Mesh motion alternatives

CFD with OpenFOAM

Andreu Oliver González

December 2009, Göteborg (Sweden)

INDEX

1 INTRODUCTION

The aim of this project is to study different alternatives of mesh motion. There is presented an overview of the different classes that can be used in order to define a mesh with motion with the purpose to give some information to be able to select the appropriate class for each situation. After this overview of the different classes available for mesh manipulation, a deep description is carried out for the *dynamicInkJetFvMesh* and a modification of this class will be done.

The solver used for problems with moving mesh is *IcoDyMFoam* solver; this solver is a solver for incompressible and non turbulent flow. In the case of compressible and turbulent flow the *turbDyMFoam* can be used.

The *icoDyMFoam* application is a transient solver for incompressible, laminar flow of Newtonian fluids on a moving mesh; that solver is used in version 1.5.x of the OpenFOAM, as well as the *turbDyMFoam* one. In 1.6.x version, they have both been collected in the *pimpleDyMFoam*.

2 MESH MOTION APPROACHES AND THE DIFFERENT CLASSES

There are two mesh manipulation approaches; the difference between them is the topology changing during the simulation or not. These two types are named *dynamicFvMesh* and *topoChangerFvMesh*. For each approach, there are different classes and they are the ones that follow:

- *dynamicFvMesh*: automatic mesh motion, for the case where the mesh topology does not change. There are five different classes for this method:

  1) *staticFvMesh*, where the mesh has no motion.

  2) *dynamicMotionSolverFvMesh*, it is used in cases where the resolution is not changing too much during the mesh motion, for relatively small changes.

  3) *dynamicInkJetFvMesh*, similar to the one before, but in this case the mesh movement is based on harmonic motion around a reference plane along x axis (the subdictionary *dynamicInkJetFvMeshCoeffs* is used).

  4) *dynamicRefineFvMesh*, it is similar to the *staticFvMesh* class but in this case a refinement or unrefinement of the mesh in the three directions is carried out by adding points.

5) *solidBodyMotionFvMesh*, similar to *dynamicMotionSolverFvMesh* used to describe solid body motion of the mesh specified by a run-time selectable motion function.

- *topoChangerFvMesh*: topological mesh changes, when the mesh topology changes during the simulation. There are four types of classes for this approach:

  1) *linearValveFvMesh*, to use sliding meshes at the interface of 2 pieces of mesh in relative linear motion

  2) *linearValveLayersFvMesh*, used as the class before but layer addition and removal is the extra feature instead of pure squeezing or stretching of the nodes and cells.

  3) *mixerFvMesh*, used when sliding interface needed between one moving part and a fixed one.

  4) *movingConeTopoFvMesh*, first is preformed simply by squeezing and stretching, but when cell layer thickness reach a critical value a new cell layer is added or an old cell layer is removed.

3 PROCEDURE FOR DEFINING A MESH WITH MOTION

The structure of files in order to solve this kind of applications is the usual one, where you find folder with the initial values (0) and two folders, which are:

- *constant,* where it can be found some files and folders such as *dynamicMeshDict* file, *transportProperties* file, *polyMesh* folder.
- *system*, where it is found *controlDict* file, *fvSchemes* file and *fvSolution* file, but also other files are found depending on the approach followed for the mesh motion.

First of all, the mesh has to be defined in the *blockMeshDict* file inside the *constant* folder. In order to have mesh motion in any direction, for some classes the boundary type should be set to *patch* for the moving and changing cells in the direction where motion can be defined. Then, moving-mesh boundary conditions have to be specified to allow the movement in the desired direction.

A part from that, *dynamicMeshDict* file has to be added inside the *constant* folder, where the different definitions used and needed for the moving mesh are specified (mesh manipulation dictionaries, solvers, classes, diffusivities and coefficients required for the case).

3.1 Mesh motion method

3.1.1 Automatic mesh motion (*dynamicFvMesh*)

The mesh motion is obtained by solving a mesh motion equation, where boundary motion acts as a boundary condition and determines the position of mesh points. The motion is characterized by the spacing between nodes, which changes by stretching and squeezing. This mesh motion equation can be simplified, and there are mainly four types:

- Spring analogy, which is insufficiently robust.
- Linear plus torsional spring analogy, which is complex, expensive and non-linear.
- Laplace equation with constant and variable diffusivity.
- Linear pseudo-solid equation for small deformations.

The mesh spacing and quality is controlled by variable diffusivity (*3.3 Diffusivity model*). Changing the diffusivity implies redistribution of the boundary motion through the volume of the mesh. And referring to the mesh quality, in order to preserve it, definition of valid motion from an initially valid mesh implies that no forces or cells are inverted during motion.

