);` with following code:

```
constraints
{
    maxIterations 500000000;
        fixedLine1
        {
               sixDoFRigidBodyMotionConstraint fixedLine;
               tolerance 1e-6;
               relaxationFactor 0.7;
               fixedLineCoeffs
               {
               refPoint (1.2 0.9 0.05);
               direction (0 1 0);
               }
         }
}
```

The final `pointDisplacement` file is the following:

pointDisplacement

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
```

```
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       pointVectorField;
    location    "0.01";
    object      pointDisplacement;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 0 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    Walls
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    floatingBox
    {
        type sixDoFRigidBodyDisplacement;
        centreOfMass (1.2 .9 0.05);
        momentOfInertia (1000000 1000000 0.08622222);
        mass 10;
        rhoName rhoInf;
        rhoInf 1000; // needed only for solvers solving for kinematic pressure
        report on;
        value uniform (0 0 0);
        constraints
        {
            maxIterations 500000000;
            fixedLine1
            {
                sixDoFRigidBodyMotionConstraint fixedLine;
                tolerance 1e-6;
                relaxationFactor 0.7;
                fixedLineCoeffs
                {
                    refPoint (1.2 0.9 0.05);
                    direction (0 1 0);
                }
            }
        }
    }
    floatingObject
    {
        type fixedValue;
        value uniform (0 0 0);
    }
    freeSurface
    {
        type fixedValue;
        value uniform (0 0 0);
    }
```

```
    frontAndBack
    {
        type fixedValue;
        value uniform (0 0 0);
    }
}
```

5.5 Mesh motion solution

Besides the motion characteristics of the rigid body, the dynamic characteristics of the mesh and how the adaptivity should be performed must also be defined. The adaptivity characteristics of the mesh are specified via a dictionary called `dynamicMeshDict` in the `constant/` directory. This dictionary doesn't exist in the original oscillatingBox tutorial. It was copied from the **interDyMFoam** tutorial floatingObject:

```
cp $FOAM_TUTORIALS/multiphase/interDyMFoam/\
  ras/floatingObject/constant/dynamicMeshDict constant
```

The `dynamicMeshDict` didn't require any significant changes. Only the moving object name had to be changed from `floatingObject` to `floatingBox`:

```
diffusivity inverseDistance (floatingObject);
```

was changed to

```
diffusivity inverseDistance (floatingBox);
```

The final `dynamicMeshDict` file is the following:

dynamicMeshDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      motionProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dynamicFvMesh      dynamicMotionSolverFvMesh;

motionSolverLibs ("libfvMotionSolvers.so");

solver            displacementLaplacian;

diffusivity       inverseDistance (floatingBox);


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

The solution method to the mesh motion is specified in `system/fvSolution`. Checking the `fvSolution` file of the `floatingObject` tutorial of **interDyMFoam**, the following code was added to `system/fvSolution` in the new case file:

```
cellDisplacement
{
    solver GAMG;
    tolerance 1e-5;
    relTol 0;
    smoother GaussSeidel;
    cacheAgglomeration true;
    nCellsInCoarsestLevel 10;
    agglomerator faceAreaPair;
    mergeLevels 1;
}
```

The final `fvSolution` file is the following:

fvSolution

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    cellDisplacement
    {
        solver          GAMG;
        tolerance       1e-5;
        relTol          0;
        smoother        GaussSeidel;
        cacheAgglomeration true;
        nCellsInCoarsestLevel 10;
        agglomerator    faceAreaPair;
        mergeLevels     1;
    }
    p_gh
    {
        solver          GAMG;
        tolerance       1e-7;
        relTol          0.1;
        smoother        DICGaussSeidel;
        nPreSweeps      0;
        nPostSweeps     2;
```

```
39            cacheAgglomeration true;
40            nCellsInCoarsestLevel 10;
41            agglomerator    faceAreaPair;
42            mergeLevels     1;
43            maxIter         100;
44        }
45
46        p_ghFinal
47        {
48            $p_gh;
49            tolerance       1e-7;
50            relTol          0;
51        }
52
53        U
54        {
55            solver          smoothSolver;
56            smoother        GaussSeidel;
57            tolerance       1e-7;
58            relTol          0.1;
59        }
60
61        UFinal
62        {
63            $U;
64            tolerance       1e-7;
65            relTol          0;
66        }
67    }
68
69    PIMPLE
70    {
71        momentumPredictor   no;
72        nOuterCorrectors    1;
73        nCorrectors         2;
74        nNonOrthogonalCorrectors 0;
75    }
76
77
78    // ************************************************************************* //
```

