CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Project work:

A tutorial of the sixDofRigidBodyMotion library with multiple bodies

---

Developed for OpenFOAM-2.4.x

Author:
Magnus Urquhart

Peer reviewed by:
Abishek Saraf
Håkan Nilsson

January 23, 2016

Learning outcomes

The reader will learn:

- How to use the sixDoFRigidBodyMotion library for coupled and uncoupled bodies
- How to implement and modify their own sixDofRigidBodyMotion solver
- How to use snappyHexMesh from a short guide describing mesh refinements and useful utilities

Contents

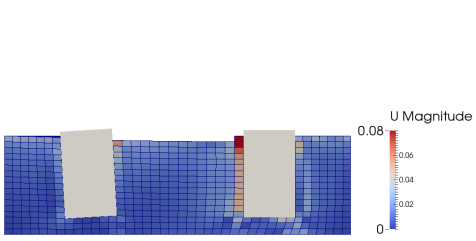
Chapter 1

Tutorial sixDoFRigidBodyMotion

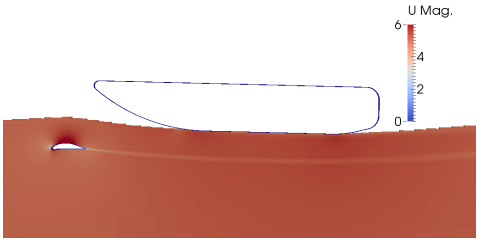
1.1 Introduction

This tutorial describes how the sixDoFRigidBodyMotion library works and how to use the library for several coupled and uncoupled rigid bodies. It also describes how to modify and implement your own sixDoFRidigBodyMotion library to output the rotational Euler angles of a rotating solid body. A short guide describing useful commands and mesh refinements for snappyHexMesh is also included the tutorial.

The use of the uncoupled and coupled sixDoFRigidBodyLibrary will be demonstrated on a tutorial case in OpenFOAM called floatingObject. The tutorial is modified to have two floating objects as in figure 1.1a. One of the implementations will be shown on a 2D case consiting of a boat with a hydrofoil (figure 1.1b) demonstrating the use of snappyHexMesh and when the sixDoFRigidBodyMotion coupled solver is useful.



(a) TwoFloatingObjects simulation



(b) Boat with hydrofoil simulation

Figure 1.1: Simulations showcasing the different implementations of sixDoFRigidBodyMotion

1.2 Background

By solving the motion of a solid body caused by the forces acting on it we can for example predict the transient behaviour of a boat or the output power of a turbine more accurately. OpenFOAM improved the capability of the sixDoFRigidBodyMotion solver in version 2.3.x which can now handle mesh motion and several rigidly coupled bodies. The purpose of this report is to give the reader insight in how to use the coupled and uncoupled sixDoFRigidBodyMotion solver and in which situations they're useful for the user.

1.3 Theory

OpenFOAMs sixDoFRigidBodyMotion solver is capable of solving translational and rotational movement of a solid body in the three axises xyz. SixDoFRigidBodyMotion solves the motion of a body based on the forces acting on it. The user is able to use the sixDoFRigidBodyMotion library with the solvers using dynamic mesh such as pimpleDyMFoam or interDyMFoam.

1.3.1 Forces and moments

The forces acting on a body can be split into two categories, flow or external forces. Flow forces are forces that the fluid impose such as pressure (buoyancy) or viscous forces while external forces are forces such as gravity or spring forces. The forces combined will be called $\boldsymbol{F}$ and the combined the moments will be called $\boldsymbol{M}$. In order to calculate the motion of the body we need to know the acceleration $\boldsymbol{a}$ and the angular acceleration $\boldsymbol{\alpha}$. Those are related to the moments and forces as

$$\begin{aligned} \sum \mathbf{F} &= m\mathbf{a} \\ \sum \mathbf{M} &= \boldsymbol{\alpha}\mathbf{I}, \end{aligned} \tag{1.1}$$

here m denotes the mass and $\boldsymbol{I}$ denotes the inertia. By intergrating the angular and translational acceleration over the time step as

$$\begin{aligned} \boldsymbol{\omega}_{new} &= \int_{t_{old}}^{t} \boldsymbol{\alpha} dt = \boldsymbol{\omega}_{old} + \boldsymbol{\alpha}\Delta t \\ \mathbf{v}_{new} &= \int_{t_{old}}^{t} \mathbf{a} dt = \mathbf{v}_{old} + \mathbf{a}\Delta t, \end{aligned} \tag{1.2}$$

we get the current angular velocity $\boldsymbol{\omega}_{new}$ and velocity $\boldsymbol{v}_{new}$. From $\boldsymbol{\omega}_{new}$ and $\boldsymbol{v}_{new}$ we can get the angle rotated and the distance traveled from the last time step.

1.3.2 Mesh motion

After calculating the motion of the rigid body it is neccesary to move the boundary as well as the mesh surrounding the body in order to maintain a good quality mesh. In this tutorial we will use two different solvers for the mesh deformation, displacementLaplacian and slerp. Slerp is short for spherical linear interpolation and is accoring to the release notes of OpenFOAM 2.3.x [1] better at handling rotating mesh deformation when it comes to cell shearing. For more information on how the displacementLaplacian works see the report Mesh motion alternatives in OpenFOAM by Andreu Oliver González [2].