The corresponding library for this mesh manipulation approach is the *libDynamicFvMesh.so*.

The most important types of classes for the automatic mesh motion, where the topology does not change during simulation, are:

1) *staticFvMesh*, where the mesh has no motion.

2) *dynamicMotionSolverFvMesh*, it solves the cell movement equations and it is the simplest type of mesh motion solver. There should be specified the solver and the type of diffusivity model.

3) *dynamicInkJetFvMesh*, the subdictionary *dynamicInkJetFvMeshCoeffs* is used. In that class, an equation defines the motion and neither the solver nor the diffusivity model are needed.

3.1.2 Topological changes in the mesh (*topoChangerFvMesh*)

The number of points, faces, cells and/or mesh connectivity changes during simulation. It is used for more demanding and complex mesh motion than the automatic approach where the original topology cannot be kept or the precision of the solution would be affected by keeping the original mesh settings during the simulations. For that, mesh modifiers are required to describe what kind of mesh manipulation action is carried out:

- Attach or detach of boundary.

- Layer addition or removal.
- Sliding interface.

The class *polyTopoChanger* will look for the necessary data and extract it from the extra dictionary *meshModifiers*, otherwise, the data will be read from the *dynamicMeshDict.* The corresponding dynamic library is *libtopoChangerFvMesh.so*.

There are four types of classes for this approach, as it was presented before and now more things are said about them:

1) *linearValveFvMesh*, the dictionary *linearValveFvMeshCoeffs* to select motion solver type for mesh handling is used.

2) *linearValveLayersFvMesh*, the input variables needed are the same as the ones for the class presented before and, moreover, the ones found in the extra subdictionary *layer*.

3) *mixerFvMesh*, a part from the *dynamicMeshDict*, the dictionary *MRFZones* is important because it is where the moving parts are determined. From this dictionary, different zones are generated and those are used by the *slidingInterface* class which gives the relative motion between the two sides of the sliding interface.

4) *movingConeTopoFvMesh*, a part from the *dynamicMeshDict* file and the extra dictionary *meshModifiers* in the *movingConeTopoFvMesh.C* is required a sub-dictionary to specify the coefficients to define the moving and fixed boundaries and characteristics, besides the minimum and maximum cell layer thickness in each region.

3.2 Appropiate solver

Once the mesh is set, the moving points of the grid require models and corresponding mesh motion equations to be solved. The most used ones are:

- *displacementLaplacian*, the equations of cell motion are solved based on the Laplacian of the diffusivity and the cell displacement (*pointDisplacement* extra file is required in the starting time folder). For this solver, the final displacement of the mesh components is needed as well as the mesh displacement of the internal field.
- *velocityLaplacian*, similar to the previous solver with the difference being the equation solved, which is the Laplacian of the diffusivity and the cell motion velocity (*pointMotionU* file has to be available to be read). A part from the input variables that are the same as those in the *displacementLaplacian* case, the user has to be aware that the code deals with the boundary velocities instead of the final motions, so care have to be taken when determining the dimensions. It is used when each

time an order of magnitude of the maximum displacement is known to be not too big.

- *LaplaceFaceDecomposition*, used when the order of magnitude of the maximum displacement is not known or known to be big. The mesh is rebuild after a decomposition of all cells and faces and the Laplace smoothing equation is solved by the Finite Element Method. It increases the robustness but, in the other hand, it increases the computational cost compared to the *velocityLaplacian* solver.
- *SBRStress*, it is a displacement model, solving Laplacian of diffusivity and the cellDisplacement and it considers also the solid body rotation term in calculations (*pointDisplacement* file is also required in the *constant* directory).

It has to be added that not always a solver is required because sometimes the motion is described by an equation inside the class definition and it is solved internally.

3.3 Diffusivity model

The diffusivity model is used to determine how the points should be moved after solving the cell motion equation for each time step. There are two groups of diffusivity models:

1) Quality based methods, where diffusion field is function of cell quantity measure. There four types and they are the following ones:
   a. uniform, the mesh manipulation is done uniformly for all moving boundaries by stretching or squeezing with the same ratio all the cells in each region.
   b. directional, the mesh stretching or squeezing is done proportionally to the direction of the motion. The main idea in this case is that the mesh manipulation is done by considering the slipping boundaries. Two scalar coefficients are required, one defining the mean cell non orthogonality and the other one to determine the mean cell skewness.
   c. motionDirectional, where the mesh manipulation is done by prioritizing the moving body and adjusting the cells in a way that is more appropriate for the moving body. The same coefficients than for the above method have to be specified.
   d. inverseDistance, where the user specifies one or more boundaries and the diffusivity of the field is based on the inverse of the distance from that boundary.
2) Distance based methods, used together with the quality based methods and in which the diffusion field will be a function of the inverse of cell centre distance 'l' to the nearest selected boundary. There are three of them, which are:

a. linear, the diffusivity field is based linearly on the inverse of the cell center distance to the nearest boundary.
b. quadratic, as the one above except being a quadratic relation instead of a linear one.
c. exponential, in this case the diffusivity of the field is based on the exponential of the inverse of cell-center distance to the selected boundaries.