Finally, the libraries corresponding to all the extra functionalities added to the case must be declared in `system/controlDict`. In the end of the `controlDict` file, those libraries were declared using the following code:

```
libs
(
    "libOpenFOAM.so"
    "libincompressibleRASModels.so"
    "libfvMotionSolvers.so"
    "libforces.so"
);
```

In line 62, the reference to the `floatingObject` patch was changed to `floatingBox`.

5.6 Object geometry

The geometry of the floating boxes wasn't defined in `blockMeshDict`. It was defined using the utilities **topoSet** and **subSetMesh**. The complete description of this utilities is out of the scope of this report and will not be presented. The technical descriptions available are also very limited. **topoSet** operates on cells, faces and points creating named regions in the computational domain. In the this example, it was used to select a region of cells in the mesh that would later be eliminated to create the geometry of the floating bodies. It operates based on a dictionary. **subSetMesh** selects regions of cells, faces and points and performs operations on those regions. In this case, it was used to select the regions created with **topoSet** and eliminate them from the computational domain. More information about these two utilities can be found in `$FOAM_UTILITIES/mesh/manipulation/subSetMesh` and `$FOAM_UTILITIES/mesh/manipulation/topoSet`, including how to construct the required dictionaries.

The geometry of the box oscillating and generating waves, `floatingObject`, is already defined in the file `system/topoSetDict`. To generate the geometry for the box moving with the waves, `floatingBox`, another dictionary for **topoSet** was created, `topoSetDict2`. It was created as a copy of `topoSetDict`:

```
cp system/topoSetDict system/topoSetDict2
```

It was then adapted to create the geometry of `floatingBox`. The following code was deleted:

```
 {
     name    f0;
     type    faceSet;
     action  new;
     source  patchToFace;
     sourceInfo
     {
         name   freeSurface;
     }
 }

 {
     name    f0;
     type    faceSet;
     action  subset;
     source  boxToFace;
     sourceInfo
     {
         box (-100 0.9 -100) (0.2 100 100);
     }
 }

 {
     name    f0;
     type    faceZoneSet;
     action  new;
     source  setToFaceZone;
     sourceInfo
     {
         faceSet    f0;
     }
 }
```

All instances of `c0` were changed to `c1`, two in total. Finally in line 27

```
box (0.1 0.8 -100) (0.4 100 100);
```

was changed to

```
box (1 0.8 -100) (1.4 100 100);
```

The box field encloses a region is space, selecting all the elements contained within it. It thus defines the geometry of `floatingBox`.

The final `topoSetDict2` file is the following:

topoSetDict2

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.1.x                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       dictionary;
13     object      topoSetDict;
14 }
15 
16 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17 
18 actions
19 (
20     {
21         name    c1;
22         type    cellSet;
23         action  new;
24         source  boxToCell;
25         sourceInfo
26         {
27             box (1 0.8 -100) (1.4 100 100);
28         }
29     }
30 
31     {
32         name    c1;
33         type    cellSet;
34         action  invert;
35     }
36 
37 );
38 
39 // ************************************************************************* //
```

The remaining part was the removal of the previously selected cells, attributing to the faces of the interior domain that would be exposed when the selected cells were removed, the patch name of `floatingBox`. To keep the case as close as possible to the original floatingBox case, this was done in the `Allrun` script, since it is there that the commands for **subSetMesh** operating on the `floatingObject` patch are executed.

In `Allrun` in lines 13 to 16, the following code was added:

```
rm -r log.topoSet
rm -r log.subsetMesh
runApplication topoSet -dict system/topoSetDict2
runApplication subsetMesh -overwrite c1 -patch floatingBox
```

The lines

```
rm -r log.topoSet
rm -r log.subsetMesh
```

are only required because the `Allrun` script writes log files after the execution of the applications and, when **topoSet** and **subSetMesh** are executed for the second time, it will halt if those files already exist, created the first time the applications were executed. The final `Allrun` script is the following:

`Allrun`

```
1  #!/bin/sh
2  cd ${0%/*} || exit 1    # run from this directory
3
4  # Source tutorial run functions
5  . $WM_PROJECT_DIR/bin/tools/RunFunctions
6
7  # Set application name
8  application=`getApplication`
9
10 runApplication blockMesh
11 runApplication topoSet
12 runApplication subsetMesh -overwrite c0 -patch floatingObject
13 rm -r log.topoSet
14 rm -r log.subsetMesh
15 runApplication topoSet -dict system/topoSetDict2
16 runApplication subsetMesh -overwrite c1 -patch floatingBox
17 cp -r 0.org 0 > /dev/null 2>&1
18
19 runApplication $application
20
21 # ----------------------------------------------------------------- end-of-file
```

Analysing the `Allrun` script, it can be seen that the command to execute the solver in this case is

```
runApplication $application
```

There is no mention to the actual solver, only to a variable storing the name. The definition of the solver to be used is made in `system/controlDict`. Since this case was copied from a tutorial of **potentialFreeSurfaceFoam**, the solver was defined to be **potentialFreeSurfaceFoam**. To get the case running with **potentialFreeSurfaceDyMFoam**, in `system/controlDict`, in line 18, the code

```
application potentialFreeSurfaceFoam;
```

was changed to

```
application potentialFreeSurfaceDyMFoam;
```

The final `controlDict` file is the following:

Allrun

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     potentialFreeSurfaceDyMFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         20;

deltaT          0.001;

writeControl    adjustableRunTime;

writeInterval   0.02;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable yes;

adjustTimeStep  yes;

maxCo           0.4;

maxDeltaT       1;

functions
```

```
{
    forces
    {
        type                 forces;
        functionObjectLibs ("libforces.so");
        outputControl        outputTime;
        outputInterval        1;
        patches         (floatingBox);
        pName                p;
        UName                U;
        rhoName                rhoInf;
        log                true;
        rhoInf                1000;
        CofR                (0 0 0);
    }

    poolHeight
    {
        type            faceSource;
        functionObjectLibs ("libfieldFunctionObjects.so");
        enabled         true;
        outputControl   timeStep;
        timeInteval     1;
        log             true;
        valueOutput     false;
        source          faceZone;
        sourceName      f0;
        operation       areaAverage;
        fields
        (
            zeta
        );

    }
};


libs
(
    "libOpenFOAM.so"
    "libincompressibleRASModels.so"
    "libfvMotionSolvers.so"
    "libforces.so"
);

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

5.7 Running the case

Since a script, `Allrun`, was set up to automatically execute all sub commands required to solve this case, to run the case only `Allrun` had to be executed:

```
./Allrun
```

On a dual core 1.80 GHz laptop the total time to run the simulation was 496 s. The initial configuration of the case is represented in figure 5.1.

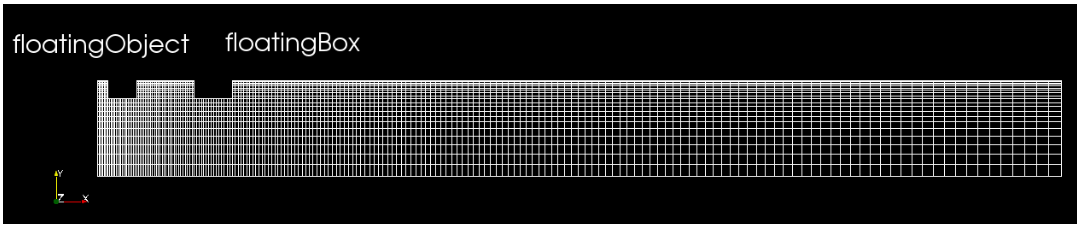

Figure 5.1: Initial configuration of the case.

Chapter 6

Results

The results of the case previously set up were visualized using **paraView**, through the `paraFoam` script:

```
paraFoam
```

In figure 6.1 it is represented the movement of the floating box after 0.66 s, where the deformation of the mesh due to the box movement is clear.

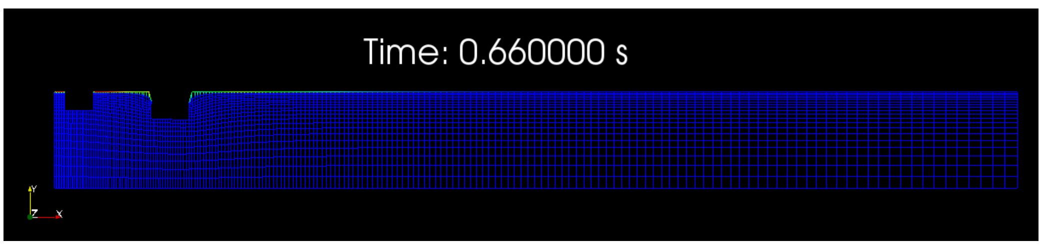

Figure 6.1: Movement of the floating box after 0.66 s. The deformation of the mesh is clearly visible.

As mentioned in 3.1, **potentialFreeSurfaceFoam** (and **potentialFreeSurfaceDyMFoam**, since it is based on **potentialFreeSurfaceFoam**) approximates the free surface profile by a field. Since there is no actual deformation of the free surface patch due to free surface waves, visualizing the free surface in **paraView** had to be done in one of two ways, each with its own limitations.

The first way to visualize the free surface field is to colour it, as it would be done for the pressure and velocity fields. The limitations of the way of visualizing are the fact that, in this tutorial, the cells of the `freeSurface` patch are very thin, making it very difficult to actually see the results when viewing the computational domain from the side. If the view angle is slightly changed, for example, rotating it so that the surface can be seen from above, the field is more discernible. However, since it is evolving over time, the colour scheme range has to constantly be adjusted to avoid saturation, figure 6.2.

The second way to visualize the results it to use the filter "warp by vector", selecting `zeta`, to be the governing parameter, figure 6.3. This way, a deformation of the free surface with an approximate geometry of the waves can be clearly seen. In some points, however, some non-physical distortions appear, caused by the fact that the grid must represent both the warped surface caused by the wave and the `floatingBox` geometry.

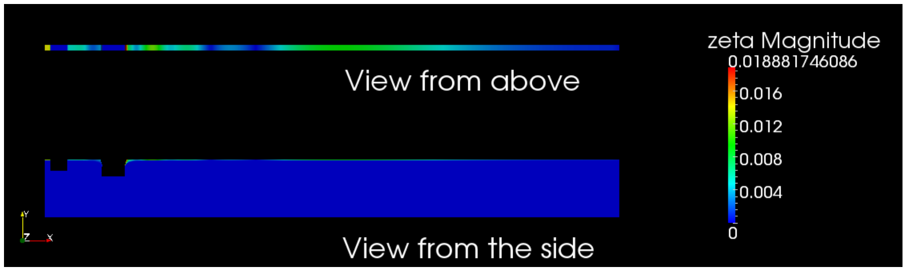

Figure 6.2: View of the surface profile, zeta, form the side and from above.

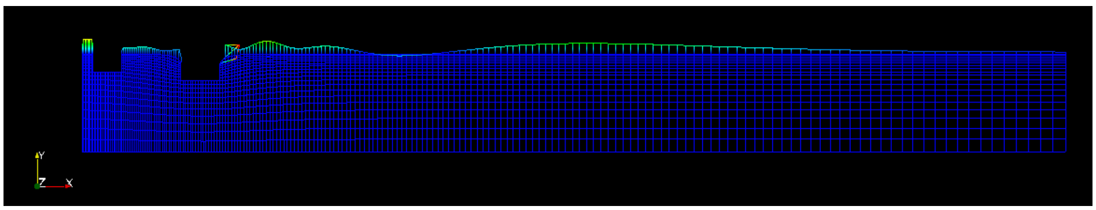

Figure 6.3: Warped representation of the zeta field. Some incorrect representations of the free surface can be seen near the box and just first wave, where the freeSurface cell faces intersect the opposite cell faces.

In figure 6.4 two zoomed in representations of the mesh are displayed.

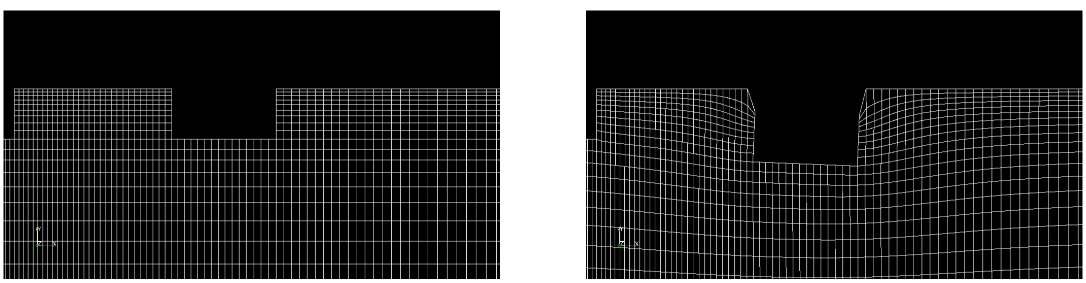

Figure 6.4: View of the lack of movement of the points defining the boundaries during the simulation, even though the floating object close to them is moving. Left - Initial mesh. Right - Deformed mesh.

