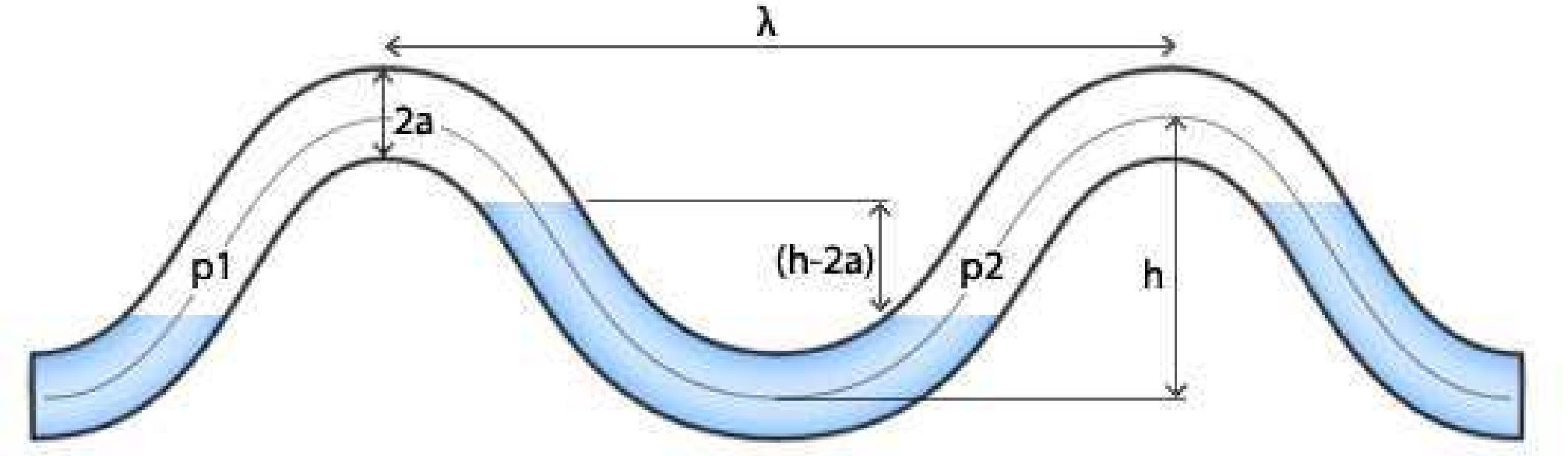
Tutorial Vigor Wave Energy Converter

- In this Tutorial a mesh motion class for simulating the Vigor Wave Energy Converter will be implemented for OF-1.6.x.
- How to implement the boundary condition `timeVaryingUniformFixedValue` is also explained.
- In the Vigor Wave Energy Converter a rubber hose with water and air is used to harvest energy created by water waves.



Creating the new class

- Copy the `dynamicInkJetFvMesh` class to the run directory and rename it.

```
>> 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.*
```

- Also copy the Make directory

```
>> cp -r $FOAM_SRC/dynamicFvMesh/Make \
$WM_PROJECT_USER_DIR/run/dynamicVigorWaveFvMesh
```

Creating the new class

- Update the ”files” file to the following:

```
dynamicVigorWaveFvMesh.C

LIB=$(FOAM_USER_LIBBIN)/libdynamicVigowWaveFvMesh
```

- Add this to the ”option” file:

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

- Compile it to make sure that everything has been done correctly.

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

Creating the new class

- In the `dynamicVigorWaveFvMesh.H` file change ”frequency” to ”waveLength” and ”refPlaneX” to ”periodTime”.
- Do the same in the `dynamicVigorWaveFvMesh.C` file.
- Also modify the bool function under ”Member Funcions” to this:

```
    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_);
    }
```

Creating the new class

- So, instead of a `newPointsreplace` function, a for loop steps through the domain of the mesh and updates it according to the sinus equation below:

$$y = tanh(x/4) * Asin(2\pi x/\lambda + 2\pi t/T) \quad (1)$$

- 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.
- The `toZero` field is used to translate the mesh to the origin.

Creating the new class

- Copy the `sloshingTank2D` tutorial case.

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

- First, the blockMeshDict file is changed to produce a thin rectangular domain:

```
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
(
);
```

Creating the new class

- Create the mesh and then check it.

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

- If ok, continue with the boundary conditions, starting with U:

```
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;
}
```

Creating the new class

- The boundary conditions for p is also changed:

```
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;
}
```

Creating the new class

- To simulate a unsteady flow of water, the `timeVaryingUniformFixedValue` boundary condition will be used for alpha.
- As indata it needs a file that holds information about which value alpha is going to have at a specific timestep.
- 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 the values specified.

Creating the new class

- A new file named ”timesteps” is created in the main directory of the case and inside it the values of `alpha` at timesteps is specified.

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

- This means that alpha equals 1 from timestep 0.0 to timestep 0.5 where it changes to 0. As `repeat` will be used for `outOfBounds`, no more than this need to be specified.

Creating the new class

- The new boundary conditions for alpha is shown below:

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

Creating the new class

- In the `constant` directory, the `dynamicMeshDict` need to call the new class `dynamicVigorWaveFvMesh` and the `amplitude`, the `waveLength` and the `periodTime` of the wave need to be specified:

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

dynamicFvMesh  dynamicVigorWaveFvMesh;

motionSolverLibs  ("libfvMotionSolvers.so");


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

Creating the new class

- In the `constant` directory the g file need to be altered so the gravitation is in the negative y-direction instead of the negative z-direction.
- The coefficients of water and air is already specified in `transportProperties`, so nothing need to be changed.
- In the `controlDict` file in the `system` directory, change the endtime value to 10, the deltaT to 0.05 and `writeCompression` from compressed to uncompressed.
- The functions formula located after `maxDelta` can be deleted in the file as its of no interest to write out probes and wallPressures at this point.
- Run the case with `interDyMFoam`.

Finished!

THANK YOU!