CFD WITH OPENSOURCE SOFTWARE

A COURSE AT CHALMERS UNIVERSITY OF TECHNOLOGY
TAUGHT BY HÅKAN NILSSON

Study Questions and Answers:

Transient simulation of opening and closing guide vanes of a hydraulic turbine

---

Developed for OpenFOAM-2.4.x

Author:
ABHISHEK SARAF

Peer reviewed by:
THEJESHWAR SADANANDA
HÅKAN NILSSON

January 21, 2016

Learning Outcomes

The reader will learn:

- How to implement a new mesh motion library to give rotational motion.
- The basics of Rodrigues rotation.
- How to set up the case to perform rotating motion.
- How to create a new dynamic mesh class combining solidBodyMotionFvMesh class and dynamicRefineFvMesh class.

Chapter 1

Guide vane rotation test case

1.1 Objective

This report is aimed to present an investigation and capability findings of OpenFOAM-2.4x in handling transient flows when there is some mesh deformation in the computational domain.

1.2 Introduction

In the case of hydraulic turbines the inflow to the rotor is controlled by opening or closing of the guide vanes in order to achieve the optimal efficiency of the turbine. In this case an investigation will be carried out on the guide vanes of a Francis turbine where the guide vanes rotate, thus causing pressure and velocity fluctuations both upstream and downstream of the guide vane stage. A two-dimensional representation of the distributor (the housing for all the guide vanes is called the distributor) is shown in the figure 1. In order to check OpenFOAM's dynamic mesh handling capability each guide vane will be given a certain rotation about their respective centres which will require adapting the grid at each time step.

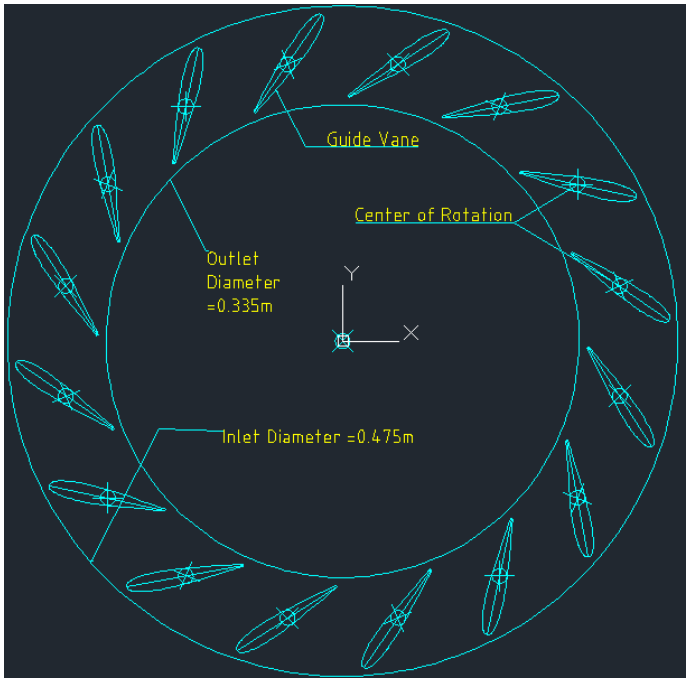

Figure 1: 2-D Schematic diagram of the distributor

This tutorial describes how to modify and use the mesh motion libraries in OpenFOAM to give rotation to each guide vane. In total, there are 16 guide vanes in this case and we shall explore the use of the transient incompressible flow solver pimpleDyMFoam to estimate the pressure distribution on each guide vane as they rotate during the simulation. The moment acting on each guide vane is also calculated. For convenience of the reader, the main dimensions of the distributor are listed in table 1.

Name	Units	Dimension
Inlet Diameter	m	0.475
Outlet Diameter	m	0.335
Guide Vane Height	m	0.104
Pitch Circle Diameter	m	0.400

Table 1: Basic dimensions of the distributor

1.3 Pre-processing

This section covers the necessary steps needed to setup the guide vane case. It includes the procedure to generate the mesh, implement the rotation of each guide vane as well discusses the necessary boundary conditions.

The case directory “TME205_asaraf” must be downloaded from the course homepage and copied to the to the reader’s local run directory of OpenFOAM 2.4.x. The case directory should then be unpacked.

1.3.1 Meshing

The geometry shown in figure 1 consists of 16 guide vanes equally spaced in an annular manner. The initial mesh is generated for one guide vane passage and subsequently OpenFOAM utilities such as “transformPoints”, “mergeMeshes” and “stitchMesh” are used to generate the mesh for the entire distributor.

In the tutorial directory “/TME205_asaraf/Guidevanerotation”, the “Test” directory will be used as a base case to carry out operations to achieve the desired mesh. In order to visualize the patch names and the mesh for one guide vane passage the following command much be entered in the terminal.

```
cd $FOAM_RUN/TME205_asaraf/Guidevanerotation
blockMesh -case Test
```

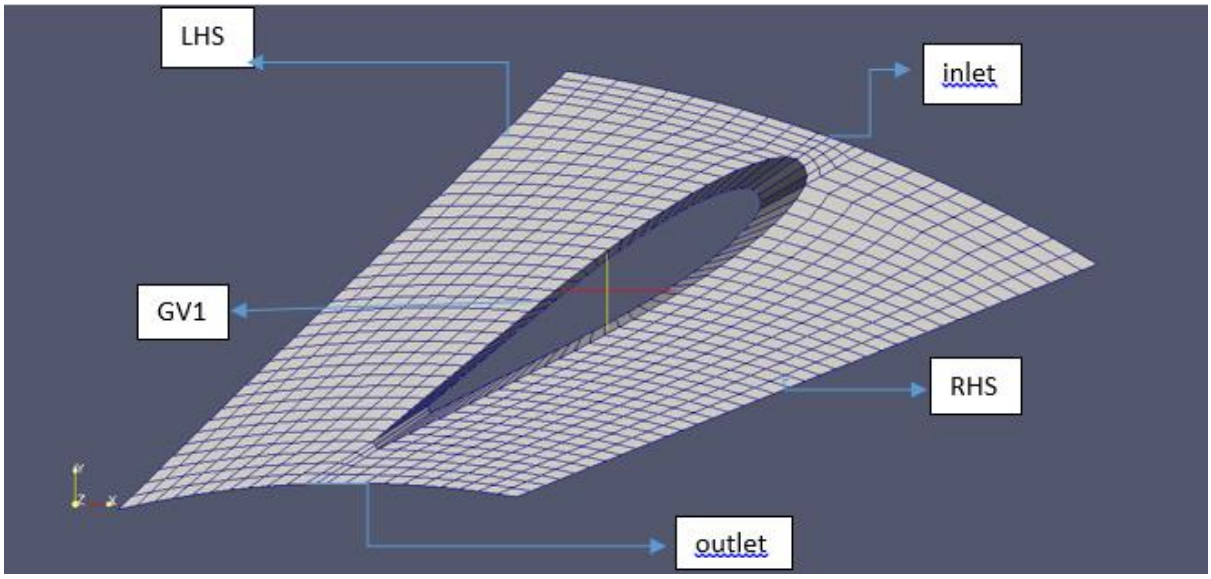

Figure 2: Initial mesh of one Guide Vane passage

The initial mesh shown in the figure 2 is a 2D mesh. The top and bottom patches have been set to type “empty” and hence are not labelled in the figure 2. In view that individual rotation is to be given to each guide vane, the names of each of the guide vane patches are to be given as (GV1,GV2,GV3,GV4.....GV16). Since there are 16 guide vanes passages that will form the complete mesh of the distributor, the mesh generated in the previous step for one guide vane passage is rotated 16 times in order to get the complete geometry of the distributor. This step is performed using the “transformPoints” utility. In order to merge and stitch the rotated meshes, master and slave patches need to be defined for each of the 16 guide vane passages. Hence, the patch names LHS and RHS are renamed as (LHS1,LHS2,LHS3,LHS4.....LHS16) and (RHS1,RHS2,RHS3,RHS4.....RHS16) respectively. After renaming the patches the “mergeMeshes” and “stitchMesh” utilities are used to generate the mesh of the distributor.

The above mentioned steps for meshing are consolidated in a script file namely “meshgeneration” which is located in “/TME205_asaraf/Guidevanerotation” directory. The mesh generation script carries out all the steps required to generate the mesh. In order to make the script executable, the following command line input must be given.

```
chmod +x meshgeneration
```

The mesh is now generated by executing the following command in the “/TME205_asaraf/Guidevanerotation” directory.

```
./meshgeneration
```

The mesh generated using the “meshgeneration” script is shown in figure 3. At this stage the complete mesh of the distributor is generated, but the boundary file located in the “/Test/constant/polyMesh” directory contains the master and slave patches and need to be removed. The modifications to the “/Test/boundary” file need to be done in accordance to what is given in Appendix A. In addition, a reference “boundary” file located in the “Guidevanerotation/readytorun/constant/polyMesh” directory. It can be copied using the following command.

```
cp -r readytorun/constant/polyMesh/boundary Test/constant/polyMesh/boundary
```

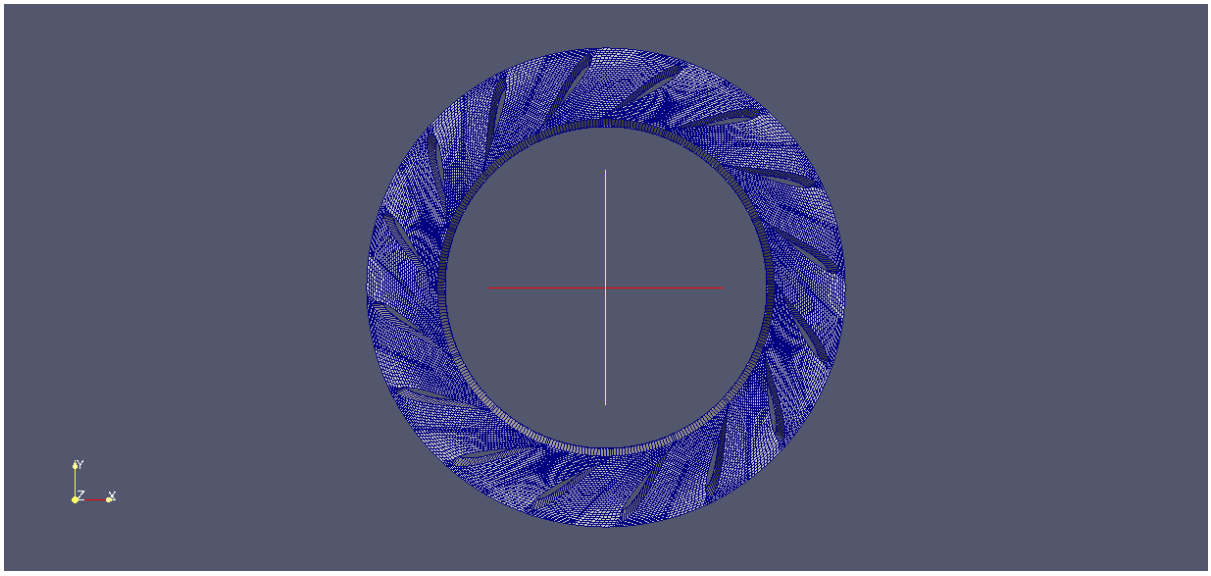

Figure 3: 2D mesh of distributor

1.3.2 Implementing rotation of each guide vane

This section describes the steps to create a new mesh motion library in order to give rotational motion to each of the guide vanes. In this case, the “angularOscillatingVelocity” library is going to be used as a reference. The library chosen here gives oscillating motion by defining velocity of each node on a patch about a fixed axis and centre. This library will be modified, in order to give only rotational motion to the patch. The library can be found at the given location.

```
$FOAM_SRC/fvMotionSolver/pointPatchFields/derived/angularOscillatingVelocity/
```

