Anders Rynell

CFD with OpenSource Software
OpenFOAM 1.5.dev
Tutorial of interTrackFoam-solver
Peer Reviewed by Anton Berce and Jelena Andric

Department of Applied Mechanics
Division of Fluid Dynamics
Chalmers University of Technology
Göteborg Sweden, 2010

Contents

Chapter 1

introduction

This tutorial will describe the solver *interTrackFoam* which is a part of the OpenFOAM 1.5.dev version. The solver is an incompressible, transient solver which uses a free surface tracking algorithm. What this really means will be explained as well as the different choices considering the settings for a modified case *modDamBreak*.
First as always when it comes to computational fluid dynamics the domain has to be discretized into small cells where the properties solved for can be computed. The equation is solved for each timestep and the computational mesh is adjusted to the shape of the free surface. This means that the mesh can not be pre-defined. All the internal points depend strongly on the motion of the free surface and will move accordingly to the chosen mesh solver. In order to get reliable results the mesh quality has to be maintained during the calculations.

Chapter 2

hydrofoil

The solver *interTrackFoam* is used when solving the case *hydrofoil* which consists of an inclined airfoil situated below a free surface in water. The pressure distribution around the profile causes waves to propagate on the water surface. The mesh in turn is adapted to the free surface motion and will adjust to it. The inlet height is constrained to represent constant inlet mass flow. The whole domain is shown in Figure 2.1.

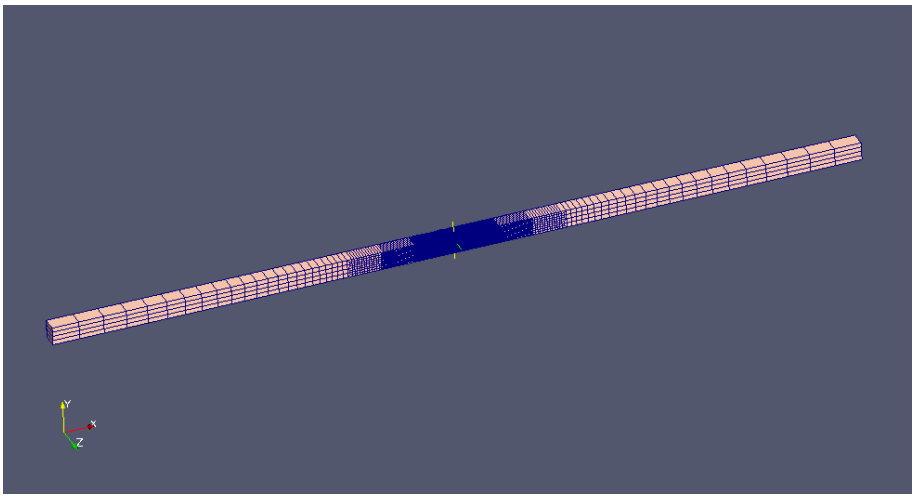

Figure 2.1: The whole *hydrofoil* domain

A closer look at the *hydrofoil* situated in the refinement mesh area is shown in Figure 2.2. As can be seen in the figures, *hydrofoil* consists of thousands of cells. In order to show the different settings, a different geometry is used, otherwise the computational time will be unacceptable long. The geometry from *damBreak* was chosen since it also includes an obstacle, which will

cause a pressure field to propagate, see Figure 2.3. This in turn will cause the free surface to move.

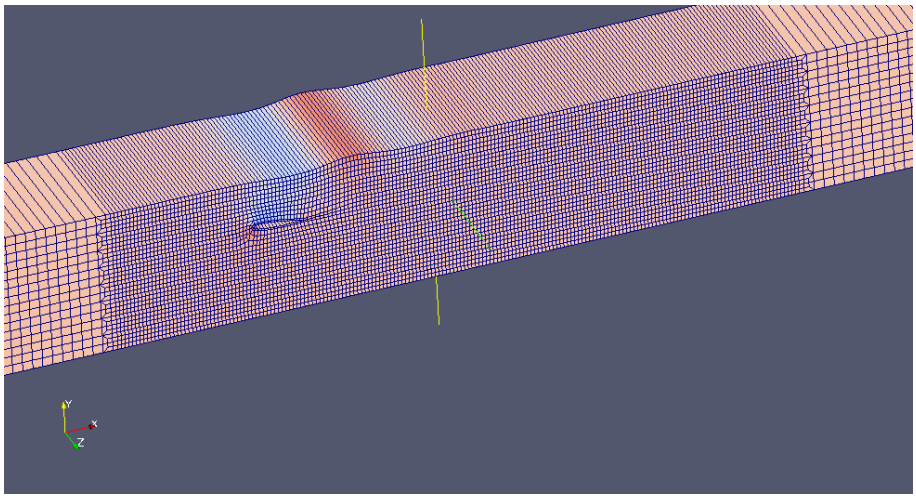

Figure 2.2: A close-up of the *hydrofoil*

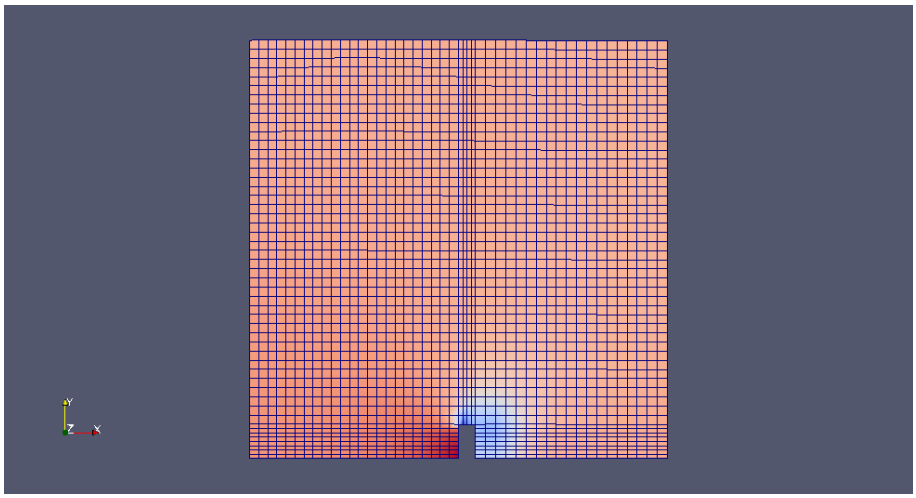

Figure 2.3: The modDamBreak domain

Chapter 3

domainSettings

In order to be able to use the solver *interTrackFoam* on *damBreak* some modifications in the files are necessary. Since the only thing that will change is the domain, the *polyMesh* folder from *damBreak* will be copied into the *hydrofoil* folder. Then *hydrofoil* is simply renamed *modDamBreak*. The *boundary* file is also deleted, since it can cause problem when using *blockMesh*. All this is done by typing;

```
run
cp -r $FOAM_TUTORIALS/interTrackFoam/hydrofoil .
rm -rf hydrofoil/constant/polyMesh
cp -r $FOAM_TUTORIALS/interFoam/damBreak/constant/\
polyMesh hydrofoil/constant/polyMesh
mv hydrofoil modDamBreak
rm -rf modDamBreak/constant/polyMesh/boundary
```

