Cite as: Huang, L.: An opensource solver for wave-induced FSI problems, In proceedings of CFD with OpenSource Software, 2017, Edited by Nilsson. H., `http://dx.doi.org/10.17196/OS_CFD#YEAR_2017`

CFD WITH OPENSOURCE SOFTWARE

A COURSE AT CHALMERS UNIVERSITY OF TECHNOLOGY
TAUGHT BY HÅKAN NILSSON

Project work:

An opensource solver for wave-induced FSI problems

Developed upon foam-extented-4.0
Requires: FSI package, waves2Foam package

Author:
Luofeng Huang
Marine Research Group
Mechanical Engineering Dep.
University College London
ucemlhu@ucl.ac.uk

Peer reviewed by:
Minghao Li
Yeru Shang
Dimitrios Koukounas
Ebrahim Ghahramani

August 14, 2018

Acknowledgements

Thanks to my friend Minghao Li, for all the discussions throughout the whole project.

Thanks to my supervisors in UCL, Professor Giles Thomas and Professor Guoxiong Wu, who always provide me powerful support.

Especially, many thanks to Professor Håkan Nilsson, who has organised this OSCFD programme for ten years. It did inspire students from all over the world and significantly help the spread and development of OpenFOAM.

The author also much appreciates Lloyd Registration Foundation, UCL Faculty of Engineering Science and China Scholarship Council, for funding his PhD.

Learning outcomes

This tutorial will mainly teach four aspects: How to use it, The theory of it, How to implement it, 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 FSI package.
- How to use waves2Foam Package.

The theory of it

- The partitioned approach of FSI algorithm.
- The theory of free-surface modelling (VOF method)
- The theory of wave modelling (relaxation zone technique).

How it is implemented

- How to realize free-surface simulation in FSI package.
- How to couple waves2Foam with FSI package.

How to modify it

- How to build a new library/solver.
- How to set up a wave-induced FSI simulation.

Contents

Chapter 1

Introduction

Fluid-Structure Interaction (FSI), known as the interaction of some deformable structure with an internal or surrounding fluid flow, has broad applications in marine engineering field. A typical example can be the wave-induced deformation of ocean structures. This work will provide an open-source package to realize such a simulation using Computational Fluid Dynamics (CFD).

Previous efforts have been made to simulate FSI problems by an opensource CFD code, OpenFOAM. The most representative one can be the FSI package developed by Tukovic et al. [1]. This package uses a partitioned algorithm, which solves the fluid and solid mechanism separately and links them together via the fluid-solid interface. It equips good capability to simulate flow-induced solid deformation. However, the fluid solver of this package has not included a two-phase library, which means it cannot be used to simulate some marine problems containing free-surface modelling. Targeting at this gap, Li [2] imported a two-phase library into the original FSI package, but several unsolve issues were also indicated, e.g. instability. Besides, further work can be conducted to make the FSI package more applicable to a realistic case, for example, to simulate the solid deformation under waves loads.

This project can be considered as a further development of Li's [2]. The development route of this work is shown in Figure 1.1, where the original FSI package will first be extended into two-phase and then be endowed the capability of generating a target wave field.

This report expects a reader can comprehensively understand how to realize a wave-induce FSI simulation via OpenFOAM and be able to reproduce the process according to the provided instruction. It is organized as follows: Chapter 2 introduces the selected materials used to compose the final package, i.e. FSI package, free-surface module and wave module, where the code of the three materials and relevant CFD theories will be demonstrated. Chapter 3 provides a step-by-step explanation about how the source code are modified and then the three materials are eventually merged into one. Chapter 4 presents a tutorial of how to set up and simulate a wave-induced FSI case, by the developed code. The simulation results are also presented to discuss the capability and stability of the new package. Chapter 5 ends this study with conclusion and suggestions on future applications.

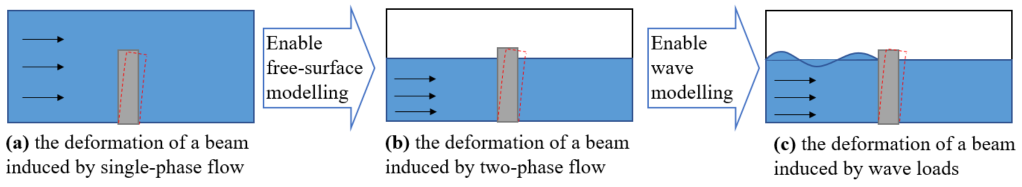
Enable
free-surface
modelling
Enable
wave
modelling
(a) the deformation of a beam induced by single-phase flow
(b) the deformation of a beam induced by two-phase flow
(c) the deformation of a beam induced by wave loads

Figure 1.1: Development route of this work

Chapter 2

Source packages

To realize wave-induced FSI simulations, three source packages (FSI package, free-surface module, wave module) are selected and will be merged into one. In this chapter, the three materials and their relevant theories are introduced separately.

2.1 FSI Package

This section introduces the basic of the FSI package, with details on the codes that relate to this work. A general introduction of this package can be found be the presentation of Yao [3].

The FSI package can be downloaded and installed with foam-extend 4.0. The instruction can be found at:

`https://openfoamwiki.net/index.php/Extend-bazaar/Toolkits/Fluid-structure_interaction`

This package contains two parts, `run` and `src` , which are the tutorial cases and source code respectively.

```
[FluidsolidInteraction]
|-- run
|-- |-- ampFsiFoam
|-- |-- fsiFoam
|-- |-- solidFoam
|-- |-- thermalStressFoam
|-- |-- weakFsiFoam
|-- src
    |-- Allwclean
    |-- Allwmake
    |-- fluidSolidInteraction
    |-- solvers
    |-- ThirdParty
    |-- utilities
```

The source code mainly consists two directories, `fluidSolidInteraction` and `solvers`.
The `fluidSolidInteraction` folder contains the `fluidSolvers`, `solidSolvers` and `fluidSolidInterface` (where the information between fluid and solid sides communicates).
The `solver` folder contains the process that how a FSI problem will be solved.

```
[src]
|-- Allwclean
|-- Allwmake
|-- fluidSolidInteraction
|-- |-- fluidSolidInterface
|-- |-- fluidSolvers
|-- |-- include
|-- |-- lnInclude
|-- |-- Make
|-- |-- solidSolvers
|-- solvers
|-- |-- ampFsiFoam
|-- |-- fluidFoam
|-- |-- fsiFoam
|-- |-- solidFoam
|-- |-- thermalSolidFoam
|-- |--weakFsiFoam
|-- ThirdParty
|-- |-- eigen3
|-- utilities
    |-- functionObjects
```

The core of this package is the solver `fsiFoam`, whose source file is located at `src/solvers/fsiFoam/fsiFoam.C`. It will call all the relevant code in the FSI package. To demonstrate the process of how this package solves a FSI problem, this code of `fsiFoam.C` is shown and explained as below:

- fsiFoam.C:line 49-64

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


    fluidSolidInterface fsi(mesh, solidMesh);

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

    for (runTime++; !runTime.end(); runTime++)
    {
        Info<< "Time = " << runTime.value()
            << " (dt = " << runTime.deltaT().value() << ")" << nl << endl;

        fsi.initializeFields();

        fsi.updateInterpolator();

        scalar residualNorm = 0;
```

The above code initializes an FSI simualtion, similar to a usual OpenFOAM solver, and then the following processes are executed for each timestel.

- fsiFoam.C:line 66-92