As said for the solvers, for the classes where the motion is solved internally in the mesh class, the diffusivity model is not needed.

4 *dynamicInkJetFvMesh*

At this point, as for my master thesis I will have to model the vertebral column to make some CFD simulations with the purpose to evaluate and study the whiplash pain causes, the *dynamicInkJetFvMesh* is studied deeply. This class is the appropriate one because the motion of the model will be given, meaning that the motion is known for a different number of time steps by analyzing some experiments with FEM; due to that no solver is needed. Even though dynamicInkJetFvMesh is the suitable class, some changes will have to be done on it. In this project only a simple modification will be done in order to get a better understanding of the class and how to make motion modifications in that particular class.

4.1 Explanation of *dynamicInkJetFvMesh* class

The code of *dynamicInkJetFvMesh.C* is provided just below with some comments to understand how it works:

```
#include "dynamicInkJetFvMesh.H"
#include "addToRunTimeSelectionTable.H"
#include "volFields.H"
#include "mathematicalConstants.H"
// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //
namespace Foam
{
    defineTypeNameAndDebug(dynamicInkJetFvMesh, 0);
```
//It call the functions typeName and debug to specify the type class used, which is dynamicInkJetFvMesh in this case, and some information for debugging.
```
    addToRunTimeSelectionTable(dynamicFvMesh, dynamicInkJetFvMesh, IOobject);
```
//It adds the dynamicInkJetFvMesh (which is *thisType*, dynamicInkJetFvMesh, inside the *baseType*, dynamicFvMesh) to the table where the classes used are defined
```
}
// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //
Foam::dynamicInkJetFvMesh::dynamicInkJetFvMesh(const IOobject& io)
:
    dynamicFvMesh(io),
    dynamicMeshCoeffs_
    (
        IOdictionary
        (
            IOobject
            (
                "dynamicMeshDict",
                io.time().constant(),            //dynamicMeshDict is located in the folder constant
                *this,
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            )
        ).subDict(typeName + "Coeffs")           //A subdictionary called dynamicInkJetFvMeshCoeffs exist inside the dynamicFvMesh with the following scalar numbers
    ),
    amplitude_(readScalar(dynamicMeshCoeffs_.lookup("amplitude"))),
    frequency_(readScalar(dynamicMeshCoeffs_.lookup("frequency"))),
    refPlaneX_(readScalar(dynamicMeshCoeffs_.lookup("refPlaneX"))),
    stationaryPoints_
    (
        IOobject
        (
            "points",
            io.time().constant(),                //the file points is also located in the folder constant
            meshSubDir,
            *this,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    )
{
    Info<< "Performing a dynamic mesh calculation: " << endl
        << "amplitude: " << amplitude_
        << " frequency: " << frequency_
        << " refPlaneX: " << refPlaneX_ << endl;
}
// * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * //
Foam::dynamicInkJetFvMesh::~dynamicInkJetFvMesh()
{}
// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //
bool Foam::dynamicInkJetFvMesh::update()         //member function for this class where the motion equation is defined and it updates the mesh
{
    scalar scalingFunction =
        0.5*(::cos(2*mathematicalConstant::pi*frequency_*time().value()) - 1.0);
    Info<< "Mesh scaling. Time = " << time().value() << " scaling: "
        << scalingFunction << endl;

    pointField newPoints = stationaryPoints_;    //new points are given the values of the stationary ones
    newPoints.replace
    (
        vector::X,
        stationaryPoints_.component(vector::X)*
        (
            1.0
          + pos
```

```
            (
              - (stationaryPoints_.component(vector::X))
              - refPlaneX_
            )*amplitude_*scalingFunction
        )
    );      //with the function replace the new points are recalculated following the motion
equation described just above. With vector::X specification it is said that the motion is only
changing the mesh in one direction, in this case in the X direction

    fvMesh::movePoints(newPoints);         //Mesh points are moved to the new points
calculated
    volVectorField& U =
        const_cast<volVectorField&>(lookupObject<volVectorField>("U"));
    U.correctBoundaryConditions();
    return true;
}
```

The code of *dynamicInkJetFvMesh.H* is provided in the Appendix.

In order to get a better idea of how it works an example is developed to show it.