Now it remains to change the settings for the modified case. Open the blockMeshDict by typing;

```
gedit constant/polyMesh/blockMeshDict
```

3.1 blockMesh

In *blockMeshDict* file, the patches is changed according to;

```
patches
(
    patch left
    (
        (0 12 16 4)
```

```
        (4 16 20 8)
    )
    patch right
    (
        (7 19 15 3)
        (11 23 19 7)
    )
    wall bottom
    (
        (0 1 13 12)
        (1 5 17 13)
        (5 6 18 17)
        (2 14 18 6)
        (2 3 15 14)
    )
    patch freeSurface
    (
        (8 20 21 9)
        (9 21 22 10)
        (10 22 23 11)
    )
    empty frontAndBackPlanes
    (
(0 1 5 4)
(4 5 9 8)
(5 6 10 9)
(2 3 7 6)
(6 7 11 10)
(12 13 17 16)
(16 17 21 20)
(17 18 22 21)
(14 15 19 18)
(18 19 23 22)
    )
);

mergePatchPairs
(
);
```

The mesh is constructed and the different boundaries are renamed. It is important that the free surface is named $freeSurface$ otherwise the solver later on will not be able to track it. No geometrical changes are made but the left and right sides of the domain are no longer of type "wall" but "patch". Also the $frontAndBackPlanes$ is added and is essential for the finite-area disretization, see $faMeshDefinition$.

3.2 boundary Conditions

Also the boundary conditions have to be modified for the velocity, pressure and displacements. It is important that the names used in $blockMeshDict$, i.e. the names for each patch are specified also in the $motionU$, p and U. Since the files have been copied from $hydrofoil$ it is important that the boundary conditions for the $hydrofoil$ is removed and the "inlet" and "outlet" are replaced by "left" and "right". Open the file $motionU$ to change its corresponding boundary conditions;

```
gedit 0/motionU
```

Change the $boundaryField$ settings in $motionU$ to

```
boundaryField
{
    bottom
    {
        type fixedValue;
        value uniform (0 0 0);
    }

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

    left
    {
        type slip;
    }

    right
```

```
    {
        type slip;
    }

    frontAndBackPlanes
    {
        type empty;
    }
}
```

Open the file p to change its corresponding boundary conditions

```
gedit 0/p
```

Change the $boundaryField$ settings in p to

```
boundaryField
{
    bottom
    {
        type zeroGradient;
    }

    freeSurface
    {
        type fixedValue;
        value uniform 0.0;
    }

    right
    {
        type zeroGradient;
    }

    left
    {
        type zeroGradient;
    }

    frontAndBackPlanes
    {
        type empty;
```

```
    }
}
```

Open the file U to change its corresponding boundary conditions

```
gedit 0/U
```

Change the $boundaryField$ settings in U to

```
boundaryField
{
    bottom
    {
        type slip;
    }

    freeSurface
    {
        type fixedGradient;
        gradient uniform (0 0 0);
    }

    right
    {
        type zeroGradient;
    }

    left
    {
        type fixedValue;
        value uniform (0.8 0 0);
    }

    frontAndBackPlanes
    {
        type empty;
    }
}
```

The finite area discretization of the $freeSurface$ also needs some settings, therefore change the $boundary$ in $faMeshDefinition$. $faMeshDefinition$ can be found in the $faMesh$ folder located in the *constant* directory.

```
boundary
{
    left
    {
        type                patch;
        ownerPolyPatch      freeSurface;
        neighbourPolyPatch  left;
    }

    right
    {
        type                patch;
        ownerPolyPatch      freeSurface;
        neighbourPolyPatch  right;
    }

    frontAndBackPlanes
    {
        type                empty;
        ownerPolyPatch      freeSurface;
        neighbourPolyPatch  frontAndBackPlanes;
    }
}
```

Open $freeSurfaceProperties$ and change the $fixedFreeSurfacePatches1(inlet)$ to $fixedFreeSurfacePatches1(left)$;. The modified damBreak can now be used. Type $blockMesh$, $makeFaMesh$ and $interTrackFoam$. In order to save time, modify the *Allrun* file so it looks like

```
#!/bin/sh

. $WM_PROJECT_DIR/bin/tools/RunFunctions


application="interTrackFoam"

runApplication blockMesh
runApplication makeFaMesh
runApplication $application
```

Despite the fact that the domain has been changed, the computational is still time consuming. The constrain on the $freeSurface$ curvature (no mass flux) makes the computation sensitive to large displacements.

Chapter 4

modDamBreak

All OpenFOAM cases contains at its minimum three directories in order to run the application. This directories are 0, *constant* and *system*. This is where the settings are made. A primary explanation of these are now presented.

4.1 0

In the time directory, 0;, the initial conditions are specified for the properties solved for, namely

```
motionU  p  U
```

where *motionU*,*p* and *U* are the displacements, pressure and velocity, respectively. The settings for each has already been setup and can be visualized in Section 3.2. The settings for the hydrofoil case differs from the ones presented here but the topology looks the same. Information of the various settings can be found at the [1].
Dependent on the settings which are made in *controlDict*, more files with results will be written and placed here. The names for each is based on the simulated time at which the data is written.

4.2 constant

Constant directory contains the following;

```
dynamicMeshDict  faMesh  freeSurfaceProperties  polyMesh
transportProperties
```

In short, in the *constant* directory the mesh settings are made, both the geometry and the settings which correspond to the mesh and its motion during the solving process. A further look into these settings is presented below.