The above mentioned library is copied to the “TME205_asaraf/Guidevanerotation” directory, using the following commands.

```
cp -r $FOAM_SRC/fvMotionSolver/pointPatchFields/derived\
angularOscillatingVelocity $FOAM_RUN/TME205_asaraf/Guidevanerotation
```

The files and folders are renamed as follows.

```
cd $FOAM_RUN/TME205_asaraf/Guidevanerotation
mv angularOscillatingVelocity librotationVelocity
cd librotationVelocity
mv angularOscillatingVelocityPointPatchVectorField.C\
librotationVelocityPointPatchVectorField.C
mv angularOscillatingVelocityPointPatchVectorField.H\
librotationVelocityPointPatchVectorField.H
```

In order to make a distinction between the original library and this one the string “angularOscillating” in the names of the “.H” and “.C” files is replaced with “librotation”. This replacement has to be done inside these source files. Then the string “angularOscillating” is changed to librotation in the .C- and .H -files as follows.

```
sed -e "s/angularOscillating/librotation/g"\
librotationVelocityPointPatchVectorField.C > tmp.C
mv tmp.C librotationVelocityPointPatchVectorField.C
sed -e "s/angularOscillating/librotation/g"\
librotationVelocityPointPatchVectorField.H > tmp.H
mv tmp.H librotationVelocityPointPatchVectorField.H
```

In order to compile the code, the “Make” directory needs to be created inside the “librotationVelocity” directory. In addition the “Make/files” and “Make/options” files need to be created. This is done using the following commands.

```
mkdir Make
cd Make/
touch files options
```

The “Make/files” and “Make/options” files should be set in the manner as mentioned in code 1 and code 2 respectively.

Code 1: “Make/files” file

```
librotationVelocityPointPatchVectorField.C
LIB = $(FOAM_USER_LIBBIN)/librotationVelocity
```

Note the addition of “USER” in line 2 of code 1, this places the library in the user library directory and makes it impossible for the user to overwrite any original OpenFOAM libraries.

Code 2: “Make/options”

```
EXE_INC = \
        -I$FOAM_SRC/triSurface/lnInclude \
        -I$FOAM_SRC/meshTools/lnInclude \
        -I$FOAM_SRC/dynamicMesh/lnInclude \
        -I$FOAM_SRC/finiteVolume/lnInclude \
        -I$FOAM_SRC/fvMotionSolver/lnInclude

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

At this stage it is advised to test that the library compiles or not in order. This is done using the following command.

```
wmake libso
```

In case there are no compilation errors, the user can proceed with modifying the library as follows.

The declared variables located at line 47 in the “librotationVelocityPointPatchVectorField.H” file are as follows.

```
vector axis_;
vector origin_;
scalar angle0_;
scalar amplitude_;
scalar omega_;
pointField p0_;
```

The declared variables need to be modified and set as follows:

```
vector axis_;
vector origin_;
scalar angle0_;
pointField p0_;
```

These variables are used to choose the different settings for the boundary conditions. This is further explained in section 1.3.3.

The next step is to remove the old initialized variables in the constructors of "librotationVelocityPointPatchVectorField.C" file with the variables initialized in the "librotationVelocityPointPatchVectorField.H" file. The changes to the code between line 47 and line 52, line 65 and line 69, line 97 and line 102, line 114 and line 119 in the "librotationVelocityPointPatchVectorField.C" file are shown in code 3, code 4, code 5 and code 6 respectively.

Code 3: Changes in line 47 to line 52 in "librotationVelocityPointPatchVectorField.C" file

```
axis_(vector::zero),
origin_(vector::zero),
angle0_(0.0),
p0_(p.localPoints())
```

Code 4: Changes in line 65 to line 69 in "librotationVelocityPointPatchVectorField.C" file

```
axis_(dict.lookup("axis")),
origin_(dict.lookup("origin")),
angle0_(readScalar(dict.lookup("angle")))
```

Code 5: Changes in line 97 to line 102 in "librotationVelocityPointPatchVectorField.C" file

```
axis_(ptf.axis_),
origin_(ptf.origin_),
angle0_(ptf.angle_),
p0_(ptf.p0_)
```

Code 6: Changes in line 114 to line 119 in "librotationVelocityPointPatchVectorField.C" file

```
axis_(ptf.axis_),
origin_(ptf.origin_),
angle0_(ptf.angle_),
p0_(ptf.p0_)
```

The code 3 initializes the values of the variables namely "axis", "origin", "angle0" and "p0". Here "p0" gives the absolute coordinates of the points on the patch. Code 4 instructs the code to look up the names "axis", "origin" and "angle0" which will be entered by the user.

At this stage, the variables have been updated in the constructors. Now the write functions need to be updated. The write functions are located between line 181 and line 199 in the "librotationVelocityPointPatchVectorField.C" file. These code between these need to be edited and should be set as shown in code 7.

Code 7: Changes in line 181 to line 199 in “librotationVelocityPointPatchVectorField.C” file

```
void librotationVelocityPointPatchVectorField::write
(
    Ostream& os
) const
{
        pointPatchField<vector>::write(os);
        os.writeKeyword("axis")
                << axis_ << token::END_STATEMENT << nl;
        os.writeKeyword("origin")
                << origin_ << token::END_STATEMENT << nl;
        os.writeKeyword("angle0")
                << angle0_ << token::END_STATEMENT << nl;
        p0_.writeEntry("p0", os);
        writeEntry("value", os);
}
```

The main part of this library which defines the movement of the patches is presented between line 162 and line 175 in the “librotationVelocityPointPatchVectorField.C” file. Here, the velocity of each point on a specific patch is calculated for each time step. But since the library copied was meant for producing oscillating motion, some modifications are made to the member functions in order to make this newly created library only implement rotational motion. Hence we start by replacing the old member functions which define motion. The old member function is shown in code 8 and is to be replaced by the updated member function shown in code 9.

Code 8: Old member function in “librotationVelocityPointPatchVectorField.C” file

```
        scalar angle = angle0_ + amplitude_*sin(omega_*t.value());
        vector axisHat = axis_/mag(axis_);
        vectorField p0Rel = p0_ - origin_;
        vectorField::operator=
        (
                (
                p0_ + p0Rel*(cos(angle) - 1)
                + (axisHat ^ p0Rel*sin(angle))
                + (axisHat & p0Rel)*(1 - cos(angle))*axisHat
                - p.localPoints()
                )/t.deltaT().value() ) ;

        fixedValuePointPatchVectorField<vector>::updateCoeffs();
}
```

Code 9: Updated member function in “librotationVelocityPointPatchVectorField.C” file

```
        scalar rotangle = angle0_*((Foam::constant::mathematical::pi)/180)*t.value();
        vector axisHat = axis_/mag(axis_);
        vectorField p0Rel(p0_ - origin_);
        vectorField::operator=
        (
                (
                p0_
                + p0Rel*(cos(rotangle) - 1)
                + (axisHat ^ p0Rel*sin(rotangle))
                + (axisHat & p0Rel)*(1 - cos(rotangle))*axisHat
                - p.localPoints()
                )/t.deltaTValue()
        );
```

The mechanism of the rotating velocity function is shown in the figure 4. In this the origin of rotation is located in the centre of the guide vane.

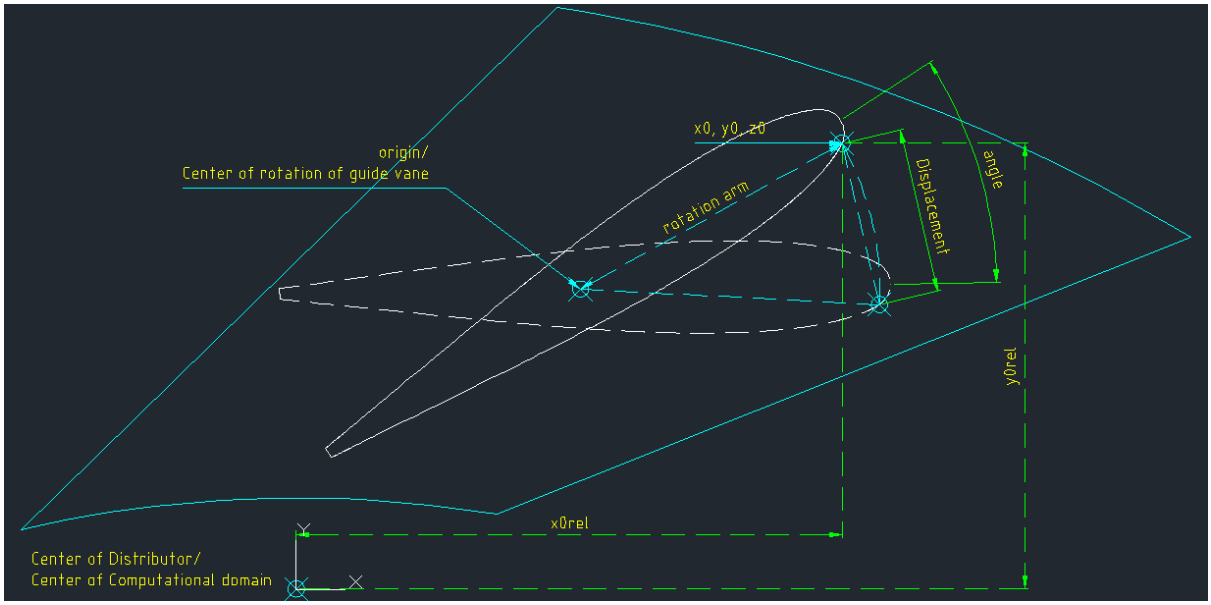

Figure 4: Schematic diagram of mechanism of rotation velocity function

In line 1 of code 9, the angle of rotation namely “angle0” is used to calculate a new angle for the rotation arm. The rotation arm connects the points on the rotating body to the centre of rotation is computed at each time step employing the following equation.

$$rotangle = angle0*(pi/180)*t$$

The angle has been converted to radians as mathematical calculations in OpenFOAM uses radians as units for angles. For this purpose, the following header file needs to be added to the “librotationVelocityPointPatchVectorField.C” file.

