Chalmers University of Technology

CFD with OpenSource software, assignment 3

Tutorial: Mesh motion class of the Vigor Wave Energy Converter

Developed for OpenFOAM-1.6.x

Author:
Mattias Olander

October 29, 2010

Chapter 1

Tutorial Vigor Wave Energy Converter

1.1 Introduction

In this Tutorial a mesh motion class for simulating the movement of the rubber hose in the Vigor Wave Energy Converter will be implemented.

This will be done by editing the `dynamicIncJetFvMesh` class to perform the desired mesh movement resembling a rubber hose riding on a water wave, which can be seen in figure 1.1. This involves using a for loop to generate a sinus function that moves the nodes of a basic rectangular mesh in a way that simulates a traveling wave. It is considered that the rubber hose is fixed at one end. Also, the edited `dynamicInkJetFvmesh` will be used together with the `interDyMFoam` solver to show how movement of the waves moves water inside the rubber hose. This includes information on how to set up a case from an already excisting case and how to implement the boundary condition `timeVaryingUniformFixedValue`, which pumps water and air into the rubber hose. For those who seek more information about the Vigor Wave Energy Converter can download this pdf: `http://www.tfd.chalmers.se/~hani/pdf_files/VigorWaveEnergyConverter.pdf`

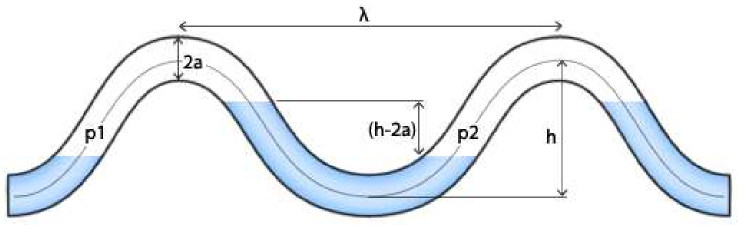


Figure 1.1: The principle of the Vigor Wave Energy Converter; water and air have been inserted in such proportions that water columns are created which produces a pressure difference

1.2 Creating the new class

1.2.1 Copying the dynamicInkJetFvMesh class

First the `dynamicInkJetFvMesh` class is copied to the run directory and renamed to `dynamicVigorWaveFvMesh`. Every word corresponding to `dynamincInkJetFvMesh` has to be changed to `dynamicVigorWaveFvMesh` which is done using the "sed" command. To be able to compile the class, a Make directory has to be copied from the `dynamicFvMesh` directory.

```
>> cp -r $FOAM_SRC/dynamicFvMesh/dynamicInkJetFvMesh/ \
$WM_PROJECT_USER_DIR/run/dynamicVigorWaveFvMesh
>> cd $WM_PROJECT_USER_DIR/run/dynamicVigorWaveFvMesh
>> sed s/dynamicInkJetFvMesh/dynamicVigorWaveFvMesh/g \
<dynamicInkJetFvMesh.C >dynamicVigorWaveFvMesh.C
>> sed s/dynamicInkJetFvMesh/dynamicVigorWaveFvMesh/g \
 <dynamicInkJetFvMesh.H >dynamicVigorWaveFvMesh.H
>> rm dynamicInkJetFvMesh.*
>> cp -r $FOAM_SRC/dynamicFvMesh/Make $WM_PROJECT_USER_DIR/run/dynamicVigorWaveFvMesh
```

The Make directory contains two files, "files" and "options" that need to be edited as stated below.

The file "files" is modified to only include the two lines displayed in *Box* 1 so just the `dynamicVigorWaveFvMesh` class will be compiled:

```
dynamicVigorWaveFvMesh.C

LIB=$(FOAM_USER_LIBBIN)/libdynamicVigorWaveFvMesh
```

Box 1: Code of the files file

In the file "options" the line below is added to include the files from the original library:

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

Box 2: To be included in the options file

The "option" file should now look like this:

```
EXE_INC = \
    -I$(LIB_SRC)/triSurface/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/dynamicMesh/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/dynamicFvMesh/lnInclude

LIB_LIBS = \
    -ltriSurface \
    -lmeshTools \
    -ldynamicMesh \
    -lfiniteVolume
```

Box 3: The code of the option file

The class can now be compiled to make sure that everything has been done correctly.

```
>> cd $WM_PROJECT_USER_DIR/run/dynamicVigorWaveFvMesh
>> wmake libso
```