```
if (fsi.predictor())
        {
            fsi.updateForce();
            fsi.stress().evolve();

            residualNorm =
                fsi.updateResidual();
        }

        do
        {
            fsi.outerCorr()++;

            fsi.updateDisplacement();

            fsi.moveFluidMesh();

            fsi.flow().evolve();

            fsi.updateForce();

            fsi.stress().evolve();

            residualNorm =
                fsi.updateResidual();
        }
```

For each iteration (`outerCorr()++`), do:

`fsi.updateDisplacement();`
update the displacement of the solid. The displacement is solved by the solid solver, according to the fluid load on the solid body.

`fsi.moveFluidMesh();`
change the fluid mesh according to the deformed solid

`fsi.flow().evolve();`
go to fluid solver and solve the fluid state

`fsi.updateForce();`
update the fluid force on the solid, according to the calculated fluid state

`fsi.stress().evolve();`
go to solid solver and solve the solid stress/strain/displacement.

`residualNorm = fsi.updateResidual();`
update the residual. Here the `residualNorm` is the difference of fluid-interface-mesh displacement and solid-interface-mesh displacement, so this residual is set to keep the fluid and solid mesh coupled.

- fsiFoam.C:line 93-115

```
        while
        (
            (residualNorm > fsi.outerCorrTolerance())
         && (fsi.outerCorr() < fsi.nOuterCorr())
        );

        fsi.flow().updateFields();
        fsi.stress().updateTotalFields();

        runTime.write();

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

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

    return(0);
}


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

When the calculated residual satisfies the tolerance set or the iteration time exceeds the max value, update both the fluid and solid fields, go to next timestep. If not, go back and do the iteration process again.

To sum up, Figure 2.1 provides a process diagram to help illustrate how an FSI problem is solved by `fsiFoam`.

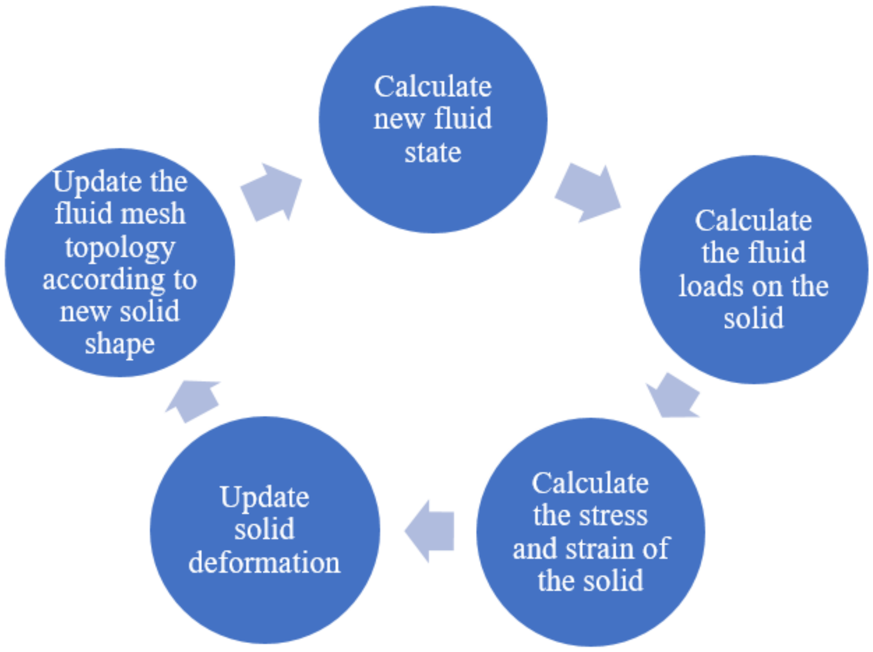
Calculate new fluid state
Calculate the fluid loads on the solid
Calculate the stress and strain of the solid
Update solid deformation
Update the fluid mesh topology according to new solid shape

Figure 2.1: The solving process of fsiFoam (for a single timestep)

2.2 Free-surface module

Free-surface modelling can be carried out by The Volume of Fluid (VOF) method [4]. This method introduces a parameter α to define the fractional volume of a cell occupied by a specific phase. Commonly, a value of $\alpha = 1$ corresponds to a cell full of liquid, and a value of $\alpha = 0$ indicates a cell full of gas. Thus, the free surface, which is a mix of the two phases, is formed by the cells with $0 < \alpha < 1$.
The present study will add the core code of `interDyMFoam` into the original FSI package to enable its free-surface modelling. The `interDyMFoam` solver, as described in its source file, is a "solver for 2 incompressible, isothermal immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach, with optional mesh motion and mesh topology changes including adaptive re-meshing.". The governing equations used by `interDyMFoam` are the continuity equation, the Navier-Stokes equation and the VOF equations:

$$\nabla \cdot U = 0 \tag{2.1}$$

$$\frac{\partial(\rho U)}{\partial t} + \rho \nabla U \cdot U = -\nabla P + \rho \nabla \cdot (\mu \nabla U) + \rho g \tag{2.2}$$

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot U\alpha = 0 \tag{2.3}$$

$$\rho = \alpha \rho_{water} + (1-\alpha)\rho_{air} \tag{2.4}$$

$$\mu = \alpha \mu_{water} + (1-\alpha)\mu_{air} \tag{2.5}$$

Where `U` is velocity, `P` is pressure, ρ is density, `t` is time, μ is dynamic viscosity and `g` is gravitational acceleration.
To give an introduction on the this solver, the `interDyMFoam.C` file is shown with explanations as below:

- interDyMFoam.C:line 62-74

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

    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"
```

For each timestep, the `readTimeControls.H` reads the control parameters like `DeltaT`, `adjustTimeStep`, `maxCo` and `correctPhi`. `CourantNo.H` calculates the mean and maximal courant number. `setDeltaT.H` will choose an optimal timestep if "adjustTimeStep" is enabled.

- interDyMFoam.C:line 76-81

```
        runTime++;

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

        bool meshChanged = mesh.update();
        reduce(meshChanged, orOp<bool>());
```

The topology of the mesh is updated over each timestep, as the `interDyMFoam` solver incorporates dynamic meshes. This is the difference between `interDyMFoam` and `interFoam`.

- interDyMFoam.C:line 83-99

```
#       include "volContinuity.H"

        volScalarField gh("gh", g & mesh.C());
        surfaceScalarField ghf("ghf", g & mesh.Cf());

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

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

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

`volContinuity.H` calculates volume continuity errors. Then, two fields `gh` and `ghf` are defined for further calculations. `meshCourantNo.H` calculates and outputs the mean and maximum courant numbers for the meshes. For a FSI problem, when the mesh changes over a timestep, the `correctPhi` is acquiescently true, then `correctphi.H` is executed, which makes sure the total inflow and outflow of mass is conserved after the mesh movement.
Here the fluxes are made relative to mesh motion as the meshes may have been changed, which corresponds the fluxes are made absolute (at the beginning of `interDyMFoam.C`) to initialize the case.