```
#include "mathematicalConstants.H"
```

The above mentioned header file is to be added after the “#include “polyMesh.H”” header file. Hence the user will be asked to input the rotation angle in degrees in the boundary condition in the “pointMotionU” file.

As per the updated member function mentioned in code 9, “p0” is the coordinate for the rotating grid point, (x0, y0, z0); “p0Rel” is the relative coordinate for the grids on the moving body with respect to the centre of rotation, that is the origin ; “axisHat” is a unit vector which defines the axis of oscillation.

After computing the new angle (“rotangle”) of rotating arm, updated position of the grids on the moving body is computed by applying the concept of Rodrigues rotation. According to this concept a vector can be rotated by specifying the axis of rotation and the angle by which is should rotate. A basic concept of Rodrigues rotation concept is shown in the figure 5 and followed by a general equation.

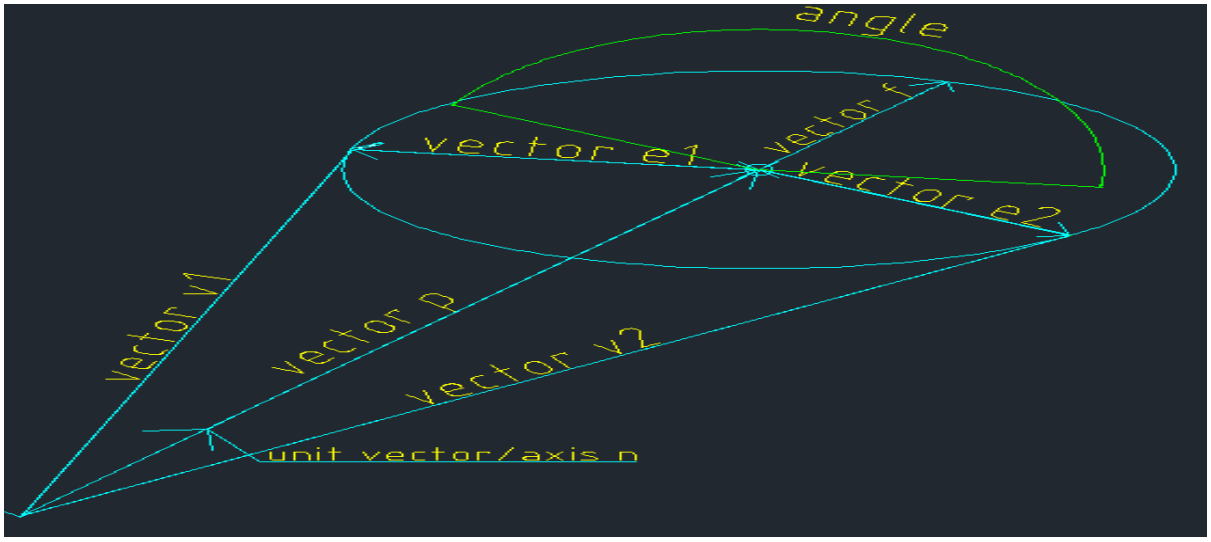

Figure 5: Example of Rodrigues rotation

As per the figure 5.

$$\overrightarrow{v2} = \vec{p} + \overrightarrow{e2}$$

$$\overrightarrow{v2} = (\overrightarrow{v1} \wedge \vec{n})\,\vec{n} + \overrightarrow{e1}\cos(angle) + \vec{f}\sin(angle)$$

$$\overrightarrow{v2} = \overrightarrow{v1}\cos(angle) + (\overrightarrow{v1} \wedge \vec{n})\,\vec{n}(1 - \cos(angle)) + (\vec{n}\,\&\,\overrightarrow{v1})\sin(angle)$$

A similarity can be observed between the above mentioned equation and the vector field operator mentioned in line 15 of code 9, which calculates the position of the rotated point. The updated positions of the moving grids on rotated body are divided by the time step, “deltaT”, which means that the output of this function is a velocity.

Now, the new library can be used to implement rotational motion. To make the current library available for the other applications, it should be compiled through the following command.

```
wmake libso
```

In addition, a reference directory “TME205_asaraf/librotationVelocity” is provided to verify the setup of the newly defined library.

1.3.3 Boundary conditions and case set up

This section discusses the main boundary conditions that need to be set in order to run the tutorial case.

Now that the library has been compiled, the library needs to be made usable during run time. There should be a link to this new library in the “controlDict” file located in the “Test/system/” directory, in order to let the employed applications know about it. Inserting the following line to the “controlDict” file does this.

```
libs ("librotationVelocity.so");
```

In this case the water enters the distributor both radially and tangentially. The water is entering the distributor with a flow rate of $5m^3/s$. The velocity magnitude at the inlet of the distributor is calculated as follows.

Velocity magnitude (U) = Flow Rate/Inlet Area = 5/1.019= 4.92 m/s

In this case the relative flow angle (beta), is given as 37 degrees. Therefore, radial and tangential components of the velocity can be determined using trigonometric relations.

*Radial velocity = U*cos (beta) = 4.9*cos (37) = 3.92m/s*
*Tangential velocity = U*sin (beta) = 4.9*sin (37) = 2.95m/s*
*rpm = Radial velocity/(0.5*0.10472*outer diameter) = 2.95/(0.5*0.10472*0.475) = 120*

The setup of the "0.org/U" file is shown in code 10. Since the flow entering the entering the distributor, is velocity driven, the boundary type for the "inlet" is taken as "cylindricalInletVelocity". The "outlet" is prescribed a Neumann boundary condition that is "zeroGradient". Since this case is in 2D, the "top" and "bottom" patches are set to "empty". The boundary type of "GV1" in the "U" file is specified as "movingWallVelocity". Similarly the boundary conditions are to be set for the remaining 15 guide vanes as per what is shown in Appendix B.

Code 10: "0.org/U" file

```
boundaryField
{
    inlet
    {
        type           cylindricalInletVelocity;
       axis           (0 0 1);
       centre         (0 0 0);
       axialVelocity   0;
       radialVelocity -3.92;
       rpm             120;
       value          uniform (0 0 0);
    }
    outlet
    {
        type           zeroGradient;
    }
    top
    {
        type            empty;
    }

    bottom
    {
        type            empty;

    }
    GV1
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
}
```

The boundary conditions for mesh motion for one of the guide vanes (GV1) is shown in code 11. The "0.org/pointMotionU" file primarily calls the new library that gives velocity at each node of the boundary. Hence the boundary type for "GV1" is specified as "librotationVelocity". The variable "angle0" defines the velocity of grid points in degrees per second. Similarly the boundary conditions are to be set for the remaining 15 guide vanes. The setup of the "0.org/pointMotionU" file should be similar to what is shown in Appendix C.

Code 11: "0.org/pointMotionU" file

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

```
        value           uniform (0 0 0);
    }
    top
    {
        type            empty;
    }
    bottom
    {
        type            empty;
    }
    GV1
    {
        type                librotationVelocity;
        axis                (0 0 1);
        origin              (0.0780350 .3923143 0);// Centre of Rotation
        angle               -3;// Degrees per second
        value               uniform (0 0 0);
    }
}
```

A snippet of the "0.org/p" file for one guide vane is shown in code 12. In code 12, it can be seen that Neumann boundary condition is applied to the "inlet" and "GV1" patches respectively. The boundary conditions are to be set for the remaining 15 guide vanes. The setup of the "0.org/p" file should be similar to what is shown in Appendix D.

Code 12: "0.org/p" file

```
boundaryField
{
    inlet
    {
        type            zeroGradient;
    }
    outlet
    {
       type            fixedValue;
       value uniform   0;
    }
    top
    {
        type            empty;
    }
    bottom
    {
        type            empty;
    }
    GV1
    {
        type            zeroGradient;
    }
}
```

The setup of the "dynamicMeshDict" file located in the "/Test/constant" directory is shown in code 13. The "dynamicMeshDict" file is already setup in the "/Test/constant" directory and code 13 does not need to be implemented and the settings in the code are discussed.

Code 13: "dynamicMeshDict" file

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      dynamicMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dynamicFvMesh   dynamicMotionSolverFvMesh;

motionSolverLibs ("libfvMotionSolvers.so");

solver velocityLaplacian;
velocityLaplacianCoeffs
```

```
{
diffusivity  uniform;
}
```

In this case we are using the dynamic mesh type that is, “dynamicMotionSolverFvMesh” and are using the “velocityLaplacian” solver to solve for motion. It should be noted that, the “velocityLaplacian” solver interprets the motion of the patch points in terms of velocity of moving points.

The forces and moments acting on each guide vane are calculated using the “forces” function object. The “forces” file is located in the “/Test/system” directory. The setup of the “forces” file is shown in code 14, which calculates the forces and moments acting on one guide vane (GV1).

Code 14: “forces” file

```
forces1
{
    type                forces;
    functionObjectLibs  ( "libforces.so" );
    outputControl       timeStep;
    timeInterval        1;
    log                 yes;
    patches             (GV1);
    pName               p;
    UName               U;
    rhoName             rhoInf;     // Indicates incompressible
    log                 true;
    rhoInf              1;          // Redundant for incompressible
    CofR                (0.07804     0.39231 0       );// Axis coordinates
}
```

Similarly, the forces and moments need to be calculated for the remaining 15 guide vanes. The setup of the “forces” file must be similar to what is shown in Appendix E. Additionally, the forces and moments on each of the guide vanes needs to be calculated during run time therefore the force function object needs to be called in “controlDict” file located in the “/Test/system” directory. This is done by adding the following lines at the end of the “controlDict” file.

```
functions
{#include "forces"}
```

It is required to carry out some settings in “fvSolution” file located in the “/Test/system” directory and define the solver for mesh motion application. This is done by adding the following lines to the “solvers” sub-dictionary in the “fvSolution” file. The lines to be added are shown in code 15.

Code 15: “fvSolution” file

```
  cellMotionU
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-08;
        relTol          0;
    }
    cellMotionUx
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-08;
        relTol          0;
    }
```

The complete setup of the “fvSolution” file is located in Appendix F. The solvers in the “fvSolution” file specifies each linear-solver that is used for each discretised equation. The syntax for each entry within solvers uses a keyword that is the word relating to the variable being solved in the particular equation. The choices for “solvers” are presented in Table 2.

Solver	Keyword
Preconditioned (bi-)conjugate gradient	PCG/PBiCG*
Solver using a smoother	smoothSolver
Generalised geometric-algebraic multi-grid	GAMG
*PCG for symmetric matrices, PBiCG for asymmetric	

Table 2. Solver Options

There is a range of options for preconditioning of matrices in the conjugate gradient solvers, represented by the preconditioner keyword in the solver dictionary. The preconditioners are listed in Table 3.

Preconditioner	Keyword
Diagonal incomplete-Cholesky (symmetric)	DIC
Faster diagonal incomplete-Cholesky (DIC with caching)	FDIC
Diagonal incomplete-LU (asymmetric)	DILU
Diagonal	diagonal
Geometric-algebraic multi-grid	GAMG
No preconditioning	none

Table 3. Preconditioner options

1.3.4 Running the application

The main steps for the setup of this tutorial have been enumerated above. A reference directory namely "TME205_asaraf/Guidevanerotation/readytorun" has been provided which can be used to verify the setup of this tutorial. In the "TME205_asaraf/Guidevanerotation/Test" directory the following commands are executed to run the case.

```
mv 0.org 0
pimpleDyMFoam
```

1.3.5 Results

The results of the simulation are visualized using "paraview" and shown in figure 6 and figure 7. In this case each guide vane has alternate rotation. It means that if guide vane1 rotates in counter clockwise direction then the guide vanes adjacent to it rotate in the clockwise direction.