As can be seen, even though the floating box is moving and deforming the mesh, the points defining the free surface boundary do not move. This is apparent by the extremely large deformation of the first layer of cells below the free surface near box limits, when compared to other deformed cells. This lack of adaptivity of the boundary points has two important consequences. First, it will cause the mesh to collapse even for very small movements the the body, if it intersects the free surface cells. Second, the dynamic free surface effects over the body when it is submerged will not be computed correctly. The overall result of this is the limitation of the simulations to bodies that do not get completely submerged and that only have very small movements around their initial position.

Chapter 7

Modifications to the work presented

The modifications proposed will be directed to test case presented and not to the solver.

The characteristics and number of bodies are easily modified. The shape of the objects can be modified by changing the dimensions of the boxes that define them in the corresponding **topoSet** dictionary file. In the same way, the position of the objects may be modified by translating the same boxes.

To add new objects to simulations, the corresponding new patches must be added to `0.org/U`, `0.org/p_gh`, `0.org/p`, `blockMeshDict` and `0.org/pointDisplacement` and new `topoSet` dictionary files must be created for each new body. The characteristics of the new patches to be added in these files are the exactly the same as for the `floatingBox`, except in `0.org/pointDisplacement`. In this file, if the body is intended to move, the correct rigid body characteristics (mass, moment of inertia, centre of gravity) must be individually defined for each body. As a starting point, the definitions for `floatingBox` may be used as a guidance. In case the object is to be stationary in the mesh, then, the definitions for it, in all files except `0.org/U`, are the same as for `floatingObject`. In `0.org/U`, if the body is intended to simulate a fictitious forced motion in the same fashion as `floatingObject`, then the motion definitions should be set here, in a similar manner as for `floatingObject`. Otherwise, it's definitions in `0.org/U` should be the same as for the remaining patches.

Any additional moving bodies must be declared the in the `constant/dynamicMeshDict` file, for the solver to know which bodies will cause mesh deformations.

The structure of the new **topoSet** dictionaries is the same of `system/topoSetDict2`. Only the box coordinates and set name must be changed. The set name is not required to have the same name as the body. However, each body must have a distinct set name in the **topoSet** dictionaries. To automatically create these new bodies for each simulations (i.e, without having to run **topoSet** and **subSetMesh** for each simulation), the `Allrun` script should also be changed. Below the last line executing the application **subSetMesh**, the following commands should be added:

```
rm log.topoSet
rm log.subSetMesh
run Applications topoSet -dict pathToNewTopoSetDictionary
runApplication subsetMesh -overwrite SetName -patch NewObjectName
```

where `pathToNewTopoSetDictionary` is the relative path to the **topoSet** dictionary for the new object, `SetName` is the name of the set created in the **topoSet** dictionary and `NewObjectName` is the name of the new object as defined in `blockMeshDict`, `0.org/U`, `0.org/p_gh`, `0.org/p` and `0.org/pointDisplacement`.

The amplitude and frequency of the wave generation can be changed in the file `0.org/U`, by changing the values of the `amplitude` and `frequency` fields of the `floatingObject` patch.

Chapter 8

Conclusions

The application of dynamic meshes to **potentialFreeSurfaceFoam** was successful. No validation of the actual solution was performed, as that was not the objective of the project.

The motion of the box had to be restrained in the horizontal direction to avoid large mesh deformations. This situation is not ideal and limits the applicability of **potentialFreeSurfaceDyMFoam**.

One problem with the approach of **potentialFreeSurfaceDyMFoam** is that floating bodies are constructed by removing cells from the top layers of the domain. Even though the dynamic mesh capability is applied to **potentialFreeSurfaceDyMFoam**, the points of the domain that define the boundaries are not able to move. This means that, if the floating body is allowed to move too much in any direction, the restrictions imposed by being connected to fixed points will cause the simulation to crash. This is not a problem with solvers such as **interDyMFoam**, since the floating bodies defined at the water free surface will not be connected the boundaries of the domain, but to the interior points.

Another problem with this approach is that by removing free surface boundary cells to define the floating body, in the time instants where the body is submerged, no free surface effects will be computed above the body, since the patch is not defined there. This will generate some errors in the propagation of the waves and in the forces acting on the body. The same effect happens when the body leaves the water surface, but in this last case, the simulation looses its validity, as **potentialFreeSurfaceDyMFoam** is not prepared to handle the water sloshing that would appear when the body fell back onto the water.

The floating bodies are created via **topoSet** and **subSetMesh**, by removing cells from the initial computational domain. This approach doesn't allow the definition of body geometries above the free surface.

The main limitations of the solver presented are the inability to cope with large mesh motions, the geometric definition of bodies that are only partially submerged, with part of their geometry above the free surface and the computation of the motion when the bodies are completely submerged.

Chapter 9

Future work

As recommendations for future work, the possibility of using the solver to compute large motions of bodies near the free surface, implying the motion of the boundaries points, should be investigated.

The modifications to the solver in order to accurately compute the solution when bodies that initially intersect the free surface get completely submerged should also be investigated;

Finally, determining ways to define bodies with geometry above the free surface, in a manner that is compatible with **potentialFreeSurfaceDyMFoam** should be investigated.

Bibliography

[1] Joel .H Ferziger and Milovan Peric. *Computational Methods for Fluid Dynamics.* Springer, 2002.

[2] OpenFOAM Foundation. Openfoam v2.1.0: Free surface flow. http://www.openfoam.org/version2.1.0/free-surface-flow.php, October 2012.