4.2.1 dynamicMeshDict

```
twoDMotion          yes;

solver              laplaceFaceDecomposition;

diffusivity         patchEnhanced;

distancePatches     1 (freeSurface);

frozenDiffusion     yes;


pseudoSolid
{
    poissonsRatio           0.3;
    nCorrectors             3;
    convergenceTolerance    1e-9;
};
```

The various choices of settings in this files affect the characteristics of the mesh. If $twoDMotion$ is chosen only two dimensional movements are allowed and $twoDPointCorrectors$[2] is used during the calculations. $twoDPoint-Correctors$ prevents the mesh from twisting. It remains to determine the motion of the mesh points. The *solver*-types available are $laplaceFaceDecomposition$, $pseudoSolidFaceDecomposition$ and $RBFMotionSolver$. The $laplaceFaceDecomposition$ uses a Laplacian equation to solve the motion of the mesh points emanating from the motion of the freeSurface. Since the magnitude of the point motion is not uniformly distributed through the whole domain, the diffusivity type chosen will help to conserve the mesh quality. One advantageous way is to confine the largest deformation to the internal part of the mesh. Table 4.1 shows the various diffusivity options.
In the quality-based methods, the diffusion field is a function of the cell quality measurements while the distance-based methods uses the distance to the nearest boundary, specified by *distancePatches* [3]. The other option $file$ uses a diffusivity field specified in a file. Since the free surface mo-

diffusivity	type
distancebased	*linear*
	quadratic
	exponental
	patchEnhanced
qualitybased	*uniform*
	distortionEnergy
	deformationEnergy
other	*file*

Table 4.1: The diffusivity options

tion in this case is very smooth and the fact that the mesh is coarse, the choice of diffusivity type will not affect the solution. An example where the diffusivity options makes a huge difference can be illustrated in [4]. *frozenDiffusion* means that the initial diffusion rate is "frozen" and does not change during the calculation process. The main difference between the *pseudoSolidFaceDecomposition* and
laplaceFaceDecomposition is that the former also deals with rotation. Since the pseudo-solid solver requires a greater storage because of the coupled motion vector components, it is a matter of computational effort. If the mesh quality is not substantially improved its choice as a solver, can not be justified. Figure 4.2 illustrates the mesh rotation obtained from solving the pseudo-solid equation while Figure 4.1 is the laplacian.
The *RBFMotionSolver*[5] uses a different approach. It uses radial basis function (therof the abbreviation RBF) to interpolate the motion of the moving boundary. "A radial basis function (RBF) is a real-valued function whose value depends only on the distance from the origin" [6]. A subset of controlpoints along the *freeSurface* are taken, which are used in the interpolation process. This is an algebraic formulation compared to the partial equations of laplacian and pseudo-solid type. For more information about its settings, a deeper look into the solver *icoDyMFoam*[7] is an alternative.