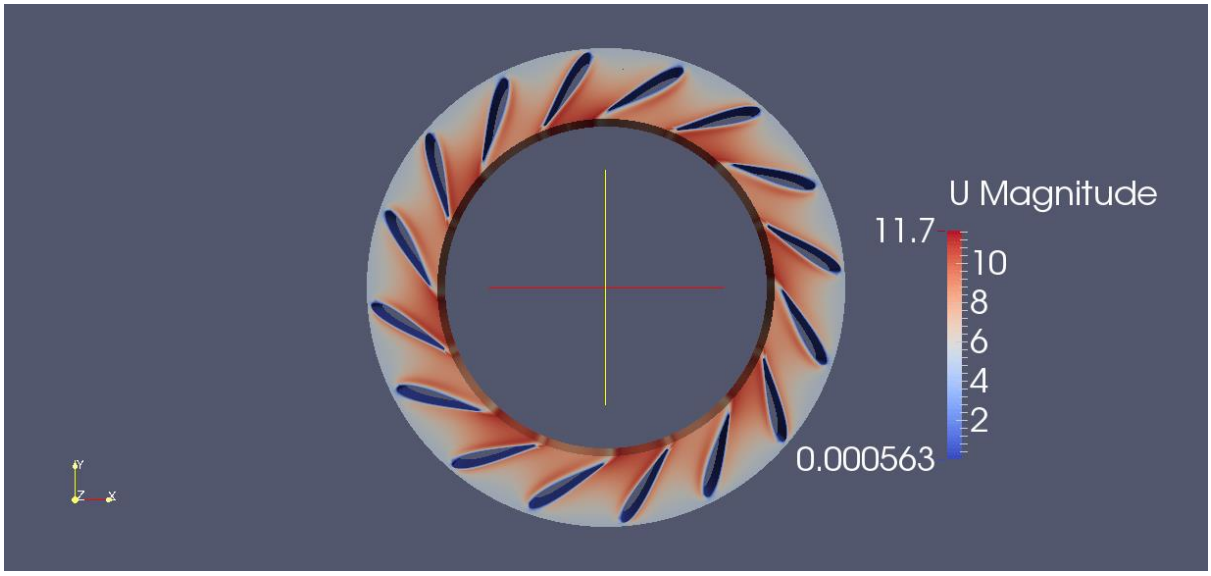

Figure 6: Velocity surface contours at time=1second and total rotation of 3 degrees

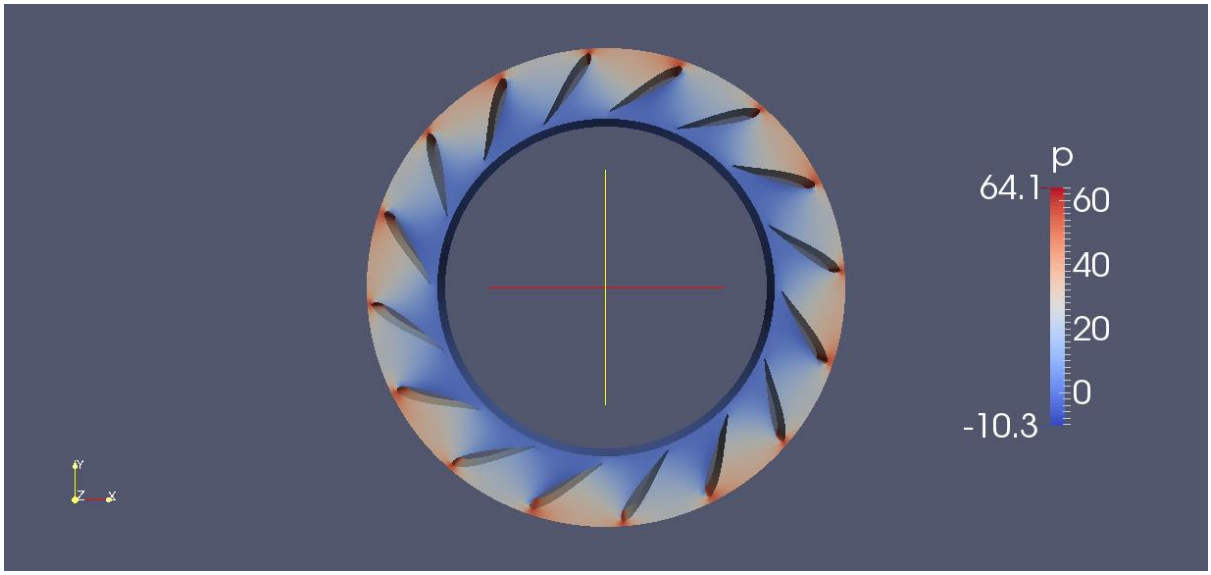

Figure 7: Pressure Surface Contours time=1second and total rotation of 3 degrees

After the simulation is complete the moments on each of the guide vanes will be calculated and stored in the "Test/postProcessing" directory. There will be 16 sub directories namely "forces1, forces2, forces3….forces16" inside the "Test/postProcessing" directory. For example, the data for guide vane1 will be stored in the "Test/postProcessing/forces1/0" directory in the "forces" file. The data from this file cannot to used directly and needs to be rearranged in order to extract the moments. This is done by using a python script namely "new.py" which is located in the "Test" directory. The python script will give the output in the form of a ".txt" which has the simulation time in column 1, the lift forces in column 2, the drag forces in column 3 and the moments in column 4. This ".txt" file can be used to plot the data for the guide vane. The python script needs to be executed in the "Test" directory as follows.

```
./new.py
```

In case the user wants to extract data for each of the guide vanes, the user will have to specify the path of the "postProcessing/forces1/0/forces" file in line 7 of the "new.py" script. In addition, the name of the output ".txt" file can be controlled by changing the name specified in the line 48 of the "new.py" script. The "new.py" script is attached in the Appendix G.

The moments acting on each guide vane is calculated with respect to their centre of rotations is shown in figure 8.

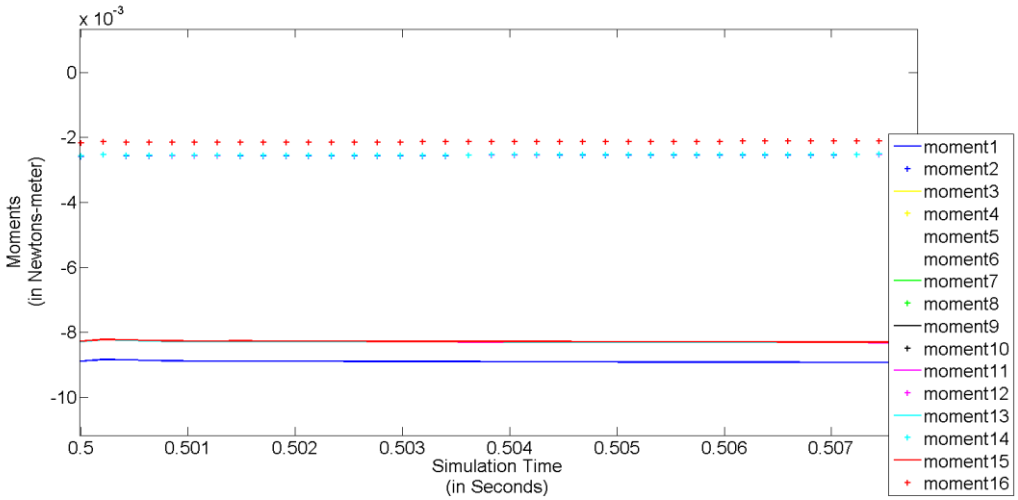

Figure 8: Moments versus simulation time

Chapter 2

Alternate Finding - Building a new dynamic mesh class

This section discusses the process of building of a new dynamic mesh class, which combines solid body motion and adaptive hexahedral mesh refinement. The “solidBodyMotionFvMesh” class gives linear and rotational motion to the mesh whereas the “dynamicRefineFvMesh” class refines the mesh based on a scalar field value. This new dynamic mesh class cannot be used for the above mentioned guide vane case as there is no scalar transport quantity which can be used as a basis to refine the mesh. Hence in this section the procedure to construct a new dynamic mesh class will be discussed and a tutorial will be executed to test the validity of the new dynamic mesh class.

2.1 Background

The “dynamicFvMesh” is an abstract class of the “fvMesh” class. Hence, while combining two dynamic mesh classes namely solidBodyMotionFvMesh and dynamicRefineFvMesh, multiple inheritance cannot be used as it would lead to calling the constructor of fvMesh twice and would cause multiple meshes to be allocated. A simplified class diagram of dynamic mesh classes is shown in figure 9.

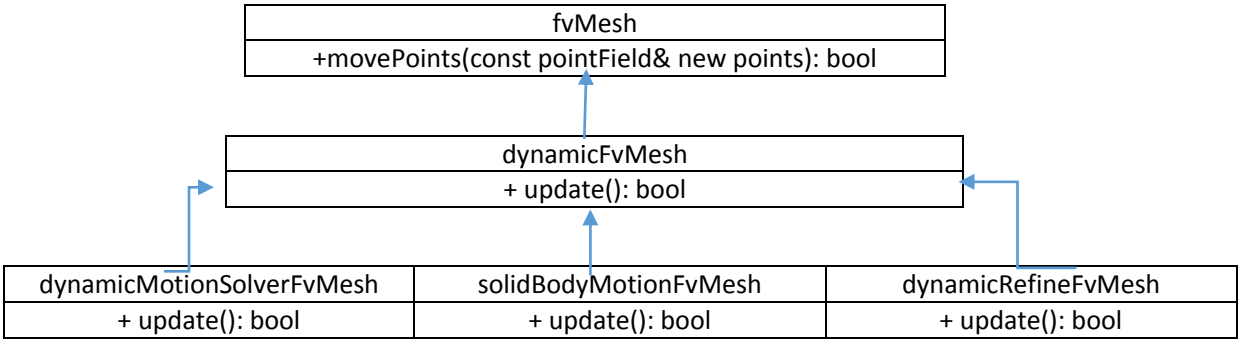

Figure 9: Class diagram for dynamic mesh classes

Therefore we proceed by combining the dynamic mesh classes by using the concept of virtual inheritance as shown in figure 10.