- interDyMFoam.C:line 101-144

```
        // Pressure-velocity corrector
        while (pimple.loop())
        {
            twoPhaseProperties.correct();

#           include "alphaEqnSubCycle.H"

#           include "UEqn.H"

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

            p = pd + rho*gh;

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

            turbulence->correct();
        }

        runTime.write();

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

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

    return 0;
}


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

Above is the main process of this solver, where `U` and `P` are solved. So far, the solution of a fluid field with free surface is obtained. This can be used as the fluid solver in the FSI package, which will provide the necessary capacity to simulate a free-surface FSI problem.

2.3 Wave module

In physics, wave is actually the distribution of the free surface. To model the loads of waves on a solid body, a desired wave field should be generated in the CFD model, which is known as a numerical wave tank.
In OpenFOAM, the `waves2foam` toolkit can be used to model a numerical wave tank. Its installation instruction and theoretical details can be found in the manual [5].
In the present study, `waves2foam` will be coupled with a two-phase FSI package, so that target waves can be generated and further induce solid deformation. The waves2foam toolkit uses a technique known as relaxation zone [6] to facilitate the modelling of a numerical wave tank. Commonly, two relaxation zones are set at the inlet and outlet of the domain, as the schematic diagram shown in Figure 2.2. These two zones can effectively help generate and absorb surface waves respectively. A relaxation zone can also be set to other shapes, e.g. cylindrical rather than rectangular.

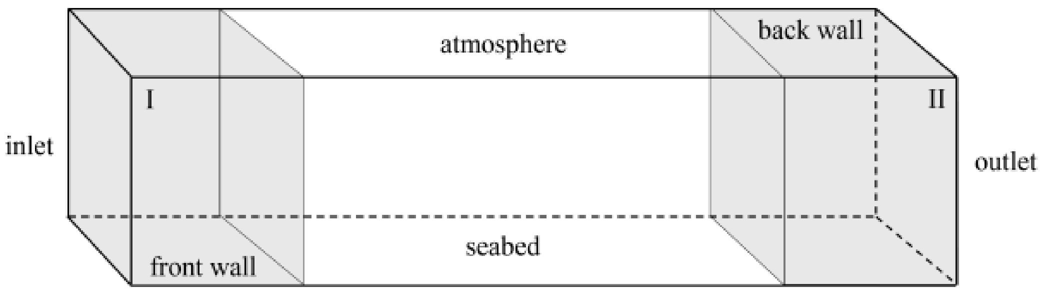

Figure 2.2: Schematic diagram for the inlet and outlet relaxation zones (grey) in a numerical wave tank. (Reproduced with permission of jacobsen [7])

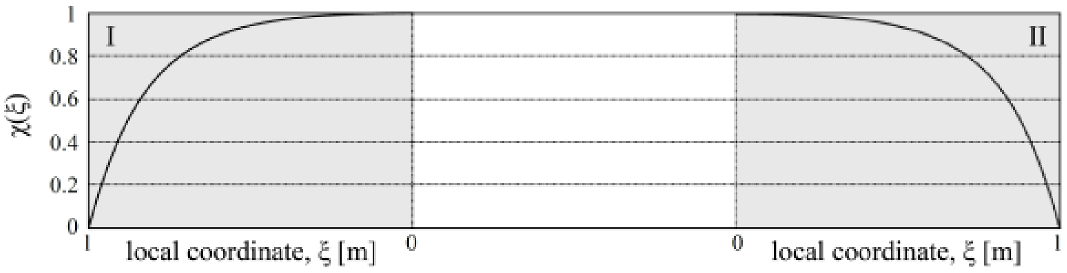

Figure 2.3: The value of spatial weighting factor, $\chi(\xi)$, as a function of local coordinate. (Reproduced with permission of jacobsen [7])

The relaxation zones can be employed to prevent wave reflection from the outlet boundary and also to prevent internally reflected waves, e.g. the waves reflected by internal structure to influence the wave generation at the inlet boundary. In the relaxation zone method, a spatial weighting factor χ is introduced as:

$$\chi(\xi) = 1 - \frac{exp(\xi^{\beta} - 1)}{exp(1) - 1} \tag{2.6}$$

where ξ is the local coordinate in the relaxation zone, which equals to 1 at the outer end and 0 at the inner end of the relaxation zone, as indicated in Figure 2.3. The shape factor β can be defined arbitrarily. Then a local value ϕ is dependent on χ as:

$$\phi = \chi\phi_{computed} + (1 - \chi)\phi_{target} \tag{2.7}$$

where ϕ_{target} is the target solution such as `U` or α, and $\phi_{computed}$ is the numerically computed value, obtained from the Navier-Stokes and VOF equations. Thus, the relaxation zone can obtain an adjusted ϕ over each timestep, thereby minimizing the interference caused by wave reflection. The target wave parameters are set according to wave theories through a file named `waveProperties.input`, which will be introduced as below.

To introduce how to set a target wave field, the `waveProperties.input` file under the tutorial case `waveFlume/constant` (within the installation of waves2Foam) will be explained as below:

- waveProperties.input:line 15-23

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

seaLevel        0.00;

// A list of the relaxation zones in the simulation. The parameters are given
// in <name>Coeffs below.
relaxationNames (inlet outlet);

initializationName outlet;
```

The file starts with the definition of still water level, the names of relaxation zones and the wave theory used to initiate the simulation (here it accords to the relaxation zone named outlet).

- waveProperties.input:line 25-60

```
inletCoeffs
{
    // Wave type to be used at boundary "inlet" and in relaxation zone "inlet"
    waveType    stokesFirst;

    // Ramp time
    Tsoft       2;

    // Water depth at the boundary and in the relaxation zone
    depth       0.400000;

    // Wave period
    period      2.0;

    // Phase shift in the wave
    phi         0.000000;

    // Wave number vector, k.
    direction  (1.0 0.0 0.0);

    // Wave height
    height      0.1;

    // Specifications on the relaxation zone shape and relaxation scheme
    relaxationZone
    {
        relaxationScheme Spatial;
        relaxationShape  Rectangular;
        beachType        Empty;

        relaxType   INLET;
        startX      (0 0.0 -1);
        endX        (5 0.0  1);
        orientation      (1.0 0.0 0.0);
    }
};
```

The wave type as well as wave parameters of the inlet relaxation zone are specified, followed by the definition of the relaxation zone. Here, stokesFirst waves are supposed to be generated and a rectangular relaxation zone is set at the inlet. Other options for `waveType`, `relaxationScheme`, `relaxationShape` and `beachType` are also available and can be selected from the munual [5].