4.2 Example of use of the *dynamicInkJetFvMesh* class

The example is going to show the motion of a very simple mesh by using the icoDyMFoam solver, but only using from it the part that solves the mesh manipulation.

To start to make the example, create the example folder:

```
>> mkdir $WM_PROJECT_USER_DIR/myExample
```

This folder has to have the following structure with the following three folders:

- 0
  - p
  - U
- constant
  - *dynamicMeshDict*.
  - polyMesh, where the *blockMeshDict* is located.
  - *transportProperties*.
- system
  - *controlDict*.
  - *fvSchemes*.
  - *fvSolution*.

First of all a simple mesh is defined in the *blockMeshDict* (code attached in the Appendix, like also the codes for *p*, *U*, *transportProperties*, *controlDict, fvSchemes and fvSolution*); the geometry chosen is a long and thin rectangle (0.006x0.075x0.001m) which is fixed from the bottom part, shown in Figure 1. The mesh is defined in the negative side of the x axis, which means that it goes from -0.006 until 0.

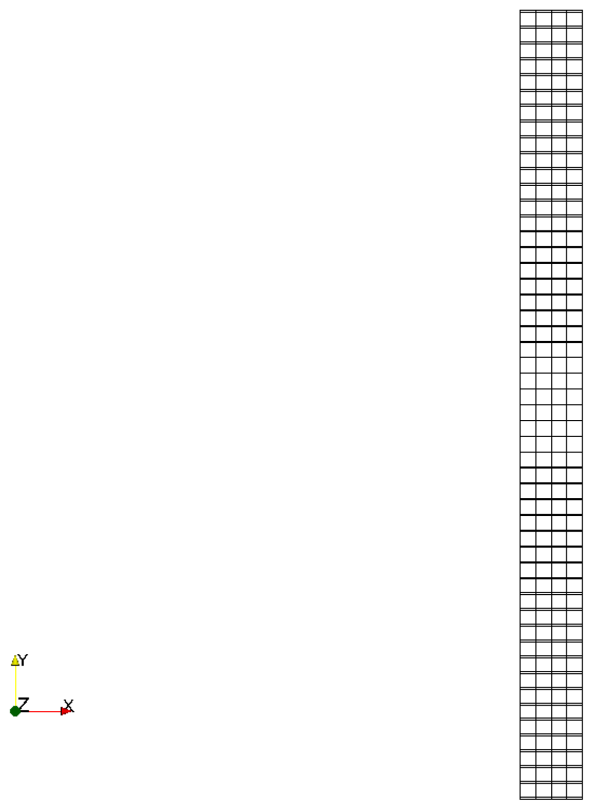

Figure 1: Mesh geometry.

Once the mesh is defined, the utility to generate the mesh is called:

```
>> blockMesh
```

Then, the file *dynamicMeshDict* has to be created inside the *constant* folder where the class for the mesh manipulation is specified and where the subdictionary *dynamicInkJetFvMEshCoeffs* has to be described with the values of the coefficients required for the class (amplitude, frequency and plane of reference). The code for *dynamicMeshDict* is shown below:

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

dynamicFvMesh dynamicInkJetFvMesh;
motionSolverLibs ("libfvMotionSolvers.so");
dynamicInkJetFvMeshCoeffs
{
        amplitude       0.06;
        frequency       2;
        refPlaneX       0;
}

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

The motion defined with this class makes the set of points to be compressed and expanded sinusoidally to impose a sinusoidal variation (Eq. 1 & 2):

$$\text{scaling\_function} = 0.5\cdot(\cos(2\pi tf) - 1) \qquad \text{Eq. 1}$$

$$x = x_{old}\cdot(1 + \text{pos}(-x_{old} - \text{refPlaneX})\cdot\text{amplitude}\cdot\text{scaling\_function}) \qquad \text{Eq. 2}$$

The coefficients, as seen in the code from *dynamicMeshDict*, are set to:

Amplitude = 0.06 Frequency = 2 Reference_Plane_X = 0

At this point, the mesh can be moved. As it is going to be used the *icoDyMFoam* solver to do that, the parts from the solver that are not used to manipulate the mesh are going to be deleted and therefore the solver is renamed as *icoDyMFoamMesh*; the code for *icoDyMFoamMesh.C* is shown in the Appendix. For that, the following steps should be followed:

```
>> cp –r $FOAM_SOLVERS/incompressible/icoDyMFoam \
$WM_PROJECT_USER_DIR/icoDyMFoamMesh
>> cd icoDyMFoamMesh
>> wclean
>> mv icoDyMFoam.C icoDyMFoamMesh.C
```

The parts from *icoDyMFoamMesh.C* that can be deleted are:

- Make the fluxes absolute.
- Make the fluxes relative to the mesh motion.
- Pimple loop.