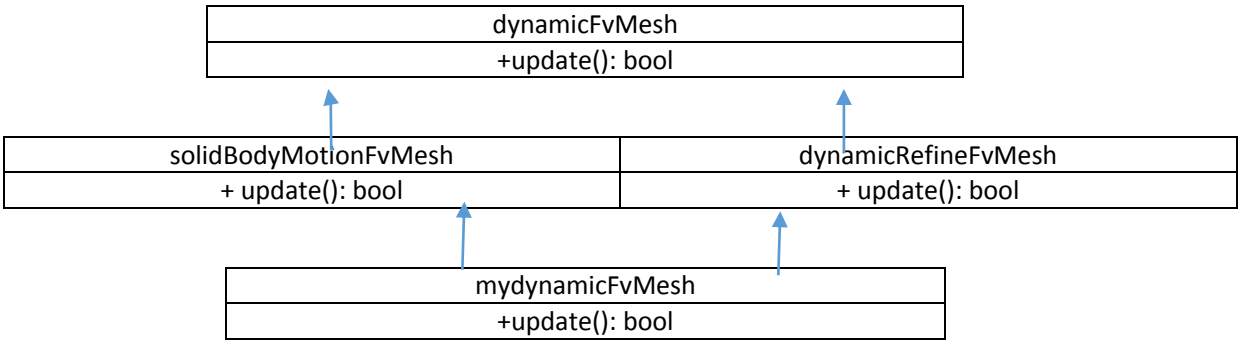

Figure 10: Class diagram of new dynamic mesh class

Upon investigation into the source codes of solidBodyMotionFvMesh and dynamicRefineFvMesh, it is observed that it is easier to rebuild the solidBodyMotionFvMesh than to rebuild the dynamicRefineFvMesh which is very complex. Therefore, the new class “mydynamicFvMesh” will inherit from dynamicRefineFvMesh and is chosen as the base class and some elements from solidBodyMotionFvMesh are reused.

2.2 Implementing a new dynamic mesh class

The first step is to create a new directory at the location "$FOAM_RUN/TME205_asaraf/meshrefinetest"

```
cd $FOAM_RUN/TME205_asaraf/meshrefinetest
mkdir mydynamicFvMesh
cd mydynamicFvMesh
```

We will now generate the class files as follows:

```
foamNew source C mydynamicFvMesh
foamNew source H mydynamicFvMesh
```

In order to compile this new library, Make/files and Make/options need to be created.

```
mkdir Make
cd Make
touch files options
```

The "Make/files" file should be setup as follows.

```
mydynamicFvMesh.C
LIB = $(FOAM_USER_LIBBIN)/mydynamicFvMesh
```

The "Make/options" file should be setup as follows.

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

EXE_LIBS = \
-lfiniteVolume \
-ldynamicMesh \
-ldynamicFvMesh
```

The newly created "mydynamicFvMesh.C" and "mydynamicFvMesh.H" files need to be cleaned up and setup as shown in code 16 and code 17 respectively.

Code 16: "mydynamicFvMesh.C" file

```
#include "mydynamicFvMesh.H"
// * * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * * //
Foam::mydynamicFvMesh::mydynamicFvMesh()
{}
// * * * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * * * * //
Foam::mydynamicFvMesh::~mydynamicFvMesh()
{}
// * * * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * * * * //
```

Code 17: "mydynamicFvMesh.H" file

```
#ifndef mydynamicFvMesh_H
#define mydynamicFvMesh_H
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
namespace Foam
{
class mydynamicFvMesh
:
public
{   //- Dictionary of solid body motion control parameters

    //- The motion control function

    //- The reference points which are transformed

public:
     // Runtime type information

    // Constructors
```

```
    // Construct from objectRegistry, and read/write options
       mydynamicFvMesh();

    // Destructor
    ~mydynamicFvMesh();

    // Member Functions

};
} // End namespace Foam
#endif
```

Now in the “mydynamicFvMesh.C” and “mydynamicFvMesh.H” files, the dynamicRefineFvMesh is inherited into the code. The modified part of the “mydynamicFvMesh.H” file is shown in code 18. In code 18, the header file “#include "dynamicRefineFvMesh.H"” is added to the code. In addition, in line 10 of code 18, the “dynamicRefineFvMesh” is defined as a public variable.

Code 18: “mydynamicFvMesh.H” file

```
#ifndef mydynamicFvMesh_H
#define mydynamicFvMesh_H
#include "dynamicRefineFvMesh.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
namespace Foam
{
class mydynamicFvMesh
:
public dynamicRefineFvMesh
```

According to figure 10, the new dynamic mesh class is to be derived using virtual inheritance, hence in the “mydynamicFvMesh.H” file the following changes need to be made to the destructor. The changes are shown in code 19.

Code 19: “mydynamicFvMesh.H” file

```
// Destructor
    virtual ~mydynamicFvMesh();
```

Now the global static variable is declared in the “mydynamicFvMesh.H” file. It is shown in code 20.

Code 20: “mydynamicFvMesh.H” file

```
public:
// Runtime type information
TypeName ("mydynamicFvMesh");
```

After declaring the type name, the corresponding header file is added which will allow the code to use “TypeName”. The addition made to the “mydynamicFvMesh.H” file is shown in code 21.

Code 21: “mydynamicFvMesh.H” file

```
#include "dynamicRefineFvMesh.H"
#include "typeInfo.H"
```

Now in the “mydynamicFvMesh.C” file, the static type name variable and debug switches are defined. This is shown in line 5 of code 22. In addition to make new dynamic mesh class usable at run time it needs to be added to the run time selectable table. This is done by line 6 in code 22 and the corresponding header file “#include "addToRunTimeSelectionTable.H"” is added as shown in line 2 of code 22.

Code 22: “mydynamicFvMesh.H” file

```
#include "mydynamicFvMesh.H"
#include "addToRunTimeSelectionTable.H"
// * * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * * * //
namespace Foam {
        defineTypeNameAndDebug(mydynamicFvMesh, 0);
        addToRunTimeSelectionTable(dynamicFvMesh, mydynamicFvMesh, IOobject);
}
```

The class is made usable by declaring the update member function. The additions to the “mydynamicFvMesh.H” file and “mydynamicFvMesh.C” file are shown in code 23 and code 24 respectively.

Code 23: “mydynamicFvMesh.H” file

```
// Member Functions
virtual bool update();
```

Code 24: “mydynamicFvMesh.C” file

```
// * * * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * //

bool Foam::mydynamicFvMesh::update()
{
dynamicRefineFvMesh::update();
return true ;
}
```

The constructors in the “mydynamicFvMesh.H” and “mydynamicFvMesh.C” files are modified to create an interface of the new class that will ask the code to follow the “dynamicRefineFvMesh” class. This is shown in the code 25 and code 26 respectively.

Code 25: “mydynamicFvMesh.H” file

```
// Construct from objectRegistry, and read/write options
explicit mydynamicFvMesh(const IOobject& io);
```

Code 26: “mydynamicFvMesh.C” file

```
// * * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * * //
Foam::mydynamicFvMesh::mydynamicFvMesh(const IOobject& io)
:
        dynamicRefineFvMesh(io)
{}
```

The solid body motion needs to be added to the “mydynamicFvMesh” class. For the newly created class, only a few attributes of the “solidBodyMotionFvMesh” class will be used. The source files of “solidBodyMotionFvMesh” class is located in the “$FOAM_SRC/dynamicFvMesh/solidBodyMotionFvMesh” directory. Note that, the new class will not be able to employ cell subset motion. The constructor of the source “solidBodyMotionFvMesh.C” file is reused, specifically the code from line 50 to line 77. This constructor will aid in adding solid body motion coefficients to the class. The constructor for the new class should be modified as shown in code 27.

Code 27: “mydynamicFvMesh.C” file

```
// * * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * * * //
Foam::mydynamicFvMesh::mydynamicFvMesh(const IOobject& io)
:
        dynamicRefineFvMesh(io),
        motionCoeffs_
        (
         IOdictionary
         (
                IOobject
                (
                 "dynamicMeshDict",
                 io.time().constant(),
                 *this,
                 IOobject::MUST_READ_IF_MODIFIED,
                 IOobject::NO_WRITE,
                 false
                )
            ).subDict(typeName + "Coeffs")
    ),
    SBMFPtr_(solidBodyMotionFunction::New(motionCoeffs_, io.time())),
    undisplacedPoints_
    (
        IOobject
        (
            "points",
            io.time().constant(),
            meshSubDir,
            *this,
            IOobject::MUST_READ,
            IOobject::NO_WRITE,
            false
        )
    )
{}
```

The “mydynamicFvMesh.H” file needs to be modified to allow the user to input solid body motion coefficients in the “dynamicMeshDict” file. These modifications are done between line 13 and line 20. In addition the “#include "solidBodyMotionFunction.H"” header file is added and shown in line 2 of code 28.

Code 28: “mydynamicFvMesh.H” file

```
#include "dynamicRefineFvMesh.H"
#include "solidBodyMotionFunction.H"
#include "typeInfo.H"
#include "dictionary.H"
#include "pointIOField.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
namespace Foam
{
class mydynamicFvMesh
:
    public dynamicRefineFvMesh
{
    //- Dictionary of solid body motion control parameters
    const dictionary motionCoeffs_;

    //- The motion control function
    autoPtr<solidBodyMotionFunction> SBMFPtr_;

    //- The reference points which are transformed
    pointIOField undisplacedPoints_;
```

Now the update member function for the new dynamic mesh class needs to be defined. The code between the line 90 and the line 113 in the source file “solidBodyMotionFvMesh.C” is used to define the solid body motion. Some parts of the above mentioned source code will be incorporated of this dynamic mesh class. The line 7 in the code 29 is added so that the points of the mesh can be moved and also ensure that the motion points are synchronized with the new mesh points generated after mesh refinement.

Code 29: “mydynamicFvMesh.C” file

```
// * * * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * * * * //
bool Foam::mydynamicFvMesh::update()
{
dynamicRefineFvMesh::update();
undisplacedPoints_ = this->points();
    static bool hasWarned = false;
    fvMesh::movePoints
    (
        transform
        (
            SBMFPtr_().transformation(),
            undisplacedPoints_
        )
    );
    if (foundObject<volVectorField>("U"))
    {
        const_cast<volVectorField&>(lookupObject<volVectorField>("U"))
            .correctBoundaryConditions();
    }
    else if (!hasWarned)
    {
        hasWarned = true;
        WarningIn("solidBodyPointMotionSolver::update()")
            << "Did not find volVectorField U."
            << "Not updating U boundary conditions." << endl;
    }
return true ;
}
```

As a final step, two more header files need to be included in the “mydynamicFvMesh.C” file. They are to be added after the “#include "addToRunTimeSelectionTable.H"” header file. They are shown in code 30.

Code 30: “mydynamicFvMesh.C” file

```
#include "volFields.H"
#include "transformField.H"
```

Now the library can be compiled using the command