- waveProperties.input:line 62-83

```
outletCoeffs
{
    waveType    potentialCurrent;
    U           (0 0 0);
    Tsoft       2;

    relaxationZone
    {
        relaxationScheme Spatial;
        relaxationShape  Rectangular;
        beachType        Empty;

        relaxType   OUTLET;
        startX      (13 0.0 -1);
        endX        (18 0.0  1);
        orientation      (1.0 0.0 0.0);
    }
};



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

The structure for setting the outlet relaxation zone is as the same as the setting of inlet but here it uses the wave theory “potentialCurrent”, which means introducing a current that is uniform over the depth (non-disturbed water). This theory is typically used for an outlet relaxation zone. At the beginning of this file, it defines “initializationName outlet”, which means this simulation will be initialized as a still water situation.
By employing the `waves2Foam` toolkit, the effect diagram and numerical view of a established numerical wave tank are presented as Figure 2.4 and Figure 2.5.

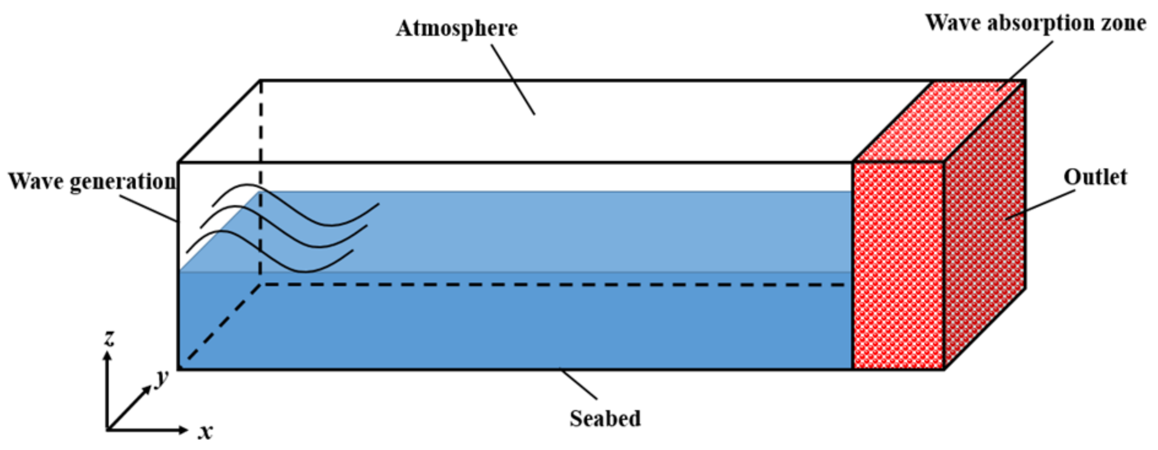

Figure 2.4: Effect diagram of a numerical wave tank, where wave generation and absorption function are realized by the application of relaxation zone

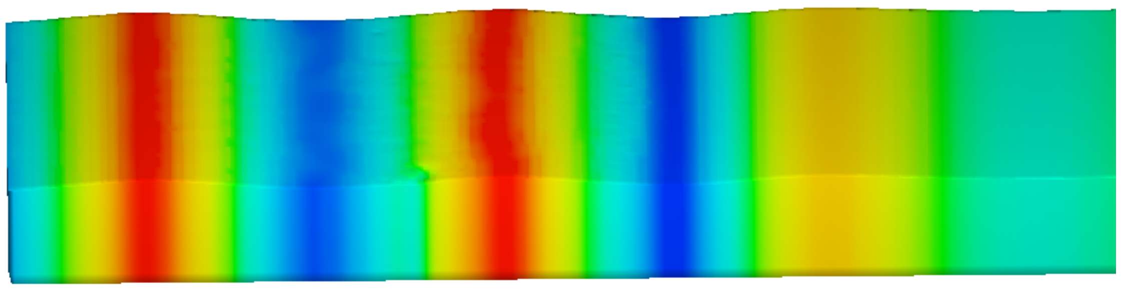

Figure 2.5: Numerical view of a wave flume, where regular waves are well generated and absorbed at the outlet boundary (red stands for crest and blue stands for trough).

Chapter 3

Code development

This chapter provides a guideline on how to develop a package for running a "wave-induced FSI simulation". Based on the explanation in the last chapter, the idea for the code development is using `interDyMFoam` to replace the fluid solver part of `fsiFoam` and coupling the new FSI package with `waves2Foam`. Before conducting the following steps, the reader should make sure that all the three parts described in the chapter 2 have been properly installed.

3.1 Enable two-phase FSI simulation

To extend the FSI package from single-phase to two-phase, essentially it is to build a new free-surface fluid library beside other existing fluid libraries, e.g. `pisoFluid, icoFluid`. The fluid libraries of FSI package are stored under `src/fluidSolidInteraction/fluidSolvers/`.
For this purpose, the code of `interDyMFoam` will be transplanted as a new two-phase fluid library under the `fluidSolvers` directory, which will be named as "interFluid". Detailed steps of building this library are shown as below:

1. Establish a new library and name it "interFluid"

```
cd FluidSolidInteraction/src/fluidSolidInteraction
cd fluidSolvers
cp -r pisoFluid interFluid
cd interFluid
mv pisoFluid.C interFluid.C
mv pisoFluid.H interFluid.H
sed -i s/pisoFluid/interFluid/g interFluid.*
```

2. Add the interFluid path into `FluidSolidInteraction/src/fluidSolidInteraction/Make/files`

```
cd ../..
echo "fluidSolvers/interFluid/interFluid.C" >> Make/files
```

3. Supply the content of `FluidSolidInteraction/src/fluidSolidInteraction/Make/options` so that it can call the free-surface package of foam-extend

```
ADD under EXE_INC:
    -I$(LIB_SRC)/transportModels/incompressible/lnInclude \
    -I$(LIB_SRC)/transportModels/interfaceProperties/lnInclude \
ADD under EXE_LIBS:
    -linterfaceProperties \
```

4. Add the headers required by `interDyMFoam` but not yet existing in `interFluid`

```
ADD after the headers of interFluid.H:
    #include "fvCFD.H"
    #include "dynamicFvMesh.H"
    #include "MULES.H"
    #include "subCycle.H"
    #include "interfaceProperties.H"
    #include "twoPhaseMixture.H"
    #include "turbulenceModel.H"
    #include "pimpleControl.H"
```

5. Update the member data of the `interFluid` library, according to `interDyMFoam.C` and its `createFields.H`

The member data are declared in interFluid.H, where the previous ones are deleted (except the `gradP` and `gradU`) and then revised as:

- member data declaration in interFluid.H

```
volScalarField pd_;
volScalarField alpha1_;
volVectorField U_;
surfaceScalarField phi_;
twoPhaseMixture twoPhaseProperties_;
dimensionedScalar rho1_;
dimensionedScalar rho2_;
volScalarField rho_;
surfaceScalarField rhoPhi_;
uniformDimensionedVectorField g_;//from readGravitationalAcceleration.H
volScalarField gh_;      // in interFoam but not in interDyMFoam
surfaceScalarField ghf_;// in interFoam but not in interDyMFoam
volScalarField p_;
interfaceProperties interface_;
autoPtr<incompressible::turbulenceModel> turbulence_;
volVectorField gradp_;//- Pressure gradient, from pisoFluid
volTensorField gradU_;//- Velocity gradient, from pisoFluid
pimpleControl pimple_;
```

Then, place their corresponding member data definition in `interFluid.C` (make sure in the same order as appear in `interFluid.H`), as:

- member data definition in interFluid.H

```
pd_
    (
        IOobject
        (
            "pd",
            runTime().timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    ),
alpha1_
    (
        IOobject
        (
            "alpha1",
            runTime().timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    ),
U_
    (
        IOobject
        (
            "U",
            runTime().timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    ),
phi_
(
    IOobject
    (
        "phi",
        runTime().timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    linearInterpolate(U_) & mesh.Sf()
),
twoPhaseProperties_(U_, phi_, "alpha1"),
rho1_( twoPhaseProperties_.rho1()),
rho2_( twoPhaseProperties_.rho2()),
rho_
    (
        IOobject
        (
            "rho",
            runTime().timeName(),
            mesh,
            IOobject::READ_IF_PRESENT
        ),
        alpha1_*rho1_ + (scalar(1) - alpha1_)*rho2_,//see Equation (2.4)
        alpha1_.boundaryField().types()//the same boundary types as alpha1
    ),
rhoPhi_
    (
        IOobject
        (
            "rho*phi",
            runTime().timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::NO_WRITE
        ),
        rho1_*phi_
    ),
g_
    (
     IOobject
        (
            "g",
            runTime().constant(),
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
     ),
gh_("gh", g_ & mesh.C()),
ghf_("gh", g_ & mesh.Cf()),
p_
    (
        IOobject
        (
            "p",
            runTime().timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        pd_ + rho_*gh_
    ),
interface_(alpha1_, U_, twoPhaseProperties_),
turbulence_
    (
        incompressible::turbulenceModel::New(U_, phi_, twoPhaseProperties_)
    ),
gradp_(fvc::grad(p_)),
gradU_(fvc::grad(U_)),
pimple_(const_cast <fvMesh&> (mesh))
```

Note: part of the syntax of `interDyMFoam` is not as same as that in the FSI package, therefore some expressions need to be revised for consistency, e.g. revising "const dimensiondScalar rho()" into "volScalarField rho()".

6. Revise the main member function of `interFluid.C` to conform with `interDyMFoam`
In the fluid library of FSi package, the main function is the `evolve()` function, as pointed in the step "fsi.flow().evolve()" of `fsiFoam.C`. Therefore the `evolve()` function in `interFluid.C` is revised

referring to the main member function of `interDyMFoam.C` (line 66-136)., as:

- evolve() function in interFluid.C