After those modifications, the solver should be recompiled and to be able to do it the file *Make/files* should be as follows:

```
icoDyMFoamMesh.C
EXE = $(FOAM_USER_APPBIN)/icoDyMFoamMesh
```

Now, the solver can be compiled:

```
>> wmake
```

Finally by calling the solver, the mesh manipulation can be seen in the paraFoam:

```
>> icoDyMFoamMesh
>> paraFoam
```

The example is used with different values for the three constants and the results achieved are shown in the figures below:

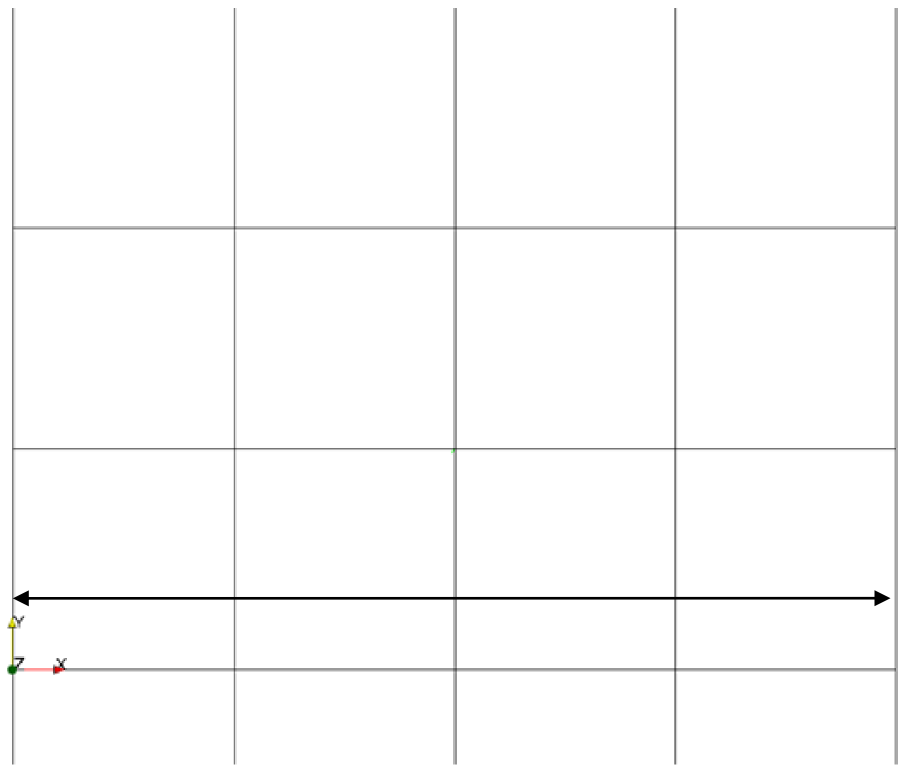

Figure 2: Initial position where motion is still not applied (t = 0s).

Figure 2 is showing the mesh in its initial position for all the cases that have been run, at t = 0s. And the arrow shown appears in all the figures to have the initial width of the mesh, which is 0.006m, and therefore have a reference to appreciate the mesh motion. In Figure 3, it is shown the mesh motion for the case using the constants defined before and it will be the reference to analyze how the three constants influence the mesh motion.