```
wmake libso
```

A reference directory namely “meshRefine” located in the “/TME205_asaraf” directory and can be used to debug the code in case there are errors during the compilation.

2.3 Running the mesh refinement tutorial case

The main objective of this tutorial is to test the newly defined dynamic mesh class and verify whether adaptive mesh refinement occurs along with the solid body motion. This tutorial will be executed using the “interDyMFoam” solver as it has dynamic mesh handling capabilities. The new dynamic mesh class will be tested using the tutorial “Test” located in the “TME205_asaraf/meshrefinetest” directory.

2.3.1 Meshing

The geometry is a cube of dimensions 1m x 1m x 2m. The mesh is generated using “blockMesh”. The “blockMeshDict” file is located in the “meshrefinetest/Test/constant/polyMesh”. The mesh generated is shown in figure 11. The “blockMeshDict” file is set as per what is mentioned in Appendix H.

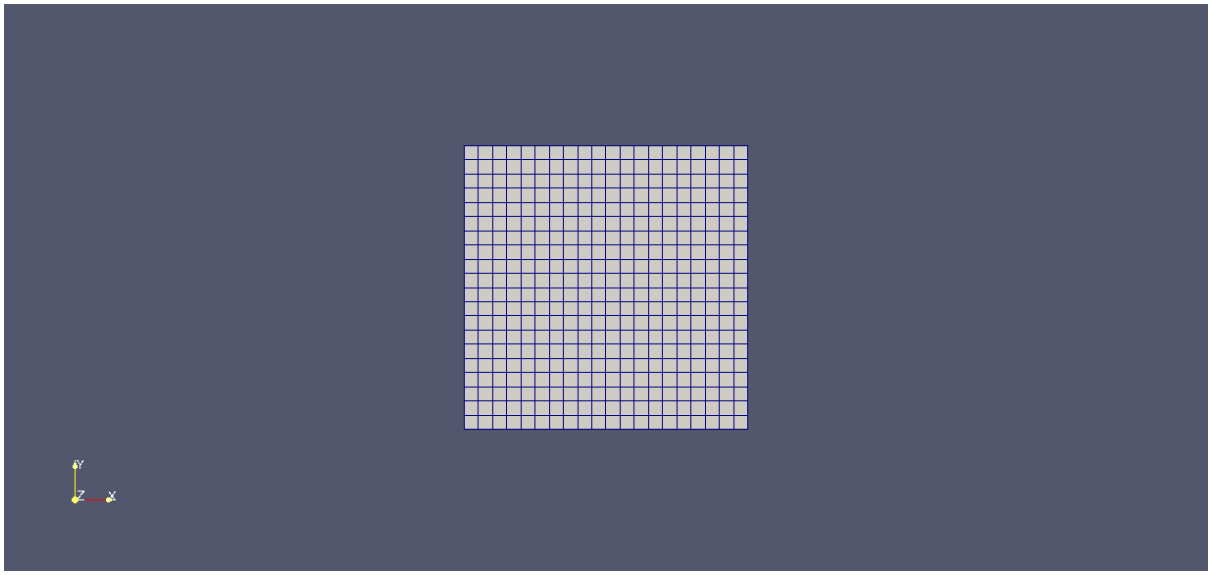

Figure 11: Hexahedral mesh of the test case

In this case since we are conducting adaptive mesh refinement we must specify some scalar field like volume fraction which can be used as a basis for conducting mesh refinement. In this tutorial, the phase is referred to as “alpha.phase1”. The volume fraction of “alpha.phase1” is set using the “setFieldsDict” dictionary located in the “meshrefinetest/Test/system” directory. The file is already setup in the tutorial, but the settings for the “setFieldsDict” are shown in the code 31 to discuss how the scalar field is introduced. As per the code 31, the “alpha.phase1” is located inside the computational domain and occupies a volume equivalent to a sphere of radius 0.15m.

Code 31: “setFieldsDict” file

```
defaultFieldValues
(
    volScalarFieldValue alpha.phase1 0
);

regions
(
    sphereToCell
    {
        centre (0.5 0.5 0.5);
        radius 0.15;
        fieldValues ( volScalarFieldValue alpha.phase1 1 );
    }
);
```

In regard to the new dynamic mesh class created, both the solid body motion as well as adaptive mesh refinement coefficents are specified in the “dynamicMeshDict” file. The “dynamicMeshDict” file is located in the “/meshrefinetest/Test/constant” directory. The setup of the “dynamicMeshDict” dictionary is shown in code 32. Since the file is already provided in the tutorial, the code 32 is shown just to discuss the properties of the “dynamicMeshDict” file pertaining to this case.

It can be seen in the code 32 that the user has entered the “dynamicFvMesh” type as “mydynamicFvmesh” that allows the user to supply both solid body motion coefficients as well as adaptive mesh refinement coefficients. The solid body motion coefficients suggest that the entire mesh is having a linear motion in the negative z-direction with a velocity of -0.1m/s.

Code 32: “dynamicMeshDict” file

```
dynamicFvMesh mydynamicFvMesh;
mydynamicFvMeshCoeffs
{
    solidBodyMotionFunction linearMotion;

    linearMotionCoeffs
    {
        velocity (0 0 -0.1);
    }
}
dynamicRefineFvMeshCoeffs
{
    // How often to refine
    refineInterval  1;
    // Field to be refinement on
    field           alpha.phase1;
    // Refine field inbetween lower..upper
    lowerRefineLevel 0.0001;
    upperRefineLevel 0.999;
    // If value < unrefineLevel unrefine
    unrefineLevel   10;
    // Have slower than 2:1 refinement
    nBufferLayers   3;
    // Refine cells only up to maxRefinement levels
    maxRefinement   1;
    // Stop refinement if maxCells reached
    maxCells        100000;
    // Flux field and corresponding velocity field. Fluxes on changed
    // faces get recalculated by interpolating the velocity. Use 'none'
    // on surfaceScalarFields that do not need to be reinterpolated.
    correctFluxes
    (
      (phi none)
      (nHatf none)
      (rhoPhi none)
      (ghf none)
      (phiAlpha none)
    );

    // Write the refinement level as a volScalarField
    dumpLevel       false;
}
// ************************************************************************* //
```

In the “dynamicRefineFvmesh” coefficients, the field is chosen as “alpha.phase1” which is the volume fraction of the scalar field. Therefore, the refinement based on the scalar field value of “alpha.phase1” will be restricted by setting the “lowerRefineLevel” and “UpperRefineLevel” values of the volume fraction of “alpha.phase1”. The other entries are used to determine as to what refinement level is desired. One must note that “dynamicRefineFvMesh” works by monitoring the scalar field values and then merges and splits the cells in order to refine the mesh. The line 31 to line 37 in code 32 deals with recalculation of fluxes after the refinement takes place. In this case, since the main concern is with implementing the new dynamic mesh class the “correctFluxes” are all set to “none”.

2.3.2 Running the tutorial

In order to make the new dynamic mesh class usable during the run time, the following line must be added to the “controlDict” file located in the “meshrefinetest/Test/system” directory.

```
libs (“mydynamicFvMesh.so”);
```

The case is supplied with an “Allrun” script in the “meshrefinetest/Test” directory which performs all the operations to run the case. It is executed as follows.

```
./Allrun
```

2.3.3 Results

The results of the simulation can be visualized in “paraview”. The total simulation time for the tutorial is 0.25 seconds. The result of the simulation are shown in figure 12, figure 13 and figure 14.

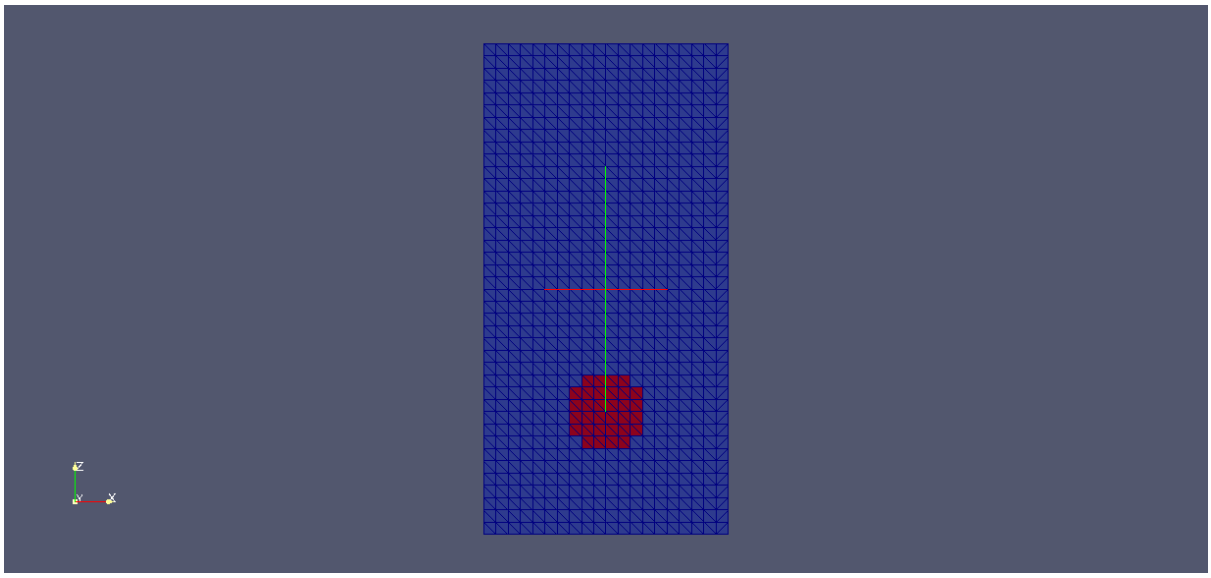

Figure 12: Initial mesh at time = 0 seconds

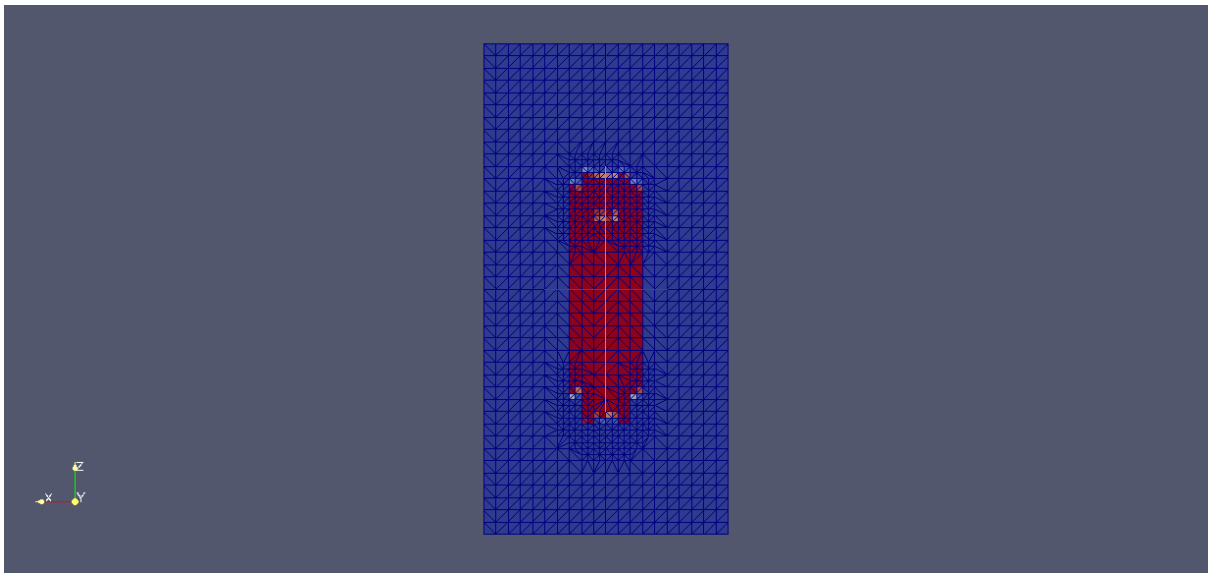

Figure 13: Mesh motion and refinement at time = 0.15 seconds

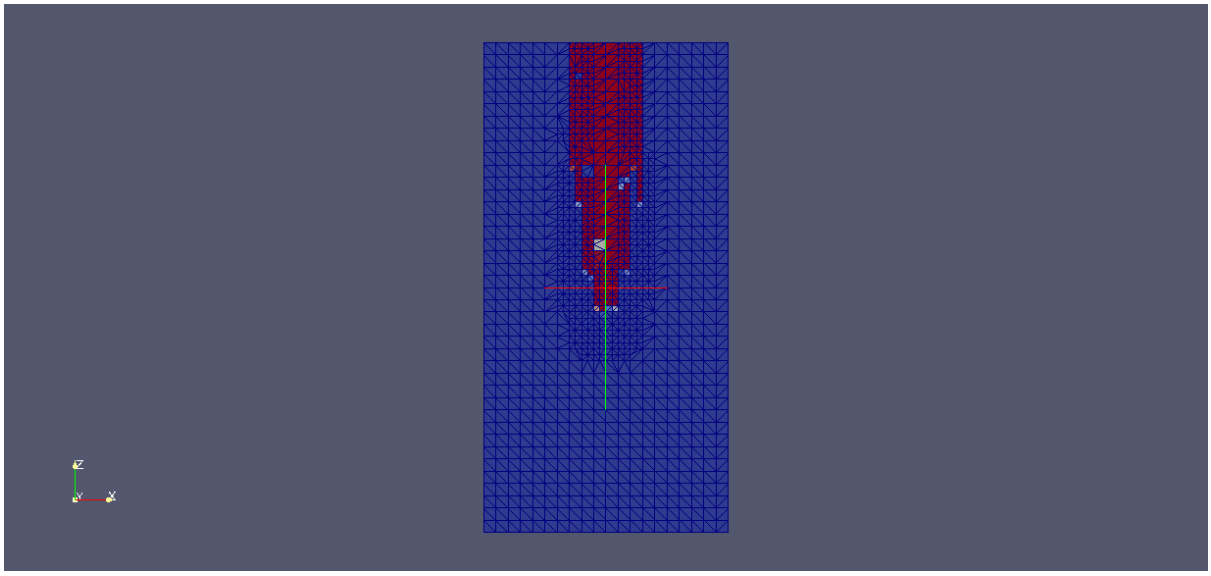

Figure 14: Mesh motion and refinement at time = 0.25 seconds

Study Questions

- If we need to run the mesh motion library we have created, what line of code should be added and where?
- How do we run the “forces” function object during run time? What line of code should be added and where?
- In the new dynamic mesh class that has been created, is cell subset motion possible?
- In the new dynamic mesh class, what is the importance of using virtual inheritance?

Appendix A: "boundary" file for the Guide vane case

Location of the file:

`TME205_asaraf/Guidevanerotation/Test/constant/polyMesh`

The following lines of code are employed in this tutorial:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       polyBoundaryMesh;
    location    "0.001/polyMesh";
    object      boundary;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
20
(
    inlet
    {
        type            patch;
        nFaces          384;
        startFace       24720;
    }
    outlet
    {
        type            patch;
        nFaces          352;
        startFace       25104;
    }
    top
    {
        type            empty;
        inGroups        1(empty);
        nFaces          12544;
        startFace       25456;
    }
    bottom
    {
        type            empty;
        inGroups        1(empty);
        nFaces          12544;
        startFace       38000;
    }
    GV1
    {
        type            wall;
        inGroups        1(wall);
        nFaces          66;
        startFace       50544;
    }
    GV2
    {
        type            wall;
        inGroups        1(wall);
        nFaces          66;
        startFace       50610;
    }
    GV3
    {
        type            wall;
        inGroups        1(wall);
        nFaces          66;
        startFace       50676;
    }
    GV4
    {
        type            wall;
        inGroups        1(wall);
        nFaces          66;
        startFace       50742;
    }
    GV5
    {
        type            wall;
        inGroups        1(wall);
        nFaces          66;
        startFace       50808;
    }
    GV6
    {
        type            wall;
        inGroups        1(wall);
        nFaces          66;
```

```
        startFace       50874;
    }
    GV7
    {
        type            wall;
        inGroups        1(wall);
        nFaces          66;
        startFace       50940;
    }
    GV8
    {
        type            wall;
        inGroups        1(wall);
        nFaces          66;
        startFace       51006;
    }
    GV9
    {
        type            wall;
        inGroups        1(wall);
        nFaces          66;
        startFace       51072;
    }
    GV10
    {
        type            wall;
        inGroups        1(wall);
        nFaces          66;
        startFace       51138;
    }
    GV11
    {
        type            wall;
        inGroups        1(wall);
        nFaces          66;
        startFace       51204;
    }
    GV12
    {
        type            wall;
        inGroups        1(wall);
        nFaces          66;
        startFace       51270;
    }
    GV13
    {
        type            wall;
        inGroups        1(wall);
        nFaces          66;
        startFace       51336;
    }
    GV14
    {
        type            wall;
        inGroups        1(wall);
        nFaces          66;
        startFace       51402;
    }
    GV15
    {
        type            wall;
        inGroups        1(wall);
        nFaces          66;
        startFace       51468;
    }
    GV16
    {
        type            wall;
        inGroups        1(wall);
        nFaces          66;
        startFace       51534;
    }
)
```