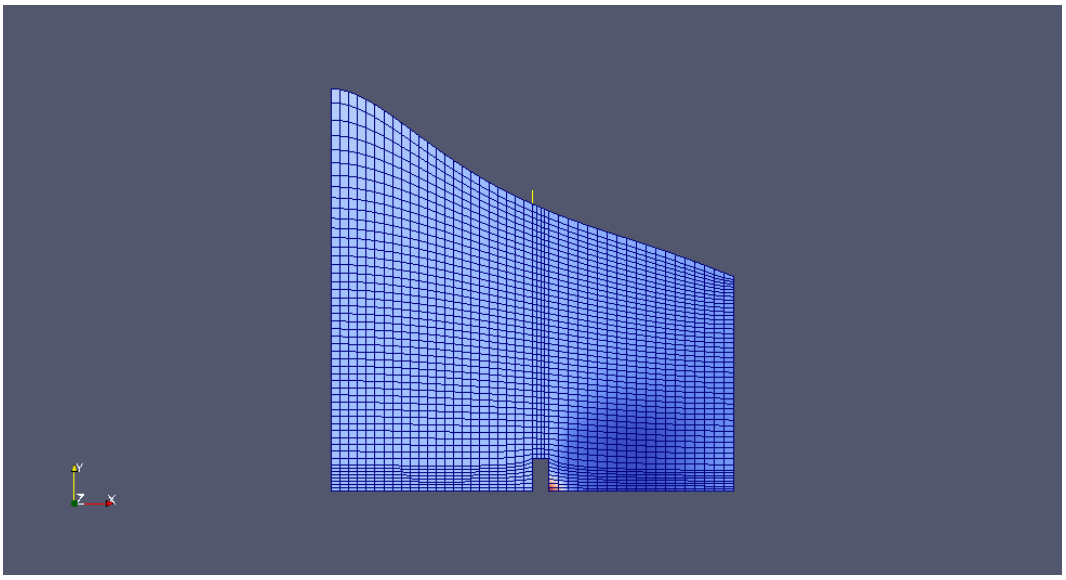

Figure 4.1: Laplace mesh motion solver

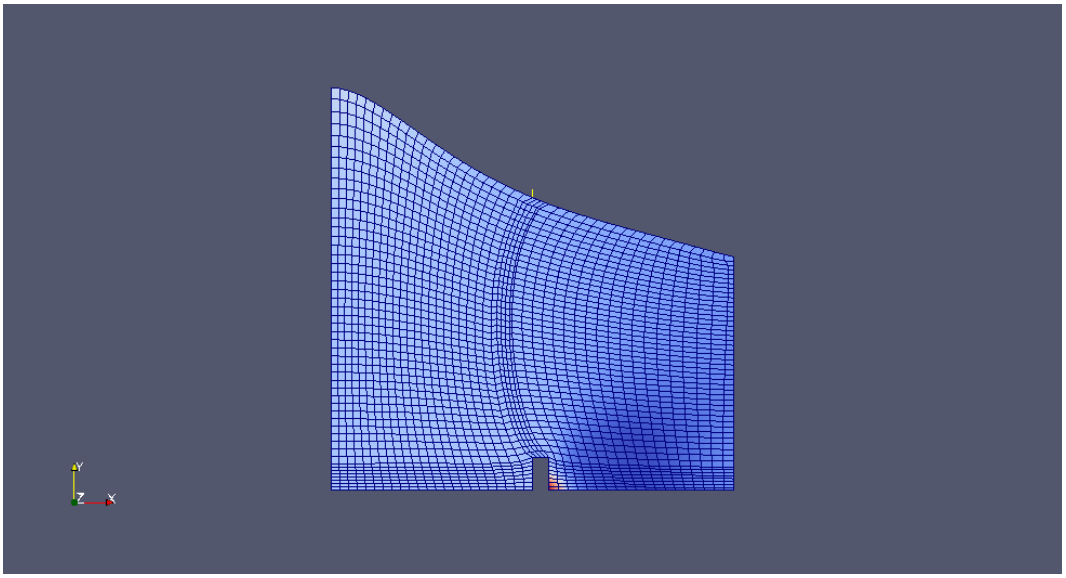

Figure 4.2: Pseudo-solid mesh motion solver

4.2.2 faMesh

The folder $faMesh$ consists of

`boundary.gz` `faceLabels.gz` `faMeshDefinition`

The two former are compressed files which simply describes the $freeSurface$ and are constructed when executing $makeFaMesh$. The conditions for the edges of $freeSurface$ are listed in $faMeshDefinition$.

```
polyMeshPatches  1( freeSurface );

boundary
{
    left
    {
        type                patch;
        ownerPolyPatch      freeSurface;
        neighbourPolyPatch  left;
    }

    right
    {
        type                patch;
        ownerPolyPatch      freeSurface;
        neighbourPolyPatch  right;
    }

    frontAndBackPlanes
    {
        type                empty;
        ownerPolyPatch      freeSurface;
        neighbourPolyPatch  frontAndBackPlanes;
    }
}
```

Only one polyMeshPatch is chosen, namely the $freeSurface$. As an example the first definition means that the condition for the inlet edge is of type *patch*, belongs to the $freeSurface$ and is neighbour with the *inlet*. In the same way the three other edges are specified.

4.2.3 freeSurfaceProperties

The $freeSurfaceProperties$ defines the conditions at the $freeSurface$. Some of the settings involve the nature of surfactants, therefore a brief explanation of what surfactants are and its given equations are given.

```
twoFluids               no;

normalMotionDir         no;

freeSurfaceSmoothing    no;

cleanInterface          yes;

muFluidA                muFluidA    [ 1 -1 -1 0 0 0 0 ]     0;

muFluidB                muFluidB    [ 1 -1 -1 0 0 0 0 ]     1.5e-5;

rhoFluidA               rhoFluidA   [ 1 -3  0 0 0 0 0 ]     1000.0;

rhoFluidB               rhoFluidB   [ 1 -3  0 0 0 0 0 ]     1.0;

surfaceTension          surfaceTension [ 1 -2 0 0 0 0 0 ] 0.0;

g                       g [ 0 1 -2 0 0 0 0 ] (0 -9.81 0);

fixedFreeSurfacePatches 1 ( inlet );

surfactantProperties
{
    bulkConc            bulkConc            [ 0 -3  0 0  1 0 0 ] 1.0e-2;

    saturatedConc       saturatedSurfConc   [ 0 -2  0 0  1 0 0 ] 5.0e-6;

    adsorptionCoeff     adsorptionCoeff     [ 0  3 -1 0 -1 0 0 ] 40.0;

    desorptionCoeff     desorptionCoeff     [ 0 -3  0 0  1 0 0 ] 8.93e-2;

    bulkDiffusion       bulkDiffusion       [ 0  2 -1 0  0 0 0 ] 1.0e-9;

    diffusion           diffusion           [ 0  2 -1 0  0 0 0 ] 1.0e-9;

    temperature         temperature         [ 0  0  0 1  0 0 0 ] 293.0;
}
```

From the top to the bottom, *twoFluids* means the possibility to simulate two fluids and their interaction along the $freeSurface$. In order to do so, a

separate mesh has to be created for each fluid and proper boundary, which bounds the fluids[8] is needed. Also the properties corresponding for each fluid has to be specified, as in Table 4.2.

muFluidA	muFluidA	[1 -1 -1 0 0 0 0]	0
muFluidB	muFluidB	[1 -1 -1 0 0 0 0]	1.5e-5
rhoFluidA	rhoFluidA	[1 -3 0 0 0 0 0]	1000.0
rhoFluidB	rhoFluidB	[1 -3 0 0 0 0 0]	1.0

Table 4.2: Viscosity and density for each fluid

$normalMotionDir$ deals with the mesh motion, and if used, the discretized cells are free to move, both horizontally and vertically. Figure 4.3 is without normal motion direction, i.e. motion is only allowed in the vertical direction while Figure 4.4 shows the case where motion in both direction is allowed.

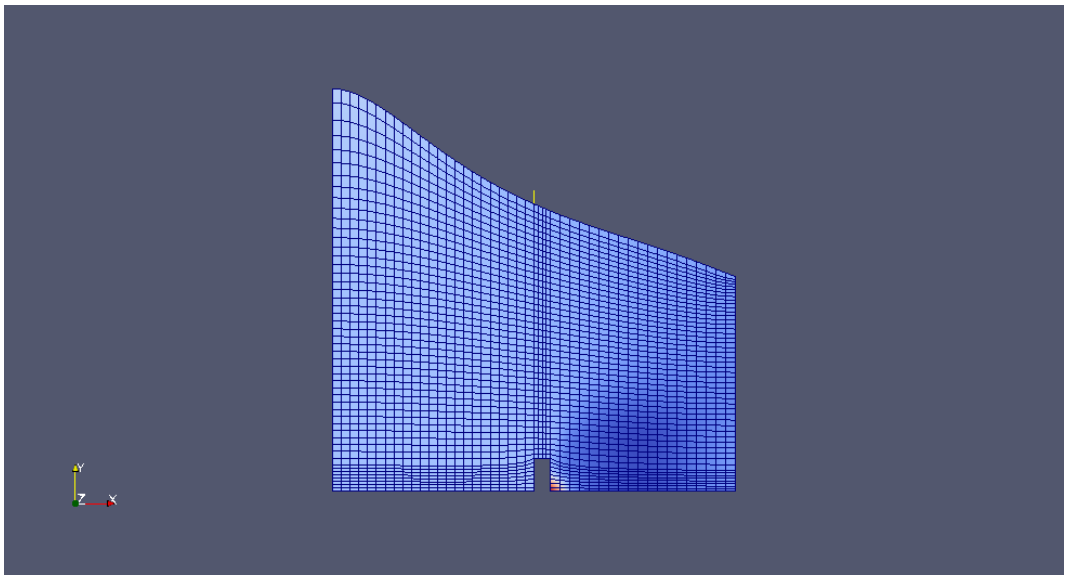

Figure 4.3: Original mesh motion

$freeSurfaceSmoothing$, smooth the $freeSurface$ curvature, taking surface tension into account. One of the most important settings is the $cleanInterface$ option. If the free surface is not chemically clean, surfactant (surface active agents) chemicals concentrate on the free surface. Surfactants are compounds that alter the interfacial tension and are convected and diffused in the bulk fluid, as well as at the interface. This, in turn, will influence the local surface