```
    for (int oCorr = 0; oCorr < nOuterCorr; oCorr++)

{

     #       include "interCourantNo.H"

     // Make the fluxes absolute
     fvc::makeAbsolute(phi_, U_);

     #       include "interVolContinuity.H"
     #           include "interCorrectPhi.H"


// Make the fluxes relative to the mesh motion
        fvc::makeRelative(phi_, U_);


// Pressure-velocity corrector
        while (pimple_.loop())
        {
            twoPhaseProperties_.correct();

#           include "interAlphaEqnSubCycle.H"

#           include "interUEqn.H"

            // --- PISO loop
            while (pimple_.correct())
            {
                    #           include "interPEqn.H"
            }

            p_ = pd_ + rho_*gh_;

            if (pd_.needReference())
            {
                p_ += dimensionedScalar
                (
                    "p",
                    p_.dimensions(),
                    pRefValue - getRefCellValue(p_, pdRefCell) //pRefValue changed to pdR...
                );
            }


gradp_ = fvc::grad(p_);   // reserve from FSI package

        gradU_ = fvc::grad(U_); // reserve from FSI package
```

```
            turbulence_->correct();
        }

}
```

7. Compile interFluid
Under `FluidSolidInteraction/src/fluidSolidInteraction`, `wmake` and debug till no error occurs.

8. Build a two-phase FSI solver, "interFsiFoam"

```
cd FluidSolidInteraction/src/solvers
cp -r fsiFoam interFsiFoam
cd interFsiFoam
mv fsiFoam.C interFsiFoam.C
sed -i s/fsiFoam/interFsiFoam/g Make/files.*
echo "-linterfaceProperties" >> Make/options
wmake
```

9. Test
The new `interFluid` and `interFsiFoam` can be tested by running the FSI dambreak case written by Li [2]. (Download available at: `http://www.tfd.chalmers.se/~hani/kurser/OS_CFD/`)

3.2 Enable wave-induced FSI simulation

The coupling of waves2Foam with FSI package is divided into two parts: on the one hand, to build a new solver that can call both waves2Foam package and FSI package; on the other hand, to include necessary wave objects into the new developed FSI fluid library (the "interFluid" that contains free-surface capability).
Detailed steps of the development process are shown as below:

1. Make a copy of `interFsiFoam` and name it "waveFsiFoam"

```
mkdir $WM_PROJECT_USER_DIR/applications/solvers/waveFsiFoam
cp -r $WM_PROJECT_USER_DIR/FluidSolidInteraction/solvers/interFsiFoam/* \
$WM_PROJECT_USER_DIR/applications/solvers/waveFsiFoam
cp -r $WM_PROJECT_DIR/applications/solvers/multiphase/interFoam/ \
alphaEqn.H alphaEqnSubCycle.H createFields.H UEqn.H pEqn.H \
$WM_PROJECT_USER_DIR/applications/solvers/waveFsiFoam
cd $WM_PROJECT_USER_DIR/applications/solvers/waveFsiFoam
mv interFsiFoam.C waveFsiFoam.C
```

2. Revise `waveFsiFoam/Make/files`, as:

```
waveFsiFoam.C
EXE = $(FOAM_USER_APPBIN)/waveFsiFoam
```

3. Supply the content of `waveFsiFoam/Make/options`, so that it can call the `waves2Foam` package (this step refers to `waves2Foam/application/solvers/solvers400_EXT/waveFoam/Make/options`, as this work is based on foam-extend 4.0)

```
ADD under EXE_INC:
    -DOFVERSION=400 \
    -DEXTBRANCH=1 \
    -DOFPLUSBRANCH=0 \
    -I$(WAVES_SRC)/waves2Foam/lnInclude \
    -I$(WAVES_SRC)/waves2FoamSampling/lnInclude \
    -I$(WAVES_GSL_INCLUDE)
ADD under EXE_LIBS:
    -L$(WAVES_LIBBIN) \
    -lwaves2Foam \
    -lwaves2FoamSampling \
    -L$(WAVES_GSL_LIB) \
    -lgsl \
    -lgslcblas
```

4. Revise the `Make/options` again so that it can call the FSI package

```
Under EXE_INC, link the following paths with the installation position of FSI, as:
    -I../../../FluidSolidInteraction/src/fluidSolidInteraction/lnInclude \
    -I../../../FluidSolidInteraction/src/ThirdParty/eigen3 \
```

5. Complie `waveFsiFoam`
Under the `waveFsiFoam` directory, `wmake` and debug till no error occurs.

6. Make a copy of `interFluid` and name it "waveInterFuild"

```
cd FluidSolidInteraction/src/fluidSolidInteraction
cd fluidSolvers
cp -r interFluid waveInterFluid
cd waveInterFluid
mv interFluid.C waveInterFluid.C
mv interFluid.H waveInterFluid.H
sed -i s/interFluid/waveInterFluid/g waveInterFluid.*
```

7.Add the path of `waveInterFluid` into
`FluidSolidInteraction/src/fluidSolidInteraction/Make/files`

```
cd ../..
echo "fluidSolvers/waveInterFluid/waveInterFluid.C" >> Make/files
```

8. Add `relaxationZone` class into the `waveInterFluid` library

```
ADD at the end of the head files in waveInterFluid.H:
    #include "relaxationZone.H"
ADD at the end of the member data in waveInterFluid.H:
    relaxationZone relaxing;
ADD at the end of the member function in waveInterFluid.C:
    relaxing(const_cast <fvMesh&> (mesh), U_, alpha1_)
ADD under {twoPhaseProperties_.correct(), in waveInterFluid.C:
    relaxing.correct();
```

9. Add `waveProperties` class into the `waveInterFluid` library (should be added after the construction of "g", as the wave theories need to call the gravitational acceleration)

```
ADD after the member data of "g", in waveInterFluid.H+
     IOdictionary waveProperties;
     dimensionedVector referencePoint;
     scalar sL;
ADD after the member function of "g", in waveInterFluid.C+
      waveProperties
    (
        IOobject
        (
            "waveProperties",
            runTime().constant(),
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    ),
    referencePoint("NULL", dimLength, vector::zero),
    sL(readScalar(waveProperties.lookup("seaLevel"))),
ADD after "Info << "Evolving fluid solver" << endl;", in waveInterFluid.C
     if (SMALL < Foam::mag(sL))
     {
      if (Switch(waveProperties.lookup("seaLevelAsReference")))
        {
         // Make a positive unit vector along the direction of gravity
            referencePoint.value() = g_.value()/Foam::mag(g_.value());
            referencePoint.value() = Foam::cmptMag(referencePoint.value());