1.4 Tutorial - sixDoF

The sixDoFRigidBodyMotion library was updated from version 2.2.x to 2.3.x to handle mesh motion (as previously mentioned) as well as several rigidly coupled bodies. For the user this means differences in how the `0\pointDisplacement` (which handles boundary conditions) and `dynamicMeshDict` file is set up in order to use the coupled and uncoupled solver (apart from the new slerp mesh motion). The coupled version of sixDoFRigidBodyMotion is capable of handling multiple rigidly connected patches while the uncoupled version is suitable for multiple unconnected patches.

In OpenFOAM 2.4.x it is possible to use both uncoupled and coupled (introduced in 2.3.x) motion of several patches. The sixDoFRigidBodyMotion library has two seperate solvers to handle coupled or uncoupled movement. The following tutorials will show the user how the coupled and uncoupled solver of sixDoFRigidBodyMotion can be used. To show the differences between coupled and uncoupled rigid body motion for seperate patches we will use the `floatingObject` tutorial as our base. The `floatingObject` tutorial shows how to simulate the motion of a floating body due to disturbed water with the use of sixDoFRigidBodyMotion. We will add a second floating object to this tutorial and simulate their motion both as a coupled rigid body and as two seperate uncoupled rigid bodies. Switching between coupled and uncoupled motion requires modifications to the `0\pointDisplacement` and `dynamicMeshDict` file which will be shown in the tutorial.

1.4.1 Case 1 - TwoFloatingObjects - Coupled rigid sixDoF

Case 1 treats the two floating objects as coupled. This means that the total integrated forces and moments acting on both bodies will be summed together and used to calculated the coupled motion of the bodies. We need to modify the `floatingObject` tutorial to contain two objects. Start this tutorial by copying the `floatingObject` tutorial to your `$FOAM_RUN` directory and renaming it to `TwoFloatingObject` by copy pasting the following in a terminal.

```
OF24x
cp -r $FOAM_TUTORIALS/multiphase/interDyMFoam/ras/floatingObject/ $FOAM_RUN
cd $FOAM_RUN
mv floatingObject TwoFloatingObjects
cd TwoFloatingObjects
```

It is a good idea to try to run the case now and look at the results in order to get familiar with the base case. Run the case with the `Allrun` script by typing `./Allrun` in the terminal, when you are finished clean the case with the `Allclean` script by typing `./Allclean` in the terminal.

The standard tutorial uses `topoSet` to extract and remove a box of cells located in the domain forming the floating object. To get two floating objects we add a second `system/topoSetDict` file so that we can create the two floating objects needed. Copy the `system/topoSetDict` file with the following.

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

We need to make modifications to the `system/topoSetDict2` file to reposition the second object as well as rename the `cellSet`. Edit the `system/topoSetDict2` with a text editor and change the `name` from `c0` to `c1` in both locations as well as decrease the distance in x-direction of the box by one meter. After the changes your `system/topoSetDict2` should look like Code 1.1.

Code 1.1: topoSetDict2

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      topoSetDict;
}

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

actions
(
    {
        name    c1;
        type    cellSet;
        action  new;
        source  boxToCell;
        sourceInfo
        {
            box (-0.65 0.35 0.1) (-0.35 0.55 0.6);
        }
    }

    {
        name    c1;
        type    cellSet;
        action  invert;
    }
);

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

To fit the new `floatingObject2` in our computational domain we need to make the appropriate changes to the domain size in the `constant/polyMesh/blockMeshDict` file. We increase the length of the domain in x-direction by moving the vertices located at `x = 0` to `x = -1`. In order to keep the same cell dimensions as previously we increase the cell count in x-direction from `20` to `40`. Finally we also need to add the second floating object by copying the first one and renaming it to `floatingObject2`. With all the changes made the `constant/polyMesh/blockMeshDict` file should look like Code 1.2 (notice the changes in `vertices, blocks` and the added `floatingObject2`).

Code 1.2: blockMeshDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
    (-1 0 0)
    (1 0 0)
    (1 1 0)
    (-1 1 0)
    (-1 0 1)
    (1 0 1)
    (1 1 1)
    (-1 1 1)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (40 20 30) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
    stationaryWalls
    {
        type wall;
        faces
        (
            (0 3 2 1)
            (2 6 5 1)
            (1 5 4 0)
            (3 7 6 2)
            (0 4 7 3)
        );
    }
    atmosphere
    {
        type patch;
        faces
        (
            (4 5 6 7)
        );
    }
    floatingObject
    {
        type wall;
        faces ();
    }
    floatingObject2
    {
        type wall;
        faces ();
    }
);

mergePatchPairs
(
);

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

The boundary conditions for the second floating object, `floatingObject2`, are set in the `0.org\` folder. The `alpha.water`, `epsilon`, `k`, `nut`, `pointDisplacement`, `p_rgh`, and `U` files in the `0.org\` need to be modified so that they contain boundary conditions for `floatingObject2`. Copy and use the same values for `floatingObject2` as for `floatingObject` in each of the files. Once the second floating object has all the boundary conditions edit the `pointDisplacement` file. To use the coupled solver for sixDoFRigidBodyMotion we add the second object to the `constants/dynamicMeshDict` as a `patch`. The physical properties need to be changed since the two floating objects are treated as one body. For the sake of simplicity we will change the `mass` and `centreOfMass` only. The `mass` is trippled and the `centreOfMass` is moved by 0.5 meter in the negative x-direction. Note that this does not represent real physical properties of the floating objects but is used in order run the simulation and build upon this case for the uncoupled movement.