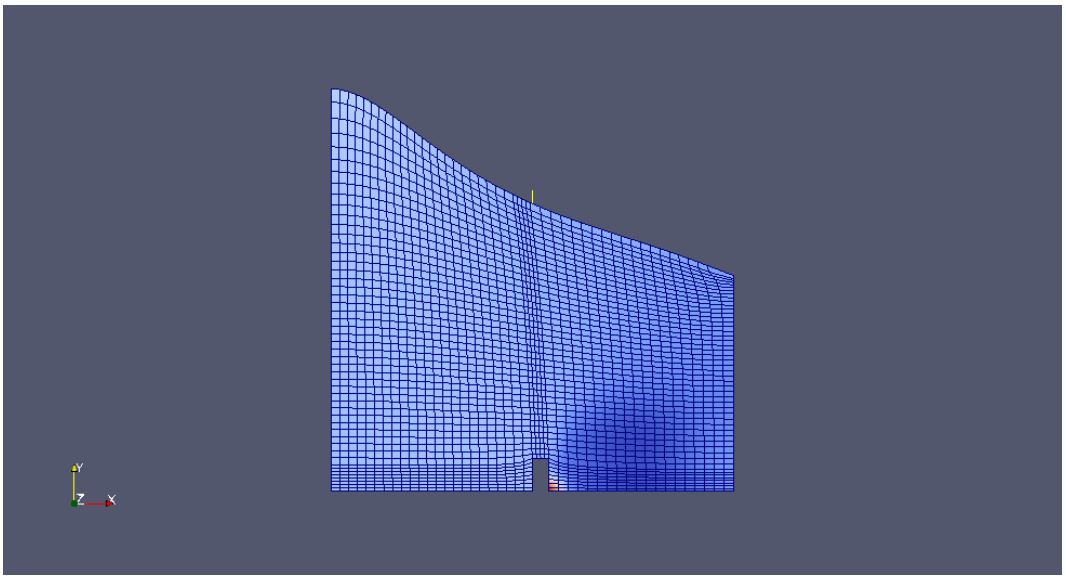

Figure 4.4: $normalMotionDir$ is used

tension and significantly influence the behaviour of the system. The concentration on the free surface acts as a boundary condition for the volume transport. The surfactant properties are shown in Table 4.3.

bulkConc	Φ / C	bulk surfactant concentration
saturatedConc	Φ_∞	saturated surfactant concentration
absorptionCoeff	k_a	parameter of absorption kinetics
desorptionCoeff	β_a	parameter of absorption kinetics
bulkDiffusion	D	bulk surfactant diffusion coefficient
diffusion	D_s	surfactant diffusion coefficient along the surface
Temperature	T	Temperature

Table 4.3: Surfactant properties

The *dynamic condition*, which states that the forces acting on the interface are in equilibrium depends on the surface tension coefficient. This coefficient is calculated using

$$\sigma = \sigma_0 + RT\Phi_\infty ln(1 - \frac{\Phi}{\Phi_\infty}) \tag{4.1}$$

where σ_0 is the surface tension of a clean surface ($surfaceTension$ given above), R is the universal gas constant and the other parameters are presented in Table 4.3. The transport of surfactant in the bulk fluid is given

by

$$\frac{d}{dt}\int_V CdV + \oint_S \mathbf{n}\cdot(\mathbf{v}-\mathbf{v_s})CdS = \oint_S \mathbf{n}\cdot(D\nabla C)dS \tag{4.2}$$

Transport of surfactant along an arbitrary surface

$$\frac{d}{dt}\int_S \Phi dS + \oint_{\partial S} \mathbf{m}\cdot(\mathbf{v_t}-\mathbf{b_t})\Phi dL = \oint_{\partial S} \mathbf{m}\cdot(D_s\nabla_s\Phi)dL + \int_S s_\Phi dS \tag{4.3}$$

where s_Φ is the source/sink of surfactant per unit area due to absorption and desorption and is given by

$$s_\Phi = k_a[C_s(\Phi_\infty - \Phi) - \beta\Phi] \tag{4.4}$$

More information can be found[8]. $surfaceTension$ is the tension of the $freeSurface$ when no surfactants are present and g is the gravitational acceleration. To ensure constant mass flow $fixedFreeSurfacePatches1(left)$ option is used. It locks the left boundary so it can not move. If the boundary also should be part of the solution the domain will look like the domain in the background of Figure 4.5 and 4.6, respectively, instead of the original one shown in foreground.