Appendix B: "0.org/U" file for the Guide vane case

Location of the file:

`TME205_asaraf/Guidevanerotation/Test/0.org`

The following lines of code are employed in this tutorial:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform (0 0 0);

boundaryField
{
    inlet
    {
         type            cylindricalInletVelocity;
         axis             (0 0 1);
         centre           (0 0 0);
         axialVelocity    0;
         radialVelocity   -3.92;
         rpm              120;
         value            uniform (0 0 0);
    }
    outlet
    {
        type             zeroGradient;
    }
    top
    {
        type            empty;
    }

    bottom
    {
        type            empty;

    }
    GV1
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }

    GV2
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
    GV3
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
    GV4
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
    GV5
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
    GV6
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
    GV7
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
GV8
    {
```

```
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
GV9
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
GV10
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
GV11
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
GV12
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
GV13
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
GV14
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
GV15
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
GV16
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }

}
```

Appendix C: "0.org/pointMotionU" file for the Guide vane case

Location of the file:

`TME205_asaraf/Guidevanerotation/Test/0.org`

The following lines of code are employed in this tutorial:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       pointVectorField;
    object      pointMotionU;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform (0 0 0);

boundaryField
{

    inlet
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    outlet
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    top
    {
        type            empty;
    }

    bottom
    {
        type            empty;
    }
    GV1
    {
        type                        librotationVelocity;
         axis                       (0 0 1);
         origin                     (0.0780350 .3923143 0);// Center of Rotation
         angle0                     -4;// Degrees per second
         value                      uniform (0 0 0);
    }
    GV2
    {
        type                        librotationVelocity;
         axis                       (0 0 1);
         origin                     (0.22223 0.33259  0);// Center of Rotation
         angle0                     3;// Degrees per second
         value                      uniform (0 0 0);
    }
    GV3
    {
        type                        librotationVelocity;
         axis                       (0 0 1);
         origin                     (0.33259 0.22223  0);// Center of Rotation
         angle0                     -3;// Degrees per second
         value                      uniform (0 0 0);
    }
    GV4
    {
        type                        librotationVelocity;
         axis                       (0 0 1);
         origin                     (0.39231 0.07804  0);// Center of Rotation
         angle0                     3;// Degrees per second
         value                      uniform (0 0 0);
    }
    GV5
    {
        type                        librotationVelocity;
         axis                       (0 0 1);
         origin                     (0.39231 -0.07804 0);// Center of Rotation
         angle0                     -3;// Degrees per second
         value                      uniform (0 0 0);
    }
    GV6
    {
```

```
        type                    librotationVelocity;
        axis                    (0 0 1);
        origin                  (0.33259 -0.22223 0);// Center of Rotation
        angle0                  3;// Degrees per second
        value                   uniform (0 0 0);
    }
    GV7
    {
        type                    librotationVelocity;
        axis                    (0 0 1);
        origin                  (0.22223 -0.33259 0);// Center of Rotation
        angle0                  -3;// Degrees per second
        value                   uniform (0 0 0);
    }
    GV8
    {
        type                    librotationVelocity;
        axis                    (0 0 1);
        origin                  (0.07804 -0.39231 0);// Center of Rotation
        angle0                  3;// Degrees per second
        value                   uniform (0 0 0);
    }
    GV9
    {
        type                    librotationVelocity;
        axis                    (0 0 1);
        origin                  (-0.07804        -0.39231 0);// Center of Rotation
        angle0                  -3;// Degrees per second
        value                   uniform (0 0 0);
    }
    GV10
    {
        type                    librotationVelocity;
        axis                    (0 0 1);
        origin                  (-0.22223        -0.33259 0);// Center of Rotation
        angle0                  3;// Degrees per second
        value                   uniform (0 0 0);
    }
    GV11
    {
        type                    librotationVelocity;
        axis                    (0 0 1);
        origin                  (-0.33259        -0.22223 0);// Center of Rotation
        angle0                  -3;// Degrees per second
        value                   uniform (0 0 0);
    }
    GV12
    {
        type                    librotationVelocity;
        axis                    (0 0 1);
        origin                  (-0.39231        -0.07804 0);// Center of Rotation
        angle0                  3;// Degrees per second
        value                   uniform (0 0 0);
    }
    GV13
    {
        type                    librotationVelocity;
        axis                    (0 0 1);
        origin                  (-0.39231        0.07804  0);// Center of Rotation
        angle0                  -3;// Degrees per second
        value                   uniform (0 0 0);
    }
    GV14
    {
        type                    librotationVelocity;
        axis                    (0 0 1);
        origin                  (-0.33259        0.22223  0       );// Center of Rotation
        angle0                  3;// Degrees per second
        value                   uniform (0 0 0);
    }
    GV15
    {
        type                    librotationVelocity;
        axis                    (0 0 1);
        origin                  (       -0.22223 0.33259  0       );// Center of Rotation
        angle0                  -3;// Degrees per second
        value                   uniform (0 0 0);
    }
    GV16
    {
        type                    librotationVelocity;
        axis                    (0 0 1);
        origin                  (-0.07804        0.39231  0);// Center of Rotation
        angle0                  3;// Degrees per second
        value                   uniform (0 0 0);
    }}
```

Appendix D: "0.org/p" file for the Guide vane case

Location of the file:

`TME205_asaraf/Guidevanerotation/Test/0.org`

The following lines of code are employed in this tutorial:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type            zeroGradient;

    }

    outlet
    {
        type            fixedValue;
         value uniform 0;
    }

    top
    {
        type            empty;

    }

    bottom
    {
        type            empty;

    }


    GV1
    {
        type            zeroGradient;

    }

    GV2
    {
        type            zeroGradient;

    }
    GV3
    {
        type            zeroGradient;

    }
    GV4
    {
        type            zeroGradient;

    }
    GV5
    {
        type            zeroGradient;

    }
    GV6
    {
        type            zeroGradient;

    }
    GV7
    {
        type            zeroGradient;

    }
    GV8
```

```
    {
        type            zeroGradient;

    }
    GV9
    {
        type            zeroGradient;

    }
    GV10
    {
        type            zeroGradient;

    }
    GV11
    {
        type            zeroGradient;