Code 1.3: dynamicMeshDict

```
patches         (floatingObject floatingObject2);       //Add second floating object
    innerDistance   0.05;
    outerDistance   0.35;

    centreOfMass    (0 0.45 0.35);                      //Change centreOfMass

    // Cuboid dimensions
    Lx              0.3;
    Ly              0.2;
    Lz              0.5;

    // Density of the solid
    rho             500;

    // Cuboid mass
    mass            #calc "$rho*$Lx*$Ly*$Lz*3";         //Change mass
```

Code 1.3 shows part of the `constants/dynamicMeshDict` file with the neccesary changes to `patches`, `centreOfMass` and `mass`. The case is now ready to be run manually however to make it easier to run we modify the `Allrun` script so it can be used for this case. The script is modified to make two objects, `floatingObject` from `cellSet c0` and `floatingObject2` from `cellSet c1`. Inbetween the making of the two floating objects we need to remove or rename the `log.subsetMesh` and `log.topoSet`. The final script is shown in code 1.4. Replace all text in the `Allrun` script with the text in code 1.4 (copy-paste).

Code 1.4: Allrun script

```
#!/bin/sh
cd ${0%/*} || exit 1    # run from this directory

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

# Set application name
application=`getApplication`

runApplication blockMesh
runApplication topoSet -dict system/topoSetDict
runApplication subsetMesh c0 -patch floatingObject -overwrite
mv log.subsetMesh log.subsetMesh0
mv log.topoSet log.topoSet0
runApplication topoSet -dict system/topoSetDict2
runApplication subsetMesh c1 -patch floatingObject2 -overwrite
cp -r 0.org 0 > /dev/null 2>&1
runApplication setFields
runApplication $application

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

Run the case by using the `Allrun` script. Type `./Allrun` in the terminal window.

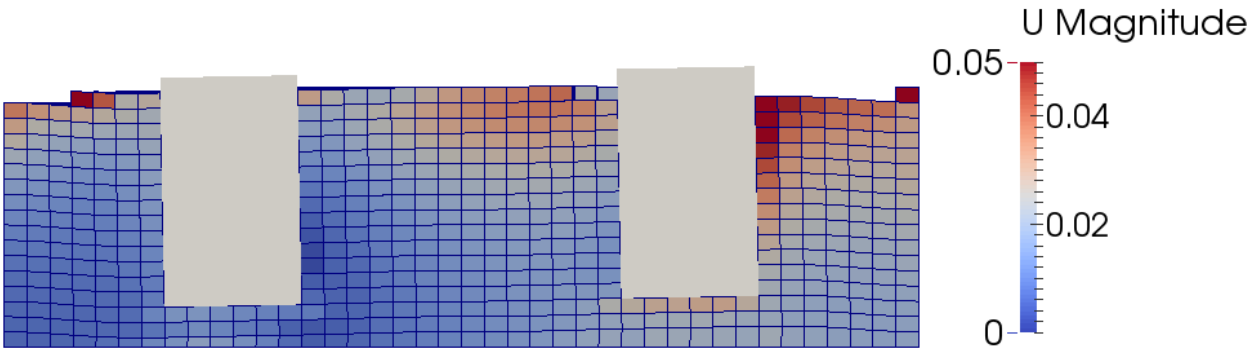


Figure 1.2: Case 1 - Two coupled floating objects

Figure 1.2 shows the results after 4.2 seconds where we can see that both bodies have move and that the mesh around them has deformed. The bodies in this tutorial have moved as one solid body which can be easily seen while playing the animation during post processing.

1.4.2 Case 2 - TwoFloatingObjects - Uncoupled sixDoF

The second tutorial, Case 2, will show how to set up two uncoupled rigid bodies that will be translated and rotated by the sixDoFRigidBodyMotion solver in the same simulation. For the second case we start off from the first case as our base.