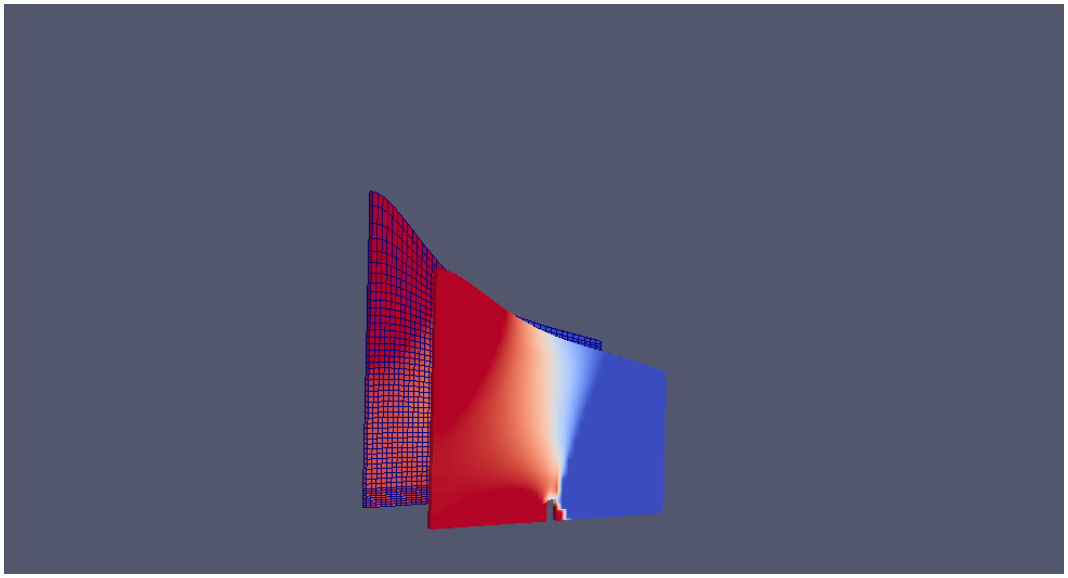

Figure 4.5: Inlet height change

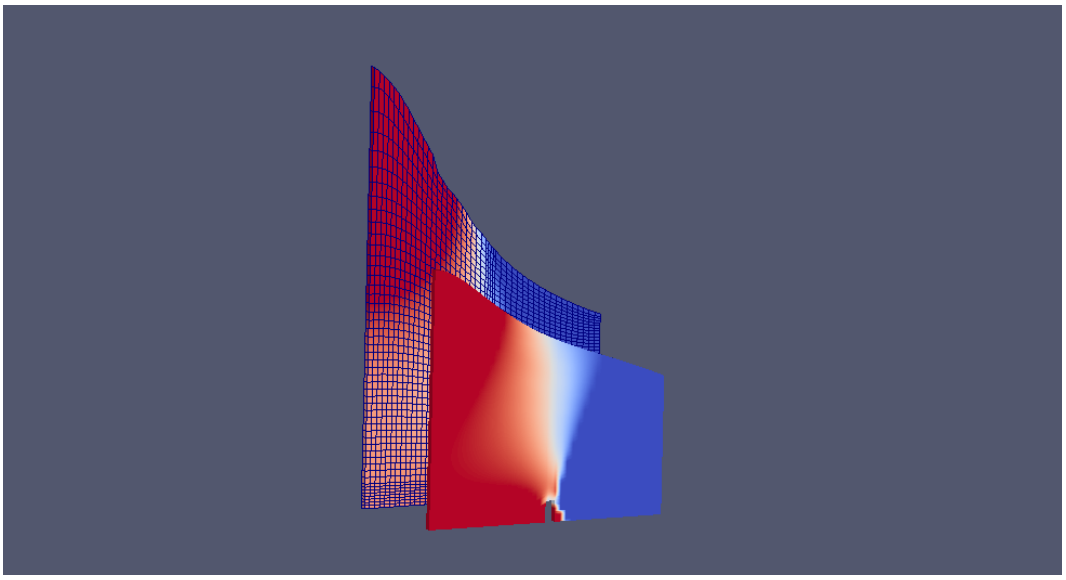

Figure 4.6: Inlet height change

4.2.4 polyMesh

As in all cases in OpenFOAM, the *polyMesh* directory contains the polymesh. The domain for *hydrofoil* has already been setup and does not include a *blockMeshDict* file but rather

```
boundary  cells.gz  faces.gz  neighbour.gz  owner.gz  points.gz
```

However, the *modDamBreak* includes this file as shown in Section 3. The polymesh is based around faces with internal faces connecting two cells and boundary faces addressing a cell and a boundary patch. The owner and neighbour cell labels therefore describe the connection between cells. OpenFoam needs all these files in order to build up a domain/mesh[1].

4.2.5 transportProperties

transportProperties specifies the viscosity for the application and contains only it, namely

```
nu              nu [0 2 -1 0 0 0 0] 1.5e-3;
```

4.3 system

In the system dictionary, the settings necessary in order to control the solution procedure are set. For the *modDamBreak* case these include;

`controlDict faSchemes faSolution fvSchemes fvSolution tetFemSolution`

In *controlDict*, solver (*interTrackFoam* for this case) and time (start/end time as well as timestep and writing controls of the output) parameters are set, for example *startTime*, *deltaT* and *writeControl*. The *faSchemes* and *fvSchemes* are taking care of the different types of discretisation schemes used in the solution and *faSolution* as well as *fvSolution* includes the equation solvers, tolerances and other algorithm controls. It is important to know that solvers in this case mean the "linear-solver", which is used when solving the sets of discretised equation and not the application solver (*interTrackFoam*), which is used to solve a specific case (*modDamBreak*, *hydrofoil* etc). Algorithm controls refer to the settings for SIMPLE algorithm used when implicity solving for pressure. Another file *tetFemSolution* is also part of *system* and it is necessary in order to solve the equations corresponding to the given mesh motion.

Chapter 5

discretisation

5.1 blockMesh

The *blockMesh* utility provided in OpenFOAM is used in order to discretise the computational space into a finite number of convex polyhedral cells bounded by convex polygons. By "convex" it is meant that the normals point in the same direction. Also the computational domain is split into a finite number of time-steps, which means that the equations also are solved in time[9]. The discretised domain *modDamBreak* is shown in Figure 5.1.

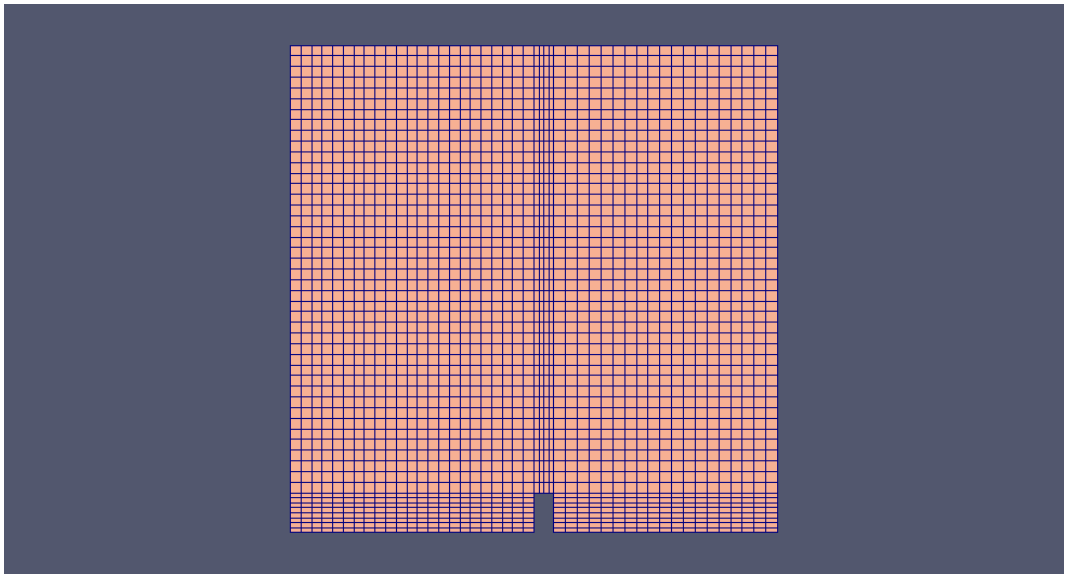

Figure 5.1: Finite-volume discretization

5.2 makeFaMesh

As has already been mentioned, the settings concerning the finite-area discretisation is done in $faMeshDefiniton$ (Section 4.2.2). The reason why the $makeFaMesh$ utility is used, is the fact that the $freeSurface$ class requires it. The $freeSurface$ is shown in Figure 5.2.