    }
    GV12
    {
        type            zeroGradient;

    }
    GV13
    {
        type            zeroGradient;

    }
    GV14
    {
        type            zeroGradient;

    }
    GV15
    {
        type            zeroGradient;

    }
    GV16
    {
        type            zeroGradient;

    }
}
```

Appendix E: “forces” file for the Guide vane case

Location of the file:

`TME205_asaraf/Guidevanerotation/Test/system`

The following lines of code are employed in this tutorial:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/

forces1
{
    type        forces;
    functionObjectLibs ( "libforces.so" );
    outputControl   timeStep;
    timeInterval    1;
    log         yes;
    patches     (GV1);
    pName       p;
    UName       U;
    rhoName     rhoInf;     // Indicates incompressible
    log         true;
    rhoInf      1;          // Redundant for incompressible
    CofR        (0.07804  0.39231  0       );    // Rotation around axis
}
forces2
{
    type        forces;
    functionObjectLibs ( "libforces.so" );
    outputControl   timeStep;
    timeInterval    1;
    log         yes;
    patches     (GV2);
    pName       p;
    UName       U;
    rhoName     rhoInf;     // Indicates incompressible
    log         true;
    rhoInf      1;          // Redundant for incompressible
    CofR        (0.22223  0.33259  0       );    // Rotation around axis
}
forces3
{   type        forces;
    functionObjectLibs ( "libforces.so" );
    outputControl   timeStep;
    timeInterval    1;
    log         yes;
    patches     (GV3);
    pName       p;
    UName       U;
    rhoName     rhoInf;     // Indicates incompressible
    log         true;
    rhoInf      1;          // Redundant for incompressible
    CofR        (0.33259  0.22223  0       );    // Rotation around axis
}
forces4
{
    type        forces;
    functionObjectLibs ( "libforces.so" );
    outputControl   timeStep;
    timeInterval    1;
    log         yes;
    patches     (GV4);
    pName       p;
    UName       U;
    rhoName     rhoInf;     // Indicates incompressible
    log         true;
    rhoInf      1;          // Redundant for incompressible
    CofR        (0.39231  0.07804  0       );    // Rotation around axis
}
forces5
{
    type        forces;
    functionObjectLibs ( "libforces.so" );
    outputControl   timeStep;
    timeInterval    1;
    log         yes;
    patches     (GV5);
    pName       p;
    UName       U;
```

```
    rhoName     rhoInf;     // Indicates incompressible
    log         true;
    rhoInf      1;          // Redundant for incompressible

    CofR        (0.39231  -0.07804 0       );    // Rotation around axis
}
forces6
{
    type        forces;
    functionObjectLibs ( "libforces.so" );
   outputControl   timeStep;
    timeInterval    1;
    log         yes;
    patches     (GV6);
    pName       p;
    UName       U;
    rhoName     rhoInf;     // Indicates incompressible
    log         true;
    rhoInf      1;          // Redundant for incompressible
    CofR        (0.33259  -0.22223 0       );    // Rotation around axis
}
forces7
{
    type        forces;
    functionObjectLibs ( "libforces.so" );
    outputControl   timeStep;
    timeInterval    1;
    log         yes;
    patches     (GV7);
    pName       p;
    UName       U;
    rhoName     rhoInf;     // Indicates incompressible
    log         true;
    rhoInf      1;          // Redundant for incompressible
    CofR        (0.22223  -0.33259 0       );    // Rotation around axis
}
forces8
{
    type        forces;
    functionObjectLibs ( "libforces.so" );
    outputControl   timeStep;
    timeInterval    1;
    log         yes;
    patches     (GV8);
    pName       p;
    UName       U;
    rhoName     rhoInf;     // Indicates incompressible
    log         true;
    rhoInf      1;          // Redundant for incompressible
    CofR        (0.07804  -0.39231 0       );    // Rotation around axis
}
forces9
{
    type        forces;
    functionObjectLibs ( "libforces.so" );
    outputControl   timeStep;
    timeInterval    1;
    log         yes;
    patches     (GV9);
    pName       p;
    UName       U;
    rhoName     rhoInf;     // Indicates incompressible
    log         true;
    rhoInf      1;          // Redundant for incompressible
    CofR        (-0.07804 -0.39231 0       );    // Rotation around axis
}
forces10
{
    type        forces;
    functionObjectLibs ( "libforces.so" );
    outputControl   timeStep;
    timeInterval    1;
    log         yes;
    patches     (GV10);
    pName       p;
    UName       U;
    rhoName     rhoInf;     // Indicates incompressible
    log         true;
    rhoInf      1;          // Redundant for incompressible
    CofR        (-0.22223 -0.33259 0       );    // Rotation around axis
}
forces11
{
    type        forces;
    functionObjectLibs ( "libforces.so" );
    outputControl   timeStep;
    timeInterval    1;
```

```
    log         yes;
    patches     (GV11);
    pName       p;
    UName       U;
    rhoName     rhoInf;     // Indicates incompressible
    log         true;
    rhoInf      1;          // Redundant for incompressible
    CofR        (-0.33259 -0.22223 0        );    // Rotation around axis
}
forces12
{
    type        forces;
    functionObjectLibs ( "libforces.so" );
    outputControl   timeStep;
    timeInterval    1;
    log         yes;
    patches     (GV12);
    pName       p;
    UName       U;
    rhoName     rhoInf;     // Indicates incompressible
    log         true;
    rhoInf      1;          // Redundant for incompressible
    CofR        (-0.39231 -0.07804 0        );    // Rotation around axis
}
forces13
{
    type        forces;
    functionObjectLibs ( "libforces.so" );
    outputControl   timeStep;
    timeInterval    1;
    log         yes;
    patches     (GV13);
    pName       p;
    UName       U;
    rhoName     rhoInf;     // Indicates incompressible
    log         true;
    rhoInf      1;          // Redundant for incompressible
    CofR        (-0.39231 0.07804  0        );    // Rotation around axis
}
forces14
{
    type        forces;
    functionObjectLibs ( "libforces.so" );
    outputControl   timeStep;
    timeInterval    1;
    log         yes;
    patches     (GV14);
    pName       p;
    UName       U;
    rhoName     rhoInf;     // Indicates incompressible
    log         true;
    rhoInf      1;          // Redundant for incompressible
    CofR        (-0.33259 0.22223  0        );    // Rotation around axis
}
forces15
{
    type        forces;
    functionObjectLibs ( "libforces.so" );
    outputControl   timeStep;
    timeInterval    1;
    log         yes;
    patches     (GV15);
    pName       p;
    UName       U;
    rhoName     rhoInf;     // Indicates incompressible
    log         true;
    rhoInf      1;          // Redundant for incompressible
    CofR        (-0.22223 0.33259  0        );    // Rotation around axis
}
forces16
{   type        forces;
    functionObjectLibs ( "libforces.so" );
    outputControl   timeStep;
    timeInterval    1;
    log         yes;
    patches     (GV16);
    pName       p;
    UName       U;
    rhoName     rhoInf;     // Indicates incompressible
    log         true;
    rhoInf      1;          // Redundant for incompressible

    CofR        (-0.07804 0.39231  0        );    // Rotation around axis
}
```

Appendix F: "fvSolution" file for the Guide vane case

Location of the file:

`TME205_asaraf/Guidevanerotation/Test/system`

The following lines of code are employed in this tutorial:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    pcorr
    {
        solver           GAMG;
        tolerance        0.02;
        relTol           0;
        smoother         GaussSeidel;
        nPreSweeps       0;
        nPostSweeps      2;
        cacheAgglomeration true;
        agglomerator     faceAreaPair;
        nCellsInCoarsestLevel 10;
        mergeLevels      1;
    }

    p
    {
        $pcorr
        tolerance        1e-7;
        relTol           0.01;
    }

    pFinal
    {
        $p;
        tolerance        1e-7;
        relTol           0;
    }

    "(U|k|omega)"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-06;
        relTol          0.1;
    }

    "(U|k|omega)Final"
    {
        $U;
        tolerance       1e-06;
        relTol          0;
    }
    cellMotionU
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-08;
        relTol          0;
    }
    cellMotionUx
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-08;
        relTol          0;
    }
}

PIMPLE
{
    correctPhi          yes;
    nOuterCorrectors    2;
    nCorrectors         1;
    nNonOrthogonalCorrectors 0;

relaxationFactors
```

```
{
    fields
    {
        p               0.3;
    }
    equations
    {
        "(U|k|omega)"   0.7;
        "(U|k|omega)Final" 1.0;
    }
}

cache
{
    grad(U);
}
```

Appendix G: “new.py” python script for the Guide vane case

Location of the file:

`TME205_asaraf/Guidevanerotation/Test/`

The following lines of code are employed in this tutorial:

```
#!/usr/bin/python

import os
import sys
import math

forces_file = "postProcessing/forces16/0/forces.dat"

if not os.path.isfile(forces_file):
        print "Forces file not found at "+forces_file
        print "Be sure that the case has been run and you have the right directory!"
        print "Exiting."
        sys.exit()

def line2dict(line):
        tokens_unprocessed = line.split()
        tokens = [x.replace(")","").replace("(","") for x in tokens_unprocessed]
        floats = [float(x) for x in tokens]
        data_dict = {}
        data_dict['time'] = floats[0]
        force_dict = {}
        force_dict['pressure'] = floats[1:4]
        force_dict['viscous'] = floats[4:7]
        force_dict['porous'] = floats[7:10]
        moment_dict = {}
        moment_dict['pressure'] = floats[10:13]
        moment_dict['viscous'] = floats[13:16]
        moment_dict['porous'] = floats[16:19]
        data_dict['force'] = force_dict
        data_dict['moment'] = moment_dict
        return data_dict

time = []
drag = []
lift = []
moment = []
with open(forces_file,"r") as datafile:
        for line in datafile:
                if line[0] == "#":
                        continue
                data_dict = line2dict(line)
                time += [data_dict['time']]
                drag += [data_dict['force']['pressure'][0] + data_dict['force']['viscous'][0]]
                lift += [data_dict['force']['pressure'][1] + data_dict['force']['viscous'][1]]
                moment += [data_dict['moment']['pressure'][2] + data_dict['moment']['viscous'][2]]
datafile.close()

outputfile = open('forces16.txt','w')
for i in range(0,len(time)):
        outputfile.write(str(time[i])+' '+str(lift[i])+' '+str(drag[i])+' '+str(moment[i])+'\n')
outputfile.close()
```

Appendix H: "blockMeshDict" file script for the "mydynamicFvMesh" class tutorial

Location of the file:

`TME205_asaraf/meshrefinetest/Test/constant/polyMesh`

The following lines of code are employed in this tutorial:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 2)
    (1 0 2)
    (1 1 2)
    (0 1 2)
);

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

edges
(
);

boundary
(
    fixed
    {
        type patch;
        faces
        (
            (2 6 5 1)
            (1 5 4 0)
            (3 7 6 2)
            (0 4 7 3)
        );
    }
    top
    {
        type patch;
        faces
        (
            (4 5 6 7)
        );
    }
    bottom
    {
        type patch;
        faces
        (
            (0 3 2 1)
        );
    }
);

mergePatchPairs
(
);

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