            // Make the sea level the reference level
         referencePoint.value() *= sL;
        }
      }
```

10. Compile `waveInterFLuid`:
Under `FluidSolidInteraction/src/fluidSolidInteraction`, `wmake` and debug till no error occurs.

11. Test: the usage of the developed code will be introduced in Chapter 4.

Chapter 4

Tutorial case

This chapter will set up a case to demonstrate how a wave-induced FSI problem is simulated through the code developed in this work. A tutorial case, which can be considered as "beam deformation in regular waves", will be demonstrated. The case is revised based on a basic case "beamInCrossFlow" (within the installation of original FSI package). The simulation view of the "beamInCrossFlow" case is shown in Figure 4.1.

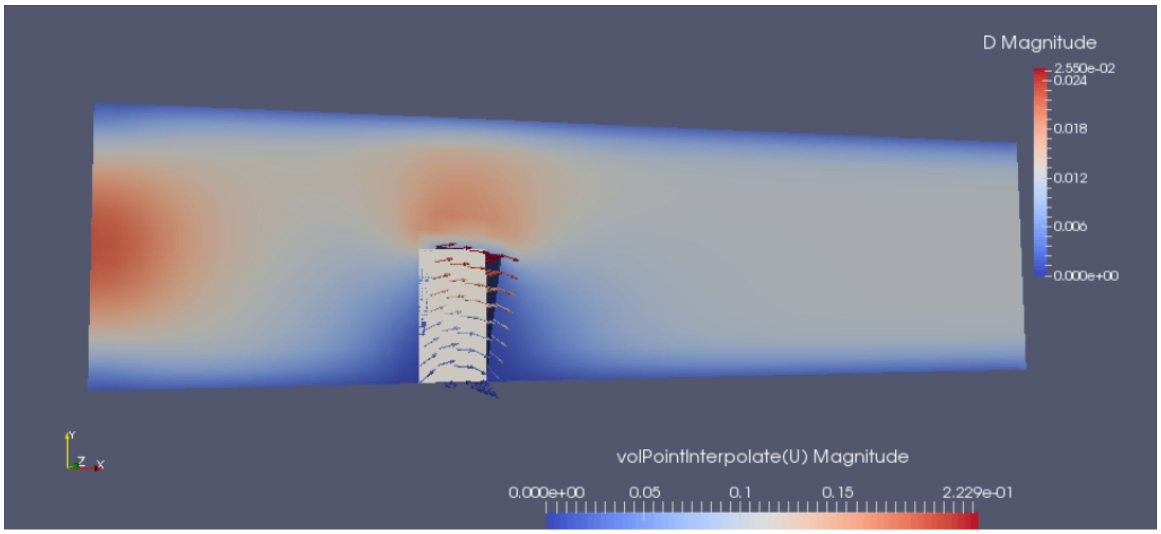

Figure 4.1: Simulation view of the tutorial case "beamInCrossFlow", where the beam is being deformed by the moving flow

4.1 Case structure

As shown in the tree diagram below, the FSI case mainly consists of two parts: `fluid` and `solid`. Each of them has its own `0`, `constant` and `system` directory, as a common OpenFOAM case, and the "createZones" and "setBatch" define the fluid/solid interface. The "makeLinks", "makeSerialLinks" and "removeSerialLinks" files manage the link between the fluid and solid. The "Allrun" and "Allclean" files are located in the fluid part, and the "AllrunPar" file is used for parallel computation. During the process of an FSI simulation, only the fluid part needs to be executed, because the solid part will be called automatically over each timestep.

```
[A FSI CASE]
|-- fluid
|--  |-- 0
|--  |-- Allclean
|--  |-- Allrun
|--  |-- AllrunPar
|--  |-- constant
|--  |-- createZones
|--  |-- foam.foam
|--  |-- gpDeflection
|--  |-- setBatch
|--  |-- system
|-- makeLinks
|-- makeSerialLinks
|-- removeSerialLinks
|-- solid
    |-- 0
    |--constant
    |-- createZones
    |-- foam.foam
    |-- setBatch
    |-- system
```

4.2 Mesh

The mesh of a FSI case also contains two parts, i.e. fluid mesh and solid mesh, defined separately in their `constant/polyMesh/blockMeshDict`. The fluid mesh and solid mesh must match each other through their `interface` and compose a whole computational domain, as shown in Figure 4.2.

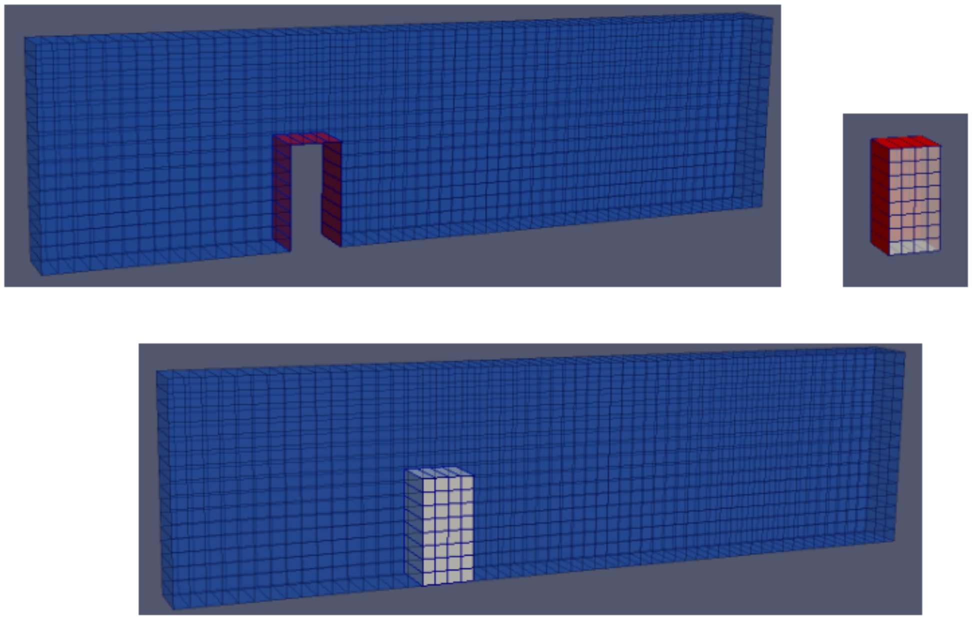

Figure 4.2: Fluid mesh, solid mesh and the integral mesh. (Blue: fluid field; Red: interface; Grey: Solid field)

Through the interface, the loads of fluid on solid is outputted to the solid solver and converted into the displacement of the solid surface (here the solid bottom displacement is constrained at zero, following the beam constraint). According to the solid deformation, dynamic mesh capacity is employed to make the fluid mesh deform together. Figure 4.3 gives an example of how the fluid mesh changes with solid deformation.

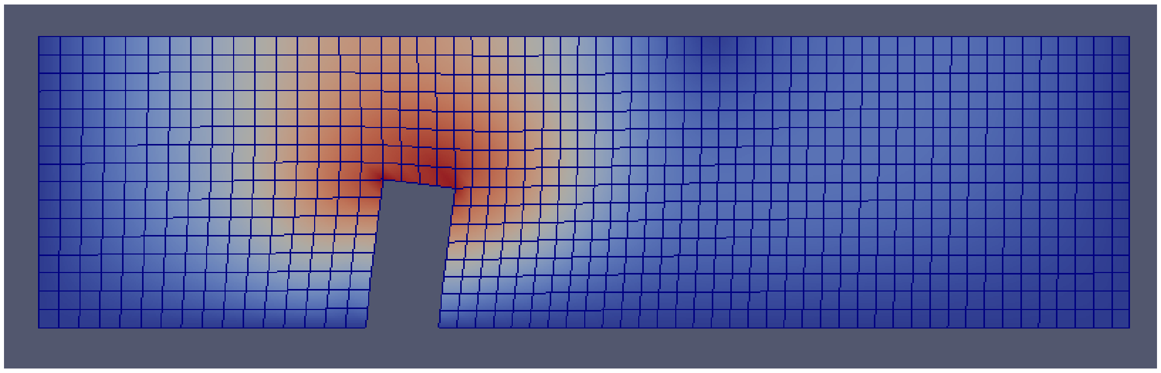

Figure 4.3: The topology of the fluid mesh changes according to solid surface displacement.

4.3 Boundary conditions

Compared with the original "beamInCrossFlow" case , the following revises are required under `fluid/0/` :
1. Copy the `alpha1.org` file from the tutorial case "damBreakWithoutObstacle" (within the installation of foam-extend 4.0), so that the VOF method will divide the domain into two phase.
2. Rename the pressure field from `p` to `pd`; Revise its outlet boundary into a `zeroGradient` condition and its top boundary into a `totalPressure` (typically used to model the atmosphere) condition.
3. To generate waves at the inlet boundary, the inlet boundary conditions of volume fraction `alpha1.org` and velocity `U` are set as `waveAlpha` and `waveVelocity` respectively, which are the boundary conditions installed with the `waves2Foam` package.