1.2.2 Modifying the code of the dynamicInkJetFvMesh class

Now it is time to modify the `dynamicVigorWaveFvMesh` files, to make the class perform as wanted. In the `dynamicVigorWaveFvMesh.H` file ”frequency” is changed to ”waveLength” and ”refPlaneX” is changed to ”periodTime”. The ”Private data” should now look like in *Box* 4.

```
        dictionary dynamicMeshCoeffs_;

        scalar amplitude_;
        scalar waveLength_;
        scalar periodTime_;

        pointIOField stationaryPoints_;
```

Box 4: The Private data of the dynamicVigorWaveFvMesh.H file

More important changes will be made in the `dynamicVigorWaveFvMesh.C` class. The motion of a traveling water wave will be simulated using a sinus function, namely equation 1.1.

$$y = tanh(x/4) * Asin(2\pi x/\lambda + 2\pi t/T) \tag{1.1}$$

Here, A = amplitude, λ = wavelength and T = ”the time for one complete oscillation”. The ”tanh” function is there to ensure that the fixed boundary condition of the left wall is enforced. It goes to 1 as x goes to infinity and it is divided by 4 to make it go slower to 1. It is really a matter of deciding how ”strong” the rubber hose is and how much it stretches by the fixed wall. The ”+” in the sinus function, makes sure that the wave moves from the right to the left.

In the `dynamicVigorWaveFvMesh.C` file, ”frequency” is changed to ”waveLength” and ”refPlaneX” is changed to ”periodTime” everywhere. Then, the bool function below the headline, ”Member Funcions”, is modified to implement the traveling wave equation.

```
bool Foam::dynamicVigorWaveFvMesh::update()
{

    pointField newPoints = stationaryPoints_;
    pointField toZero = stationaryPoints_;

    forAll(newPoints,i)
    {
toZero[i][0]=toZero[i][0]-stationaryPoints_[1][0];

newPoints[i][1]=stationaryPoints_[i][1]+tanh(toZero[i][0]/4)*amplitude_*
sin(2*mathematicalConstant::pi*toZero[i][0]/waveLength_+
2*mathematicalConstant::pi*time().value()/periodTime_);
    }

    fvMesh::movePoints(newPoints);

    volVectorField& U =
        const_cast<volVectorField&>(lookupObject<volVectorField>("U"));
    U.correctBoundaryConditions();

    return true;
}
```

Box 5: The bool function of the dynamicVigorWaveFvMesh.C file

This means that instead of using the `newPointsreplace` function, which the author knowledge about is very limited, a for loop is used to step through every node of the domain, updating the y-coordinate for each timestep with the traveling wave equation. The update is done to `newPoints[i][1]`, which is the new position of y, by adding the value of the traveling wave equation to the stationary position of y. The `toZero` field is used to translate any mesh that does not have the left wall at the origin, as the travel wave function expects the left wall to be located at x=0. This is why the x-value of `toZero` is used as this ensures that the traveling wave equation works properly.

The class can now be compiled again.

```
>> cd $WM_PROJECT_USER_DIR/run/dynamicVigorWaveFvMesh
>> wmake libso
```

In order to demonstrate that the new class works, a simple case will be set up.

1.3 Setting up a simple case

To simulate the multiphase system of water and air in the moving rubber hose, the `interDyMFoam` solver will be used. A tutorial case, `sloshingTank2D`, which uses the `interDyMFoam` solver and therefore is suitable to this case is copied into the run folder.

```
>> cp -r $FOAM_TUTORIALS/multiphase/interDyMFoam/ras/sloshingTank2D \
$WM_PROJECT_USER_DIR/run/VigorWave
>> cd $WM_PROJECT_USER_DIR/run/VigorWave
```

The Allclean and Allmake files can be removed since they will not be needed.

1.3.1 The mesh

Now the tutorial case will be altered to satisfy the new requirements. Firstly, the `blockmeshDict` file will be changed to produce the desirable geometry. It can be located in the `contant/polymesh` dictionary and it should, when updated, look like *Box* 6. This will create a simple thin 2D rectangular mesh, with just a few cells to save computer capacity. In reality, the rubber hose is supposed to be 250 meter but here the case is just set up to prove that the motion mesh class works, so therefore the mesh is set to 15 meters long and 0.6 meters thick.

```
convertToMeters 10;

vertices
(
    (0 0.06 0)
    (0 0 0)
    (1.5 0 0)
    (1.5 0.06 0)

    (0 0.06 0.01)
    (0 0 0.01)
    (1.5 0 0.01)
    (1.5 0.06 0.01)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (4 100 1) simpleGrading (1 1 1)
);

edges
(
);

patches
(
    wall movingWall
    (
(0 4 7 3)
(1 2 6 5)
    )

    patch inlet
    (
(3 7 6 2)
    )

    patch outlet
    (
        (1 5 4 0)
    )
    empty frontAndBack
    (
        (0 3 2 1)
        (4 5 6 7)
    )
);

mergePatchPairs
(
);
```