```
cd $FOAM_RUN
cp -r TwoFloatingObjects TwoFloatingObjects_c2
cd TwoFloatingObjects_c2
./Allclean
```

In order to use the uncoupled sixDoFRigidBodyMotion solver we need to edit the `constants/dynamicMeshDict` and `0.org/pointDisplacement` file. When using the uncoupled solver we only specify the solvers we want to use in the `constants/dynamicMeshDict` file and all the physical properties of the bodies are specified in the `0.org/pointDisplacement` instead. We need to add the `libfvMotionSolvers.so` library to solve the mesh motion and add the appropriate solver settings. Edit the `constants/dynamicMeshDict` file, the text in code 1.5 can be copy-pasted to replace all text in the `constants/dynamicMeshDict` file.

Code 1.5: dynamicMeshDict

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

dynamicFvMesh dynamicMotionSolverFvMesh;

motionSolverLibs
(
    "libsixDoFRigidBodyMotion.so"
    "libfvMotionSolvers.so"                                     //Add libfvMotionSolvers.so
);

solver displacementLaplacian;                                   //Change solver

displacementLaplacianCoeffs                                     //Add displacementLaplacianCoeffs
{
    diffusivity inverseDistance (floatingObject floatingObject2);
}

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

Next we need to make the appropriate changes in the `0.org/pointDisplacement` file for the two rigid bodies. The properties of the floating objects can be the same as in the original `floatingObject` tutorial with the exception of moving the `centreOfMass` and `centreOfRotation` by one meter in the positive x-direction. Copy-paste the text in code 1.6 for `floatingObject2`.

Code 1.6: pointDiscplacement

```
floatingObject2
{
    type            sixDoFRigidBodyDisplacement;
    value           uniform (0 0 0);
    centreOfMass    (-0.5 0.45 0.35);

    // Cuboid dimensions
    Lx              0.3;
    Ly              0.2;
        Lz              0.5;

    // Density of the solid
    rho             500;

    // Cuboid mass
    mass            #calc "$rho*$Lx*$Ly*$Lz";

    // Cuboid moment of inertia about the centre of mass
    momentOfInertia #codeStream
    {
        codeInclude
        #{
            #include "diagTensor.H"
        #};

        code
        #{
            scalar sqrLx = sqr($Lx);
            scalar sqrLy = sqr($Ly);
            scalar sqrLz = sqr($Lz);
            os  <<
                $mass
               *diagTensor(sqrLy + sqrLz, sqrLx + sqrLz, sqrLx + sqrLy)/12.0;
        #};
    };

    report          on;
    accelerationRelaxation 0.3;

    constraints
    {

        fixedLine
        {
            sixDoFRigidBodyMotionConstraint line;
            centreOfRotation (-0.5 0.45 0.1);
            direction (0 1 0);
        }

        fixedAxis
        {
            sixDoFRigidBodyMotionConstraint axis;
            axis (0 1 0);
        }
    }
}
```

The same settings is used for `floatingObject` with the exception of `centreOfMass` and `centreOfRotation` which should be the following.

Code 1.7: pointDiscplacement

```
    centreOfMass    (0.5 0.45 0.35);
    .
    .
    .
    centreOfRotation (0.5 0.45 0.1);
```

After you have added the correct properties to the seperate floating bodies we need to add information about the `cellDisplacement` in the `system/fvSolution` file. These settings are used for the

mesh motion solver.

Code 1.8: fvSolution

```
solvers
{
    cellDisplacement
    {
        solver          GAMG;
        tolerance       1e-5;
        relTol          0;
        smoother        GaussSeidel;
        cacheAgglomeration true;
        nCellsInCoarsestLevel 10;
        agglomerator    faceAreaPair;
        mergeLevels     1;
    }
    cellDisplacementFinal
    {
        solver          GAMG;
        tolerance       1e-5;
        relTol          0;
        smoother        GaussSeidel;
        cacheAgglomeration true;
        nCellsInCoarsestLevel 10;
        agglomerator    faceAreaPair;
        mergeLevels     1;
    }
    "alpha.water.*"
```

Copy-paste the settings in code 1.8 which adds the necessary information about the `cellDisplacement` to the `system/fvSolution` file. The case in now ready to run. Use the same `Allrun` script. Type into the terminal.

```
./Allrun
```

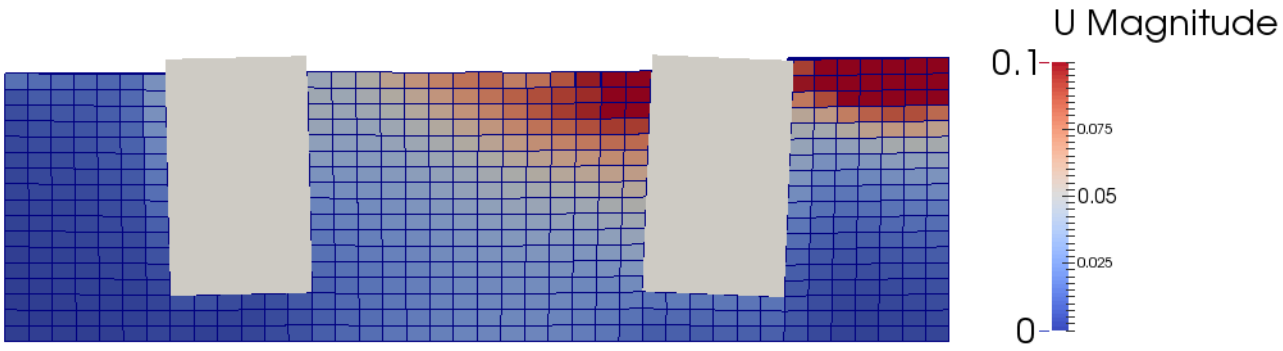


Figure 1.3: Case 2 - Two uncoupled floating objects

Figure 1.3 shows the two uncoupled floating objects. When playing the animation while post processing it's clear that the two floating objects are not mechanically coupled. From the figure we can see a small amount of mesh deformation as well as a small difference in angle between the bodies.

1.4.3 Case 3 - TwoFloatingObjects - prescribed motion and sixDoF

When using the uncoupled sixDoFRigidBodyMotion solver we can of course use different boundary conditions for the two floating objects. This case uses a prescribed oscillating motion for `floatingObject` and sixDoF for `floatingObject2`. Starting from the previous tutorial Case 2.