```
For alpha1.org
        inlet
        {
         type waveAlpha;
         refValue uniform 0;
         refGrad uniform 0;
         valueFraction uniform 1;
         value uniform 0;
        }
For U
        inlet
        {
         type waveVelocity;
         refValue uniform ( 0 0 0 );
         refGradient uniform ( 0 0 0 );
         valueFraction uniform 1;
         value uniform ( 0 0 0 );
        }
```

4.4 Constant

Compared with the original “beamInCrossFlow” case, the following revises are required under `fluid/constant/` :
1. In `fluidProperties`, set the “fluidslover” as `waveInterFluid` (this step calls the new fluid library developed in this study), also change the following “fluidcoeffs” value into `waveInterFluidCoeffs`.
2. Replace the `transportProperties` file by that of the tutorial case “damBreakWithoutObstacle”.
3. Copy `waveProperties.input` from the toturial case “waveFlume”, as well as the `g` file and `RASproperties` file. Adjust the “sealevel” and other wave parameters according to the geometry.

Note: the deformation properties of the solid is set in `solid/constant/rheologyProperties`, as:

- rheologyProperties:line 15-27

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

planeStress no;// yes for 2d no for 3d

rheology
{
    type linearElastic;
    rho rho [1 -3 0 0 0 0 0] 1000; //density
    E E [1 -1 -2 0 0 0 0] 1.4e10;  //Young's modulu
    nu nu [0 0 0 0 0 0 0] 0.4;    //Poissson ratio
}

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

4.5 Running the case

Compared with the original “beamInCrossFlow” case, the following revises are required under `fluid/system/` :
1. In `controlDict`, change the “application” value into `waveFsiFoam` (this step calls the developed new solver) and comment the previous object functions.
2. Replace the `fvSchemes` file and `fvSolution` file by those of the tutorial case “damBreakWithoutObstacle”, and revise the pressure field of the two files from `p` to `pd`.

To run the case by the command “./Allrun”, the `fluid/Allrun` file needs to be revised as below:

- Allrun:line 1-19

```
#!/bin/sh

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

# Get application name
application=`getApplication`
```

```
runApplication -l log.blockMesh.solid blockMesh -case ../solid
runApplication -l log.setSet.solid setSet -case ../solid -batch ../solid/setBatch
runApplication -l log.setToZones.solid setsToZones -case ../solid -noFlipMap

runApplication blockMesh
runApplication setSet -batch setBatch
runApplication setsToZones -noFlipMap

cd ..

./makeSerialLinks fluid solid
```

The basic operations (`blockMesh, Setset, setTozones`) of both the solid part and fluid part are made to set the mesh and interface, and then the two parts are linked.

- Allrun:line 21-30