Box 6: The code of the blockmeshDict file

When the `blockmeshDict` file has been updated, the mesh can be created using blockMesh. It is always recommended to run checkMesh to make sure that the mesh is ok for the simulation.

```
>> blockMesh
>> checkMesh
```

1.3.2 Boundary conditions

The boundary conditions must now be changed so that they correspond to the patches of the new mesh. In the 0 dictionary the three files `alpha.org`, `p` and `U` are found. In the file `alpha.org` it can be specified if the fluid should be air or water, 0 for air and 1 for water. It need to be copied and renamed to `alpha1` before running the simulation.

```
>> cd $WM_PROJECT_USER_DIR/run/VigorWave/0
>> cp alpha1.org alpha1
```

The `p` and `U` files will use pretty basic boundary conditions and how they should look can be seen below. Since the `movingWall` will move, the `movingWallVelocity` is suitable for `U` and `bouyantPressure` for `p`. For the inlet a velocity speed of 10 m/s is specifed in the negative x-direction to allow inflow.

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

internalField   uniform (0 0 0);

boundaryField
{
    movingWall
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }

    inlet
    {
        type           fixedValue;
        value          uniform (-10 0 0);
    }

    outlet
    {
        type            pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }

    frontAndBack
    {
        type            empty;
    }
}
```

Box 7: The boundary conditions of U

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

internalField   uniform 0;

boundaryField
{
    movingWall
    {
        type            buoyantPressure;
        value           uniform 0;
    }

    inlet
    {
        type            zeroGradient;

    }

    outlet
    {
        type            totalPressure;
        p0              uniform 0;
        U               U;
        phi             phi;
        rho             rho;
        psi             none;
        gamma           1;
        value           uniform 0;
    }

    frontAndBack
    {
        type            empty;
    }

}
```

Box 8: The boundary conditions of p

To simulate a unsteady flow of water, the `timeVaryingUniformFixedValue` boundary condition will be used for the inlet in the alpha file. As it uses the `interpolationTable` class, it need as input data a file that holds information about which value alpha is going to have at a specific timestep. This file should be put in the main directory of the case. Also, a `outOfBounds` string need to be specified, which decides what to do when the simulaton has passed the scope of the created timestep file. It can either be `error`, `warn`, `clamp` or `repeat`. For example, `clamp` will just continue with the last given value for the rest of the timesteps and `repeat` will just loop over the specified values. More information can be found in the `interpolationTable.H` file located at `$FOAM_SRC/OpenFOAM/interpolations/interpolationTable`.

So, a new file named "timesteps" is created in the main directory of the case and inside it the values of `alpha` at certain timesteps are specified. It should look like in *Box* 9, meaning alpha equals 1 from timestep 0.0 to timestep 0.5 where it changes to 0. As `repeat` will be used for `outOfBounds`, nothing more than this needs to be specified.

```
(
 (0.0 1)
 (0.5 0)
)
```

Box 9: The timestep file

Now the `alpha1` file can be altered to include `timeVaryingUniformFixedValue` for the inlet and `inletOutlet` for the outlet. The resulting alpha file is shown below.

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

internalField   uniform 0;

boundaryField
{
    movingWall
    {
        type            zeroGradient;
    }
    inlet
    {
        type            timeVaryingUniformFixedValue;
        fileName        "timesteps";
        outOfBounds     repeat;
    }
    outlet
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }
    frontAndBack
    {
        type            empty;
    }
}
```

Box 10: The boundary conditions for alpha

Now, when all the boundary condition have been changed, the mesh can be viewed using `paraFoam`.

1.3.3 The constant and system directory

In the `constant` directory, the `dynamicMeshDict` needs to call the new class `dynamicVigorWaveFvMesh` and the `amplitude`, the `waveLength` and the `periodTime` of the wave need to be specified. The `dynamicMeshDict` file should therefore look like this:

```
dynamicFvMeshLibs ("libdynamicVigorWaveFvMesh.so");

dynamicFvMesh  dynamicVigorWaveFvMesh;

dynamicVigorWaveFvMeshCoeffs
{
amplitude 1;
waveLength 8;
periodTime 1;
}
```

Box 11: The code of the dynamicMeshDict file

Still, in the `constant` directory the `g` file need to be altered so that the gravitation is in the negative y-direction instead of the negative z-direction. Nothing more needs to be changed as the coefficients of water and air is already specified in `transportProperties` and laminar flow is set in the `turbulentProperties`. Actually, the `RASProperties` file can even be removed, as the flow is set to laminar.

From the `constant` directory the focus is turned to the `system` directory, where a couple of files that needs modification is located. The `setFieldsDict` file can be removed since it will not be used in this case. Open the `controlDict` file and change the "endtime" value to 10 and the "deltaT" to 0.05. Also, change the `writeCompression` from "compressed" to "uncompressed" as the files in the new case is uncompressed only. The functions formula located after `maxDelta` can be deleted in the file as it is of no interest to write out probes and wallPressures at this point. The `controlDict` file should finaly look like below when finished.

```
application     interDyMFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         10;

deltaT          0.05;

writeControl    adjustableRunTime;

writeInterval   0.05;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable yes;

adjustTimeStep  yes;

maxCo           0.5;

maxDeltaT       1;
```

Box 12: The code of the controlDict file

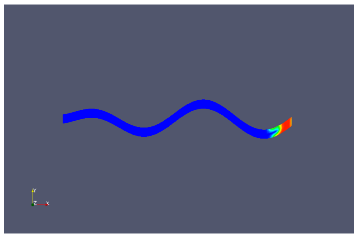
The `decomposeParDict` file is used if the simulation should be run on more than one core. Typically the `numberOfSubdomains` is put to the number of cores the computer have and the method of decomposition is chosen to, for example, ”simple”. If the computer only have one core, this file will be skipped.

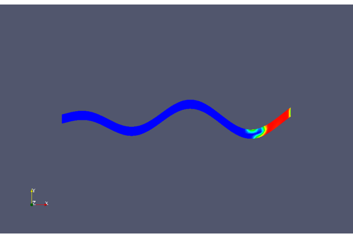
The last thing that has to be done is to check if the reference point for the pressure, which is defined in the `fvSolution` file, is defined insided the domain. By opening the `fvSolution` file and scrolling down to where the coefficients of the `PISO` loop are defined, the `pRefPoint` is found to be (0 0 0.15). That is a point inside the domain, so nothing has to be changed.

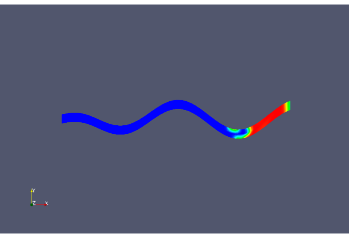
Run the case with `interDyMFoam`. The calculation will take approximately 1.5 minutes on a Intel Core 2 Duo CPU T5750 2.00GHz with 3GB ram.

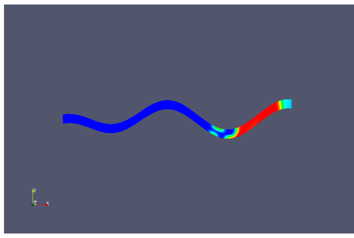
1.4 Postprocessing of the case

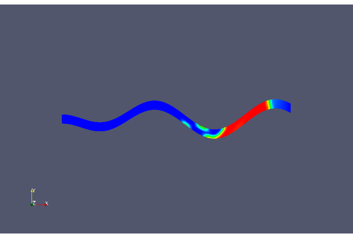
Here Paraview has been used to produce a few pictures of the alpha value, e.g the water distribution, in the simulation at different timesteps. The pictures is shown just to help the user verify that the simulation is correct and that the mesh motion class works. As seen by the figures the wave moves from the right to the left and in the meantime the water is entering the rubber hose at the right end. Also, note that the wave is slowly vanishing as it approaches the fixed left end.

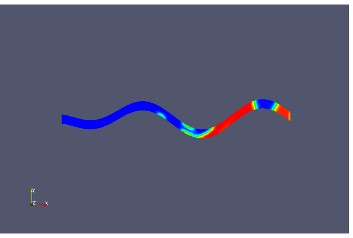

(a) t=0.1


(b) t=0.2


(c) t=0.3


(a) t=0.4


(b) t=0.5


(c) t=0.6