```
cd $FOAM_RUN
cp -r TwoFloatingObjects_c2 TwoFloatingObjects_c3
cd TwoFloatingObjects_c3
./Allclean
```

The boundary condition for `floatingObject` in the `0.org/pointDisplacement` file is changed to a oscillating displacement. Change the boundary condition for `floatingObject` in the `0.org/pointDisplacement` file accodring to code 1.9.

Code 1.9: pointDiscplacement - oscillatingDisplacement

```
    floatingObject
    {
        type            oscillatingDisplacement;
        amplitude       (0.03 0 0);
        omega           6;
        value           uniform (0 0 0);
    }
```

To show that the motion of `floatingObject2` is only influenced by `floatingObject` we remove the initial wave created in `system/setFieldsDict`. Remove the second `boxToCell` in the `system/setFieldsDict` file so that the `regions` only contains the initial water level as shown in code 1.10.

Code 1.10: setFieldsDict - No initial wave

```
regions
(
    boxToCell
    {
        box ( -100 -100 -100 ) ( 100 100 0.5368 );
        fieldValues ( volScalarFieldValue alpha.water 1 );
    }
);
```

Run the case using the same `Allrun` script as previously. Type into the terminal.

```
./Allrun
```

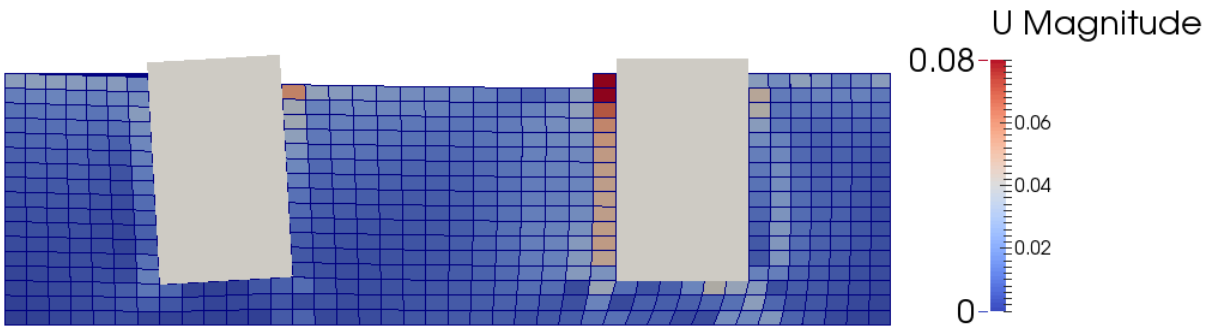


Figure 1.4: Case 2 - Two floating objects, sixDoF and prescribed motion

Figure 1.4 shows how `floatingObject2` (left object in figure 1.4) has been rotated because of the motion of `floatingObject` (right object in figure 1.4). This is easily seen when post processing and playing the animation. This type of implementation can be useful when simulating the motion of a boat induced by the opening of a dam for instance.

1.4.4 Example of rigid coupled sixDoF - Boat with hydrofoil

An example of when simulating seperate patches as one solid body (coupled) can be useful is for instance when simulating a boat with a hydrofoil. This tutorial case will show how to use `snappyHexMesh` in order to get a 2D mesh as well as show the use of coupled `sixDoFRigidBodyMotion`.

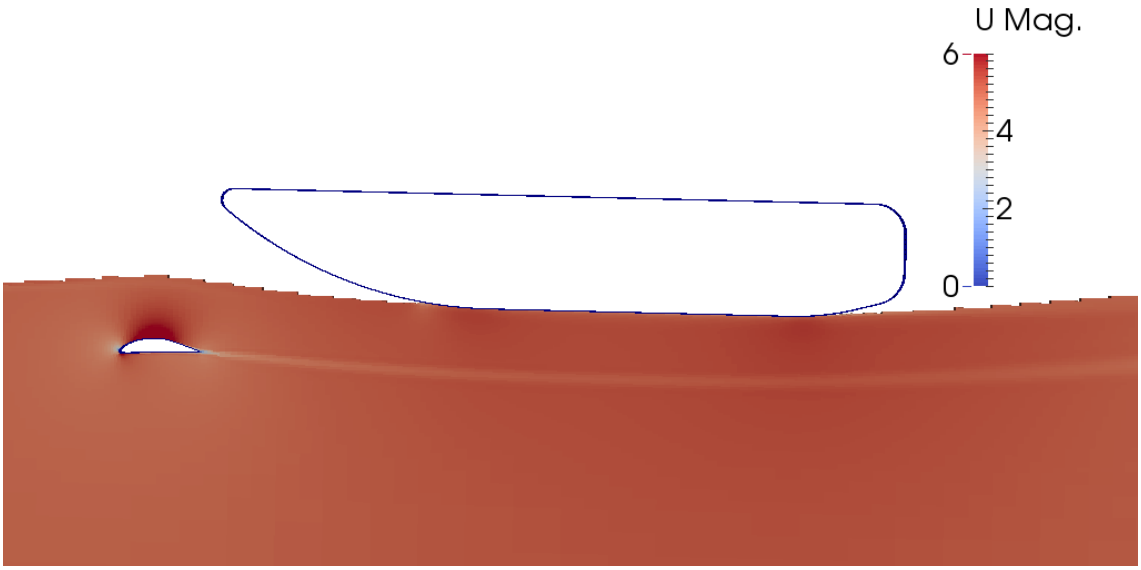


Figure 1.5: Boat with hydrofoil

The simulation seen in figure 1.5 is meshed using `snappyHexMesh` and the surface of the boat and hydrofoil was made in a seperate CAD software. Download the case from the course webpage containing the simulation files and geometry. When meshing such a case we need seperate patch names for both the hydrofoil and the hull since they have very different mesh requirements for the surface. The coupled implementation of sixDoFRigidBodyMotion solver is useful in such a case since the hydrofoil and boat are easily connected as one rigid body even though they have seperate patches, as we saw in tutorial Case 1. When simulating such a case it's important that the surrounding mesh, especially in-between the hull and hydrofoil, is deformed properly. In this case the movement of the boat is relativly large so it's important that the cells in-between the hull and hydrofoil follow the movmement of the boat.

The case is run parallel and is set up to run on 4 CPU-cores. Edit the `Allrun` script and the `system/decomposeParDict` file if you wish to alter the number of cores to run on. Run the downloaded case with the included `Allrun` script. The case is rather computationally heavy so it's recommended to start the simulation and then look at how it is setup. The case is largly based on the `DTCHull` tutorial located in `$FOAM_TUTORIALS/multiphase/interDyMFoam/ras/DTCHull/`.

Code 1.11: Allrun script