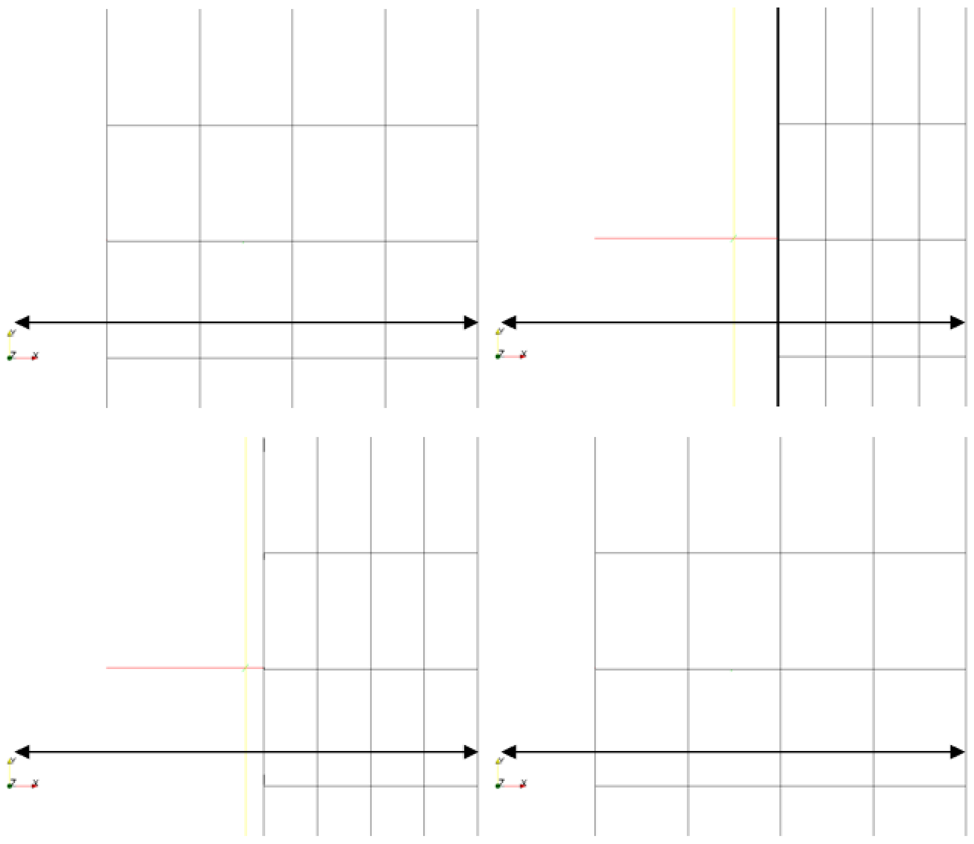
Figure 3: Mesh motion with 0.06m of amplitude, 2Hz of frequency and 0m of refPlaneX for t = 0.1, 0.25, 0.3 and 0.4s.

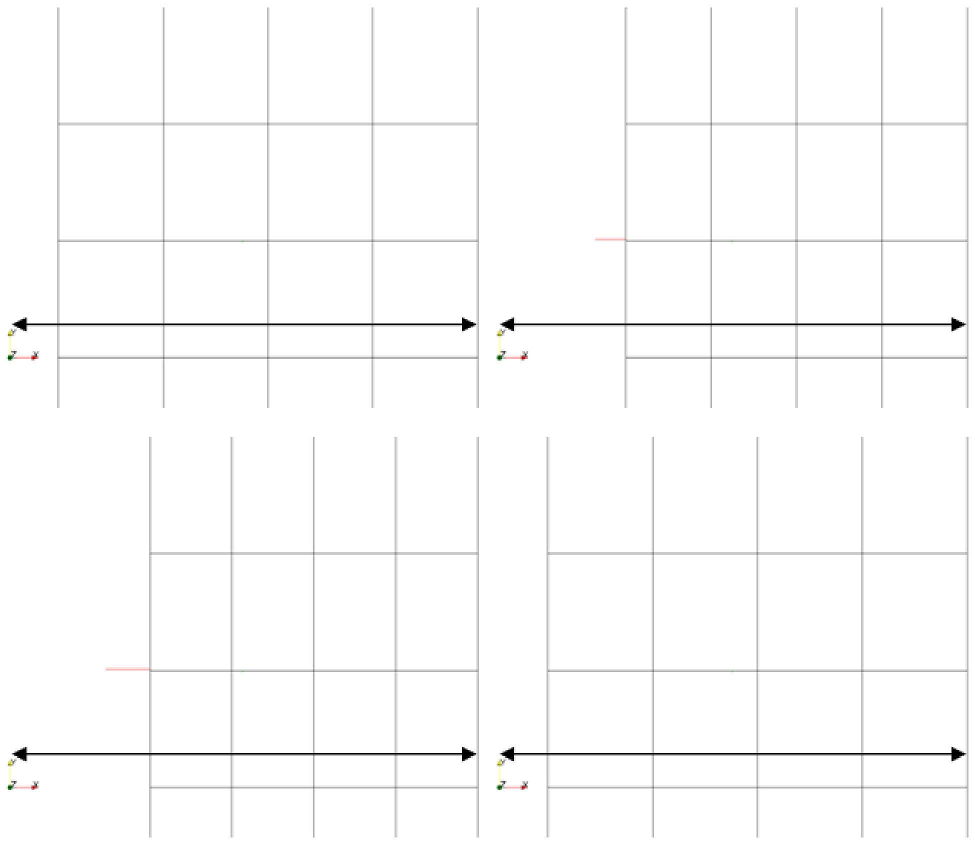
Figure 4: Mesh motion with 0.03m of amplitude, 2Hz of frequency and 0m of refPlaneX for t = 0.1, 0.2, 0.25 and 0.4s.