```
cd fluid
cp 0/alpha1.org 0/alpha1


runApplication setWaveParameters
runApplication setWaveField

runApplication $application

# ----------------------------------------------------------------- end-of-file
```

The utility `setWaveParameters` is a pre-processing utility, which computes all the necessary wave parameters based on physical meaningful properties, e.g. `setWaveParametes` converts information on water depth and wave period into a wave number for first order Stokes wave theory. In this step, it will load the `constant/waveProperties.input` and output the processed data into a new file, `constant/waveProperties` .
The utility `setWaveField` is used to set the initial conditions according to a user defined wave theory, which is defined by the keyword “initializationName” in the file `waveProperties.input` (see Section 2.3).
The last step calls the solver `waveFsiFoam`, as defined by “getAppapplication” in `fluid/controlDict`. Thus, the new solver `waveFsiFoam` will solve the case.

4.6 Results and discussion

In this case, regular ocean waves are generated from the left and propagating towards right, as shown in Figure 4.4. Figure 4.5 represents the interactions of the beam with incoming waves: when the beam is hit by a wave crest, it deforms forward; when the beam is hit by a wave trough, it deforms backward. Figure 4.6 plots the beam deformation over time, in which it can be seen that the simulation reaches converge after a while and the beam deformation reveals a periodically sinusoidal response under the regular wave loads. Stable periodical results can be expected if the simulation continues running.

So far, it is the scope of this work. According to the analysis above, the solver shows good stability and the result is qualitatively physical. It should be noted that this case is set with coarse mesh and used to test the capability of the developed package. To improve this case, finer mesh can be applied and further work can be conducted on verifying wave quality and validating the accuracy of the beam deformation.

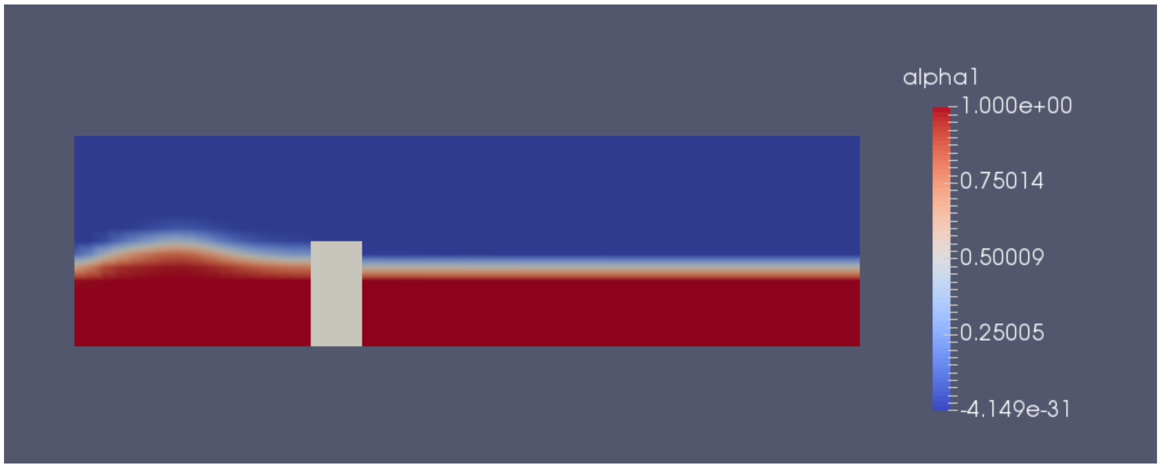

Figure 4.4: Wave generation: from the left and propagating towards right.

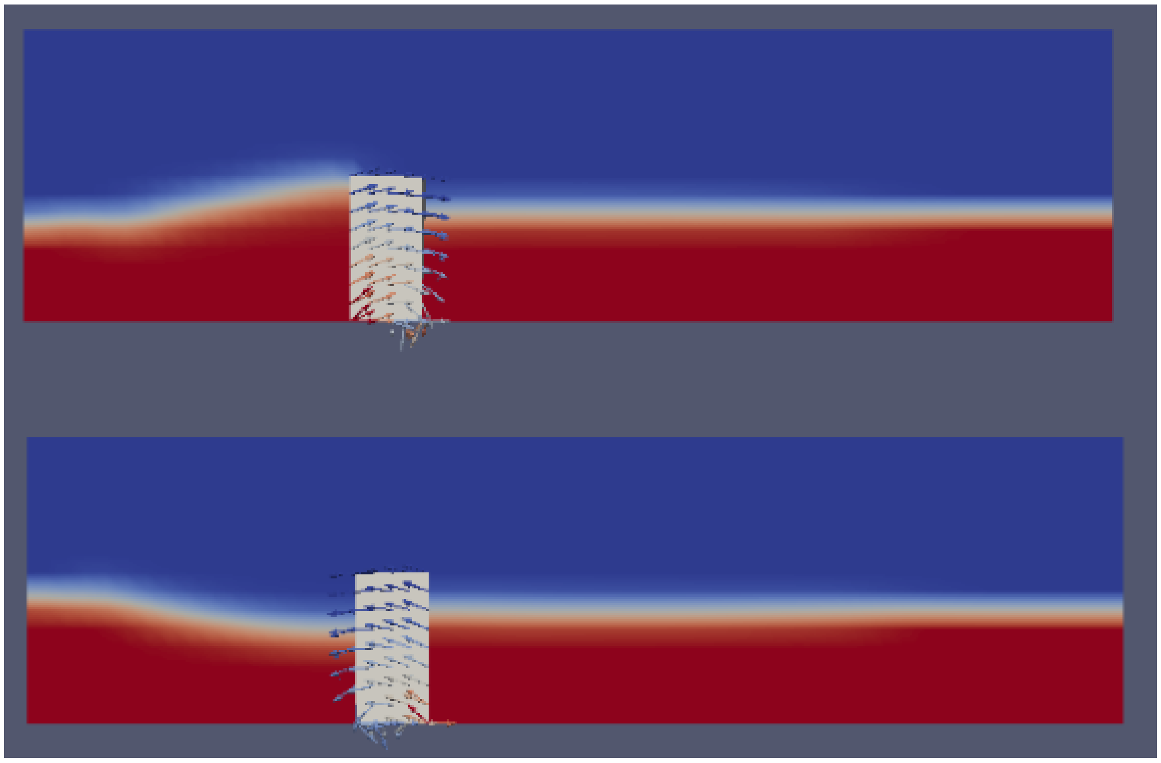

Figure 4.5: The interaction of the beam with incoming waves: when the beam is hit by a wave crest, it deforms forward (above); when the beam is hit by a wave trough, it deforms backward (below).

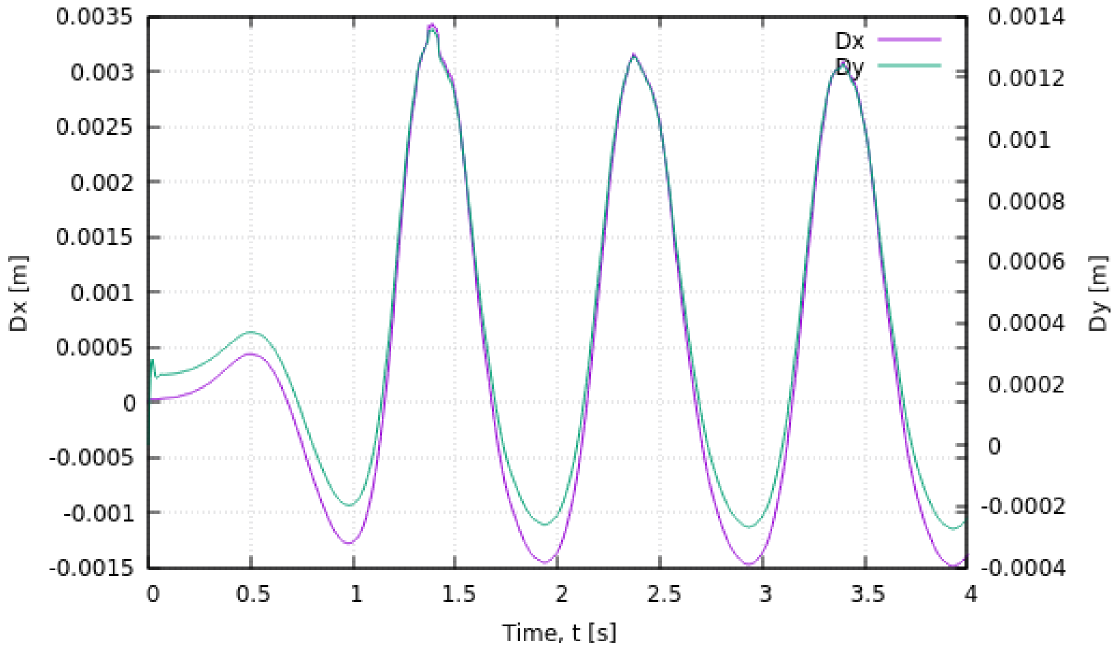

Figure 4.6: Deformation of the beam as a function of runtime

Chapter 5

Conclusion

This project introduces an approach to simulate wave-induced FSI problems upon OpenFOAM. Three opensourse packages, `FSI package, interDyMFoam and waves2Foam`, are coupled together and evolved into a new fluid library `waveInterFluid` and a new solver `waveFsiFoam`. Details are given on relevant theoretical background and the development process.

A tutorial case "beam deformation in regular waves" is provided to demonstrate the usage of the proposed code. Through the simulation of this case, the new code preforms stably with reasonable results obtained, which suggests this code may be promising to simulate similar problems.

The tutorial case is a simple application of this package, but this package should also be valid to more marine FSI cases. On the fluid side, more complex waves can be generated. On the solid side, the structure can be varied.

Study questions

How to use it

- 1 What is the structure of a FSI case?
- 2 How to run a solid solver during a FSI simulation?

The theory of it

- 1 What is the theory used in this work to realize free-surface modelling?
- 2 What is the theory used in this work to facilitate a numerical wave tank?

How it is implemented

- 1 Where should interDyMFoam be implemented into the FSI package?
- 2 How to link waves2Foam package with a FSI solver?

How to modify it

- 1 How to modify wave properties?
- 2 How to modify the deformation properties of the solid and what properties can be modified?

Reference

[1] Tukovic Z, Cardiff P, Karac A, Jasak H, Ivankovic A. OpenFOAM library for fluid structure interaction. In: 9th OpenFOAM Workshop. vol. 2014; 2014. .

[2] Li M. The implementation of interFoam solver as a flow model of the fsiFoam solver for strong fluid-structure interaction. CFD with OpenSource software Course; 2016.

[3] Yao HD. Simulation of Fluid-Structural Interaction using OpenFOAM. Simulation. 2014;(1/37).

[4] Hirt CW, Nichols BD. Volume of fluid (VOF) method for the dynamics of free boundaries. Journal of computational physics. 1981;39(1):201–225.

[5] Jacobsen NG. waves2Foam Manual. 2017;.

[6] Mayer S, Garapon A, Sørensen L, et al. A fractional step method for unsteady free-surface flow with applications to non-linear wave dynamics. International Journal for Numerical Methods in Fluids. 1998;28(2):293–315.

[7] Bruinsma N. Validation and Application of a Fully Nonlinear Numerical Wave Tank. 2016;.