```
#!/bin/sh
cd ${0%/*} || exit 1    # run from this directory
. $WM_PROJECT_DIR/bin/tools/RunFunctions
runApplication blockMesh
runApplication snappyHexMesh -overwrite
\rm -rf 0
\cp -r 0.org 0
runApplication extrudeMesh
runApplication createPatch -overwrite
runApplication setFields
runApplication decomposePar
runParallel renumberMesh 4 -overwrite
runParallel $(getApplication) 4
#runApplication $(getApplication)
runApplication reconstructPar
# ----------------------------------------------------------------- end-of-file
```

Code 1.11 shows the complete `Allrun` script for the case. Compared to the previous tutorials this case uses `snappyHexMesh` to mesh the geometry. More information on commands and tips on `snappyHexMesh` follows in the next section, section 1.4.5. Looking at the `Allrun` script we can see that after running `snappyHexMesh` we run the application `extrudeMesh`. This is done in order to extract a 2D mesh from the xy-plane since `snappyHexMesh` can only mesh in 3D. Other than `snappyHexMesh` the case is similar to the previous tutorials with the addition of running in parallel. The domain is decomposed into smaller domains to be run parallely on 4 processors. Once the simulation is finished we reconstruct the smaller domains into one domain again which can be post processed with the application `reconstructPar`. Figure 1.6 shows how the cells in-between the

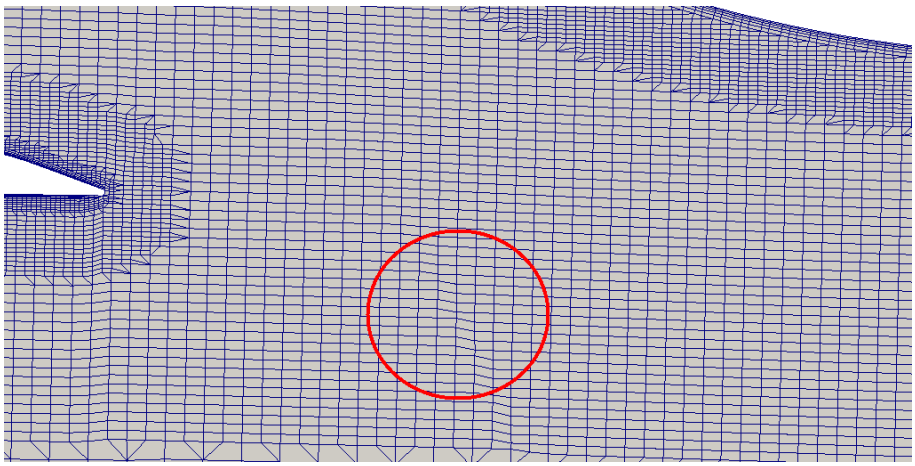


Figure 1.6: Boat with hydrofoil

hull and hydrofoil have followed the movement of the boat except further away where we can see some distortion of the cells in the red marked area. For cases with very large deformations an alternative strategy such as overset mesh can be used. To control which cells should be affected by the movement of the boat we use the `innerDistance` and `outerDistance` command in the `constant/dynamicMeshDict`. `innerDistance` sets the distance at which the cells close to the body ridigly follow the movement of that body while `outerDistance` sets how far out from the body we deform cells.

1.4.5 snappyHexMesh - Useful commands and utilities

`snappyHexMesh` is a meshing tool within OpenFOAM that is capable of meshing complex geometries, add surface layers, various refinements and more. The boat with hydrofoil simulation was meshed in `snappyHexMesh` and converted into a 2D mesh. The author has included some useful commands and utilities in this section that was used for the previous boat with hydrofoil simulation.

Refinements

Refinements in `snappyHexMesh` can be set as a volume, surface or distance refinements. Volume refinements can be set with a `searchableBox` among others, such as `searchableCylinder`, and the amount of refinement can be set by `levels` with the refinement located inside a `searchableBox` for example.