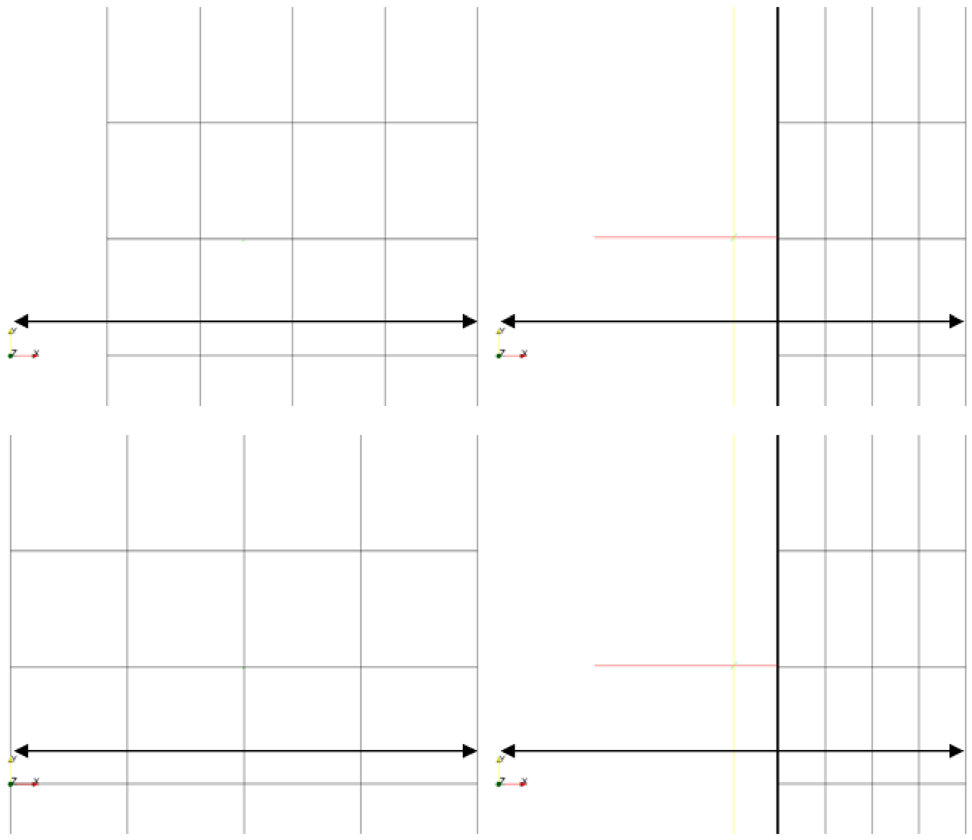

Figure 5: Mesh motion with 0.06m of amplitude, 4Hz of frequency and 0m of refPlaneX for t = 0.05, 0.125, 0.25 and 0.375s.

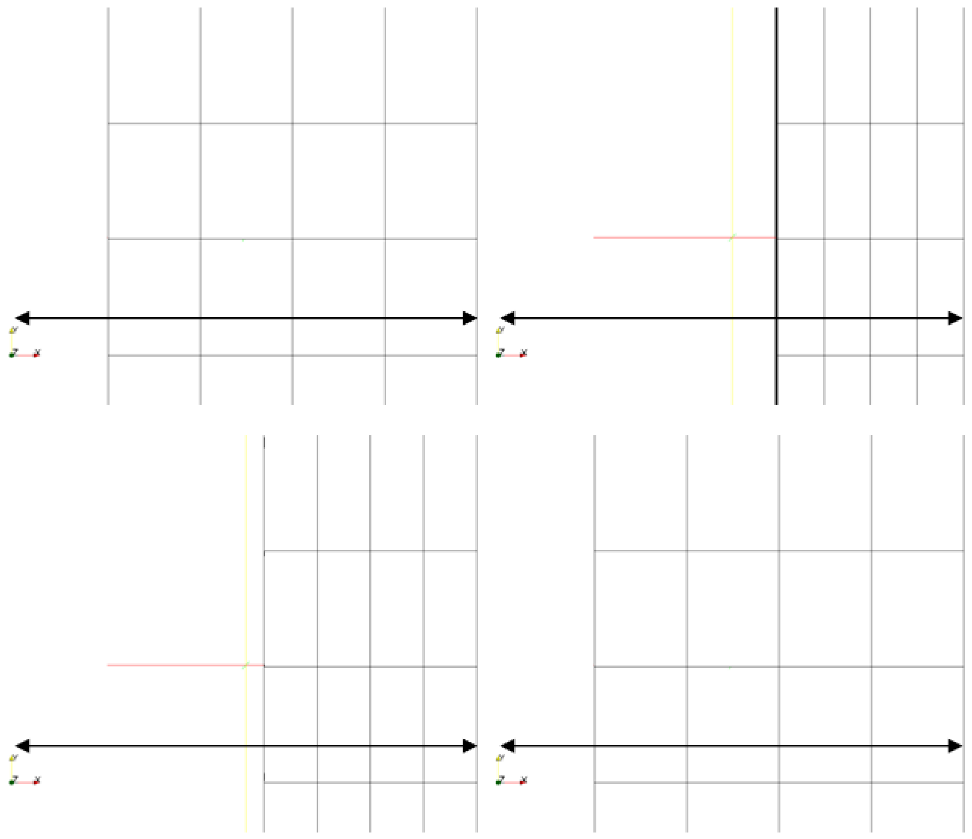

Figure 6: Mesh motion with 0.06m of amplitude, 2Hz of frequency and -0.003m of refPlaneX for t = 0.1, 0.25, 0.3 and 0.4s.