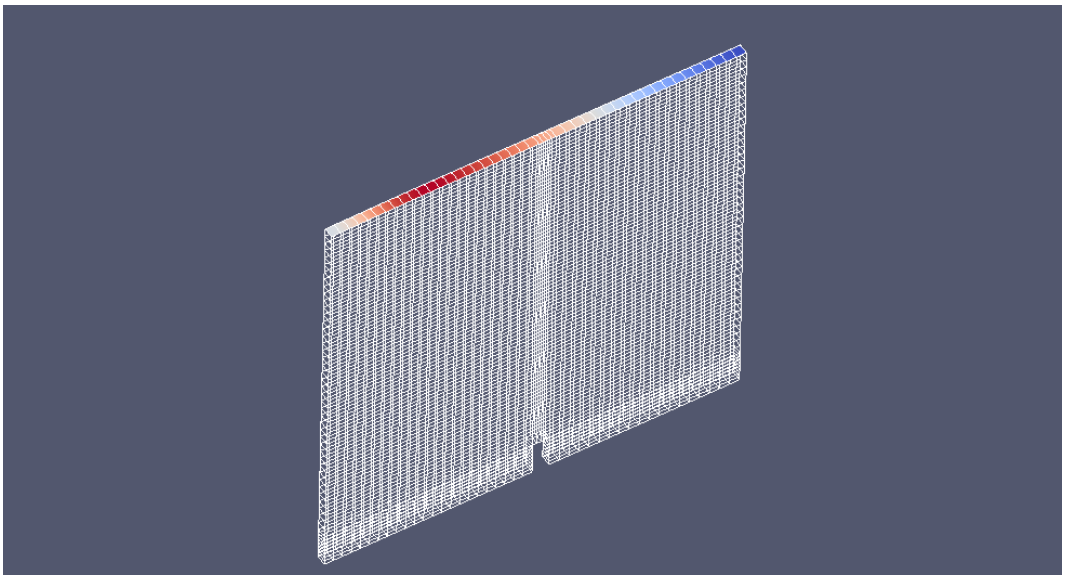

Figure 5.2: Finite-area discretization

5.3 Automatic mesh motion

The polyhedral cells are split into tetrahedrals[10]. The mesh motion equation (Section 4.2.1) is discretised on the tetrahedral decomposition using standard second-order finite element method. The finite element matrix obtained is then solved using an iterative linear equation solver (Section 4.3, $tetFemSoltution$). The motion of the boundaries are specified as boundary conditions on the $motionU$ field, and then used in the solution to update point position. It is important to note that, the interface (i.e. the $freeSurface$) is moved according to the calculated displacements after the iteration process for each time steps is carried out. This means that the $freeSurface$ is fixed during the calculation for each timestep and moved afterwards. If this was not the case the mesh-motion swept would be wrong

at the interface. The mass flux at the interface should be close to zero in order to conserve the mass.

Chapter 6

interTrackFoam

An explanation of the solver *interTrackFoam* with all its features will now be presented. First of all, as always in OpenFoam code, header files are included in order to include classes which will be used in the computational process.

```
#include "fvCFD.H"
#include "motionSolver.H"
#include "freeSurface.H"
#include "OFstream.H"

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

The $fvCFD.H$ file includes the finite volume class definitons, $motionSolver.H$ includes for instance the $twoDPointCorrector$ mentioned in Section 4.2.1, $freeSurface.H$ is used in the implementation of the freeSurface tracking method considering a moving mesh and $OFstream.H$ among others handle compressed files. Header files are mainly used to make the code structure easier to use. The initial conditions are then specified ($setRootCase.H$, $createTime.H$, $createMesh-$
$NoClear.H$,$createFields.H$ and $initContinuityErrs.H$).

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

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

#       include "readInterfaceSIMPLEControls.H"
#       include "CourantNo.H"

        interface.updateDisplacementDirections();

        interface.moveMeshPointsForOldFreeSurfDisplacement();

        interface.smooth();
```

The iteration process starts and will continue until the time set in *controlDict* is reached, Section 4.3. The non-orthogonal controls are read ($readInterfaceSIMPLEControls.H$) from the $fvSolution$ file in order to control the correction of the pressure field later on. The maximum as well as the medium Courant number($CourantNo.H$) are also calculated. When the fields was created in the header file $createFields.H$ an interface of class $freeSurface$ was created.
The subsequent tables (Table 6.1,6.2 and 6.3, respectively) will now explain the commands that are used in the solver.

$interface.updateDisplacementDirections()$
The $interface$ defined in $createFields.H$ is updated according to the displacement direction. This is only done if $normalMotionDir$ (Section 4.2.3) is allowed.

Table 6.1: Update the displacement directions

$interface.moveMeshPointsForOldFreeSurfDisplacement()$
All internal mesh points are moved according to the displacement of the interface ($motionU$) in the previous timestep.

Table 6.2: Move internal mesh points according to *motionU*

$interface.smooth()$
Smooth the interface if this option is specified, i.e. $freeSurfaceSmoothing$ (Section 4.2.3). Also the $twoDMotion$ has to be set to "no".