```
refinementBox
{
    type    searchableBox;
    min     (-999 -0.5 -1);
    max     (999 1.7 1);
}

refinementRegions
{
    refinementBox
    {
        mode inside;
        levels ((2 2));
    }
}
```

The above code shows an example where a refinement is placed within a box with a minimum and maximum refinement level of 2. The refinements level refers to how many times smaller the cell sides should be within the `searchableBox`. An increase in refinement of one level means that the sides of the cell will be half as long, or an 8 times reduction in volume for 3D meshes, which can lead to a rapid increase in cell count if the level is set too high. Volume refinements can also be set as a distance from a surface. The code below demonstrates how to set the refinement level of the cells that are located within 0.2 m from the patch named hydrofoil.

```
refinementRegions
{
    hydrofoil
    {
        mode distance;
        levels ((0.2 4));
    }
}
```

This type of refinement can be useful when the user needs a certain level of refinement close to the surface of a particular patch. `nCellsBetweenLevels` is the cell growth rate, or the number of layers of cells with the same level before changing level. `nCellsBetweenLevels` together with distance based volume refinements can be effectivly used to control the overall cell count and near surface cell size.

Surface refinements can be specified as the code below shows. The author found that the cell count can rapildy increase when increasing the level and that the best surface quality was obtained when using the same maximum and minimum refinement level.

```
refinementSurfaces
{
    hull
    {
        // Surface-wise min and max refinement level
        level (4 4);
    }
}
```

The author recommends that the user starts with the surface refinements while keeping the volume refinements coarse for a new case. This allows the user to fast find what surface refinement level is needed to capture the detail the user needs before increasing the cell count with volume refinements and growth rates. Try reducing the levels of refinement on the different refinements by one level in the `system/snappyHexMeshDict` and remeshing the case. Note how the cell count and meshing

times reduces. Do the same again but increase the levels by one level from the default case. It becomes clear that the refinements levels have a big impact on cell count and meshing time which can make fine tuning of the mesh cumbersome.

extrudeMesh - 3D to 2D mesh

extrudeMesh is a utility that can be used to convert a 3D mesh into a 2D mesh. `snappyHexMesh` is not capable of creating a 2D mesh by it's own. The utility `extrudeMesh` allows the user to convert a 3D mesh from `snappyHexMesh` into a 2D mesh. The `wingMotion` tutorial uses this utility, located at `$FOAM_TUTORIALS/incompressible/pimpleDyMFoam/wingMotion/`.

surfaceTransformPoints - Scale, Translate or Rotate a geometry (.stl file)

Another useful utility when using `snappyHexMesh` is `surfaceTransformPoints`. Most CAD software exports `.stl` files in mm. Instead of changing the settings in the CAD program one can use the utility `surfaceTransformPoints` to get the correct dimensions.

```
surfaceTransformPoints in.stl out.stl -scale (0.001 0.001 0.001)
```

The above command shows an example when scaling a `.stl` file from mm to m.

`surfaceTransformPoints` can also be used to translate and rotate a geometry. This funtion of the utility can be useful when running an airfoil simulation for many different angles of attack. The user could use this utility in a script to run many different angles of attack instead of manually rotating and saving the geometry in CAD software.

```
surfaceTransformPoints in.stl out.stl -rollPitchYaw (0 15 0)
```

The above code shows an example of a 15 degree Pitch increase.

Chapter 2

Tutorial - sixDoFRigidBodyMotion output, Euler angles

For previous tutorial cases it can be good to get the rotational movement of the bodies for post processing. This tutorial shows how to copy and modify the sixDoFRigidBodyMotion solver to output the Euler angles expressed in degrees for the three axes x, y and z.

2.1 Euler angles

Start by copying the source code to the user directory. Open a new terminal and copy paste the following

```
OF24x
foam
cp -r --parents src/sixDoFRigidBodyMotion/ $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR/src/
mv sixDoFRigidBodyMotion/ sixDoFRigidBodyMotionAngle/
cd sixDoFRigidBodyMotionAngle/
wclean
cd sixDoFRigidBodyMotion/
mv sixDoFRigidBodyMotion.C sixDoFRigidBodyMotionAngle.C
```

Add Code 2.1 to the `sixDoFRigidBodyMotionAngle.C` starting at line 340 in the `sixDoFRigidBodyMotionAngle.C` file. This will output the Euler angles while running which the user can later post process to plot the angular movement over time.

Code 2.1: Euler angle code