Table 6.3: Surfacesmoothing

The fluid flow equations are solved using a segregated SIMPLE procedure[11], taking into account the kinematic and dynamic condition of the interface, as well as the surface tension. The no-flux condition is satisfied in an iterative sequence, providing the boundary condition for mesh motion on the free surface ($motionU$). This is shown in the following code lines

```
// --- SIMPLE loop
for (label timeCorr=0; timeCorr<=nTimeCorr; timeCorr++)
{
    p.storePrevIter();

    interface.correctBoundaryConditions();

    tmp<fvVectorMatrix> UEqn
    (
        fvm::ddt(rho, U)
      + fvm::div(phiNet, U)
      - fvm::laplacian(mu, U)
    );

    UEqn().relax();

    solve(UEqn() == - fvc::grad(p));

    volScalarField AU = UEqn().A();

    U = UEqn().H()/AU;
    U.correctBoundaryConditions();

    UEqn.clear();

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

The pressure from previous iteration is stored using $p.storPrevIter()$ and the boundary conditions at the interface is corrected using $interface.correct-$

BoundaryConditions. It is important to note that the *.correct−BoundaryConditions* does a couple of things, namely

```
void freeSurface::correctBoundaryConditions()
{
    correctVelocity();
    correctVelocityGradient();
    correctSurfactantConcentration();
    correctPressure();
}
```

A temporary matrix of of type $fvVectorMatrix$ is used to solve for the velocity U and it looks like

$$\frac{\partial \rho \overrightarrow{U}}{\partial t} + \nabla \cdot \phi \overrightarrow{U} - \nabla \cdot \mu \nabla \overrightarrow{U} \tag{6.1}$$

By using a relaxation factor (UEqn.relax()) defined in $fvSolution$ (section 4.3) the iteration process is limited but less sensitive. The relaxed $UEqn$ is then solved by coupling the pressure obtained from previous iteration to it, i.e.

$$\frac{\partial \rho \overrightarrow{U}}{\partial t} + \nabla \cdot \phi \overrightarrow{U} - \nabla \cdot \mu \nabla \overrightarrow{U} = -\nabla p \tag{6.2}$$

The central coefficients from the $UEqn$ matrix (UEqn().A()) are calculated and the velocity U. The $U.correctBoundaryConditions$ command is used to update the boundary conditions for the new velocity field while $UEqn.clear()$ clears the vector to reduce peak memory. It is important to note that the velocity distribution is stored without being affected by the pressure. The velocities for each cell faces (*phi*) are obtained by interpolation. This is done and necessary in order to later on calulate the fluxes at the faces.
Next step is to calculate the pressure and then correct it. Since an explicit equation for the pressure is not available, one common way is to derive it simply by using the divergence of the momentum equation and by substituting it in the continuity equation.

```
    // Non-orthogonal pressure corrector loop
    for (label nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
    {
        tmp<fvScalarMatrix> pEqn
        (
            fvm::laplacian(1.0/AU, p) == fvc::div(phi)
        );
```

```
            pEqn().setReference(0, 0.0);
            pEqn().solve();

            if (nonOrth == nNonOrthCorr)
            {
                phi -= pEqn().flux();
            }
        }

#       include "continuityErrs.H"

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

        // Momentum corrector
        U -= fvc::grad(p)/AU;
        U.correctBoundaryConditions();

        // Move mesh
        interface.movePoints();

        // Update motion fluxes
        phiNet = fvc::interpolate(rho)*(phi - fvc::meshPhi(rho, U));

#   include "freeSurfaceContinuityErrs.H"
```

Solve the pressure equaton iteratively and correct the fluxes at the cell faces using control points[8]. Number of non-orthogonal corrector steps is set in $fvSolution, nNonOrthCorr$ and the iteration procedure is continued until it is reached. As before for the $UEqn$, a temporary matrix of type $fvScalarMatrix$ is used to define the pressure equation $pEqn$. By using $continuityErrs.H$ the continuity errors are calculated. Under-relax the pressure for the momentum corrector and then correct the boundary conditions for U. Move the mesh considering the new boundary conditions and update the motion fluxes.

```
            runTime.write();

            Info << "ExecutionTime = "
                << scalar(runTime.elapsedCpuTime())
```

```
            << " s\n" << endl << endl;
    }

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

    return(0);
}
```

The last thing done for each time step is to print the "ExecutionTime".

Bibliography

[1] OpenFOAM documentation, "User guide." http://openfoam.com/docs/user/, October 2010.

[2] OpenFOAM, "twodpointcorrector." ~/OpenFOAM-1.5-dev/src/meshTools/twoDPointCorrector, October 2010.

[3] A. O. Gonzlez, "Mesh motion alternatives in openfoam," tech. rep., Institution of Applied Mechanics at Chalmers Technical University, Gteborg, Sweden, 2009. http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2009/AndreuOliverGonzalez/ProjectReport_Corrected.pdf.

[4] H. Jasak, "Dynamic mesh handling in openfoam," tech. rep., Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb, Croatia, London, England, 2009. http://powerlab.fsb.hr/ped/kturbo/openfoam/papers/dynamicMesh_AIAA2009.pdf.

[5] H. Jasak and Z. Tukovic, "Dynamic handling in openfoam applied to fluid-structure interaction simulations," tech. rep., Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb, Croatia, London, England, 2010. http://web.univ-ubs.fr/limatb/EG2M/Disc_Seminaire/ECCOMAS-CFD2010/papers/01178.pdf.

[6] Wikipedia, "Wikipedia." http://en.wikipedia.org/wiki/Radial_basis_function, October 2010.

[7] OpenFOAM, "icodymfoam." ~/OpenFOAM-1.5-dev/applications/solvers/incompressible/icoDyMFoam, October 2010.

[8] Z. Tukovic and H. Jasak, "Simulation of free-rising bubble with soluble surfactant using moving mesh finite volume/area method," tech. rep., Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb, Croatia, London, England, 2008. http://powerlab.fsb.hr/ped/kturbo/openfoam/papers/CFD2008.pdf.

[9] Z. Tukovic and H. Jasak, "Updated lagrangian finite volume solver for large deformation dynamic response of elastic body," tech. rep., Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb, Croatia, London, England, 2007. http://powerlab.fsb.hr/ped/kturbo/openfoam/papers/TukovicJasak_NonLinElastodynamics_FAMENA_18-06-2007.pdf.

[10] Z. Tukovic and H. Jasak, "Automatic mesh motion for the unstructured finite volume method," tech. rep., Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb, Croatia, London, England, 2004. http://powerlab.fsb.hr/ped/kturbo/openfoam/papers/MeshMotionJCPManuscript.pdf.

[11] OpenFOAMWiki, "Openfoamwiki." http://openfoamwiki.net, October 2010.