```
        quaternion b = quaternion(orientation());
        vector ang = b.eulerAngles(b);
    Info<< "    Euler angles: " << 180/3.14159265359*ang << endl;   //- Print the angle vector
```

While still in the `sixDoFRigidBodyMotionAngle.C` file we can see the theory that was presented in equation 1.2 at line 271-272.

Code 2.2: Theory code, line 271-272

```
v() = tConstraints_ & (v0() + aDamp_*0.5*deltaT0*a());
pi() = rConstraints_ & (pi0() + aDamp_*0.5*deltaT0*tau());
```

There are differences from the theory. In code 2.2 we have additional translational (`tConstraints_`) and rotational (`rConstraints_`) constraints, acceleration damping (`aDamp_`) as well as a scalar `0.5`. The scalar `0.5` is used because each time step is split into two, the first half of the timestep leapfrog moves the body updating the position while the second half updates the acceleration. Together they make up the movement and acceleration update of the entire timestep. For more information on how the rotation quaternion is used in OpenFOAM see the 6-DOF VOF-solver without Damping in OpenFOAM report by Erik Ekedahl [2].

Modify the `Make/files` file to contain the new `sixDoFRigidBodyMotionAngle.C` file and change the library location.

Change the first and last line in the `Make/files` file to match Code 2.3

Code 2.3: pointDiscplacement

```
sixDoFRigidBodyMotion/sixDoFRigidBodyMotionAngle.C

...
...
...

LIB = $(FOAM_USER_LIBBIN)/libsixDoFRigidBodyMotionAngle
```

Compile the library with

```
wmake
```

To test the modified sixDoFRigidBodyMotion solver we can use the floatingObject tutorial. Copy the tutorial to your `$FOAM_TUTORIALS` directory or use one of the previous tutorials from Chapter 1.

```
cp -r $FOAM_TUTORIALS/multiphase/interDyMFoam/ras/floatingObject/ $FOAM_RUN
cd $FOAM_RUN
cd floatingObject/constant
```

Change the `dynamicMeshDict` file to use the modified sixDoFRigidBodyMotion solver by adding `Angle` to the end as Code 2.4 shows.

Code 2.4: dynamicMeshDict

```
motionSolverLibs    ("libsixDoFRigidBodyMotionAngle.so");

solver              sixDoFRigidBodyMotionAngle;

sixDoFRigidBodyMotionAngleCoeffs
```

Run the tutorial and look at the output from the `interDyMFoam` solver. The output should now contain the Euler angle (in degrees) as Code 2.5.

Code 2.5: log.interDyMFoam output

```
6-DoF rigid body motion
    Centre of rotation: (-0.5 0.449998563721 0.1)
    Centre of mass: (-0.500001532453 0.449998563721 0.349999999995)
    Orientation: (0.999999999981 0 -6.1298114399e-06 0 1 0 6.1298114399e-06 0 0.999999999981)
    Linear velocity: (0 -5.227784072e-05 0)
    Angular velocity: (0 -0.000462275194671 0)
    Euler angles: (0 -0.000351212324719 0)
```

Chapter 3

Discussion

The different sixDoFRigidBodyMotion solver gives the user an easy possibility to implement several uncoupled or rigidly coupled rigid bodies. Another possibility when using the uncoupled implementation is to have a prescribed motion on one boundary while having sixDoFRigidBodyMotion on another patch.

This way of prescribing the boundaries has the limitation of only having one type of motion per boundary. For instance prescribing a motion and sixDoF simultaneously to one body is not possible without modification to the source code. This type of implementation could be used to simulate the transient behaviour of a boat by prescribing a motion to the rudder while having the entire boats movement determined by the sixDoFRigidBodyMotion solver.

3.1 Future work

Future work can be focused on the implementation of a prescribed motion together with sixDoFRigidBodyMotion for the same body. The body could be regulated if the prescribe motion is a function of a property of the body or fluid. This could for instance be a simple proportional regulation of the hydrofoil angle to get the desired angle of the boat.

One could take this a step further by incorporating a more advanced regulation such as a proportional-integral-derivative controller (PID controller). An example of where such a implementation can be useful is regulating an airfoil angle towards a desired performance in one simulation instead of running several simulations where the airfoil angle is changed in increments.

Chapter 4

References

[1] Mesh motion sixDoF, http://www.openfoam.org/version2.3.0/mesh-motion.php available 2015-12-11
[2] Erik Ekedahl, 6-DOF VOF-solver without Damping in OpenFOAM, PhD course in CFD with OpenSource software, 2009
[3] Andreu Oliver González, Mesh motion alternatives in OpenFOAM, PhD course in CFD with OpenSource software, 2009

Study questions

- Where do we specify the physical properties of a rigid body when using the coupled sixDoFRigidBodyMotion solver?
- What does the `innerDistance` and `outerDistance` options do in the `constants/dynamicMeshDict` file?
- What is important to think of when setting the `innerDistance` for a coupled rigid body?
- What utility do we use in order to convert a 3D mesh into a 2D mesh?
- How much smaller or larger will a cell get if the refinement level is increased by 1.