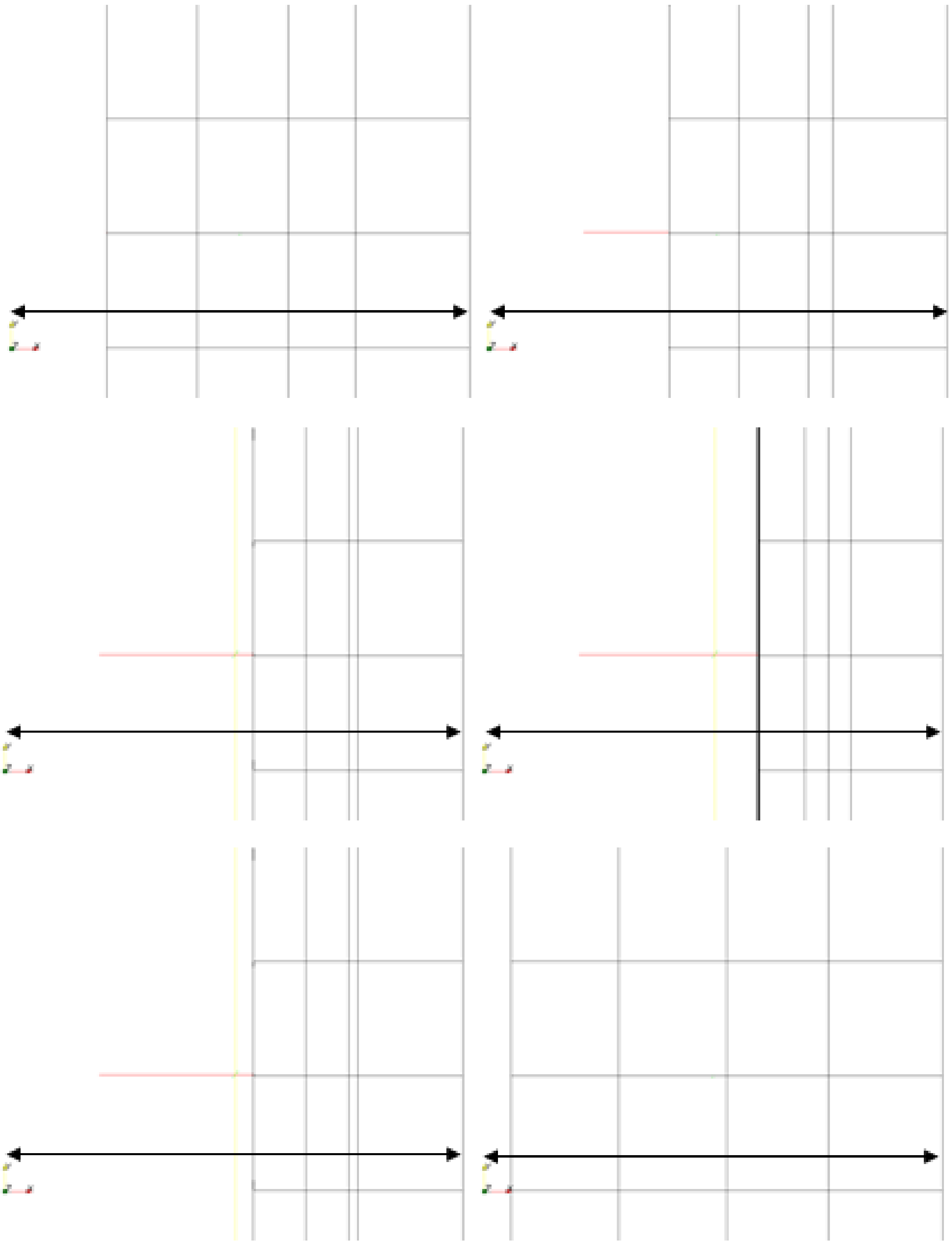

Figure 7: Mesh motion with 0.06m of amplitude, 2Hz of frequency and 0.003m of refPlaneX for t = 0.1, 0.15, 0.2, 0.25, 0.3 and 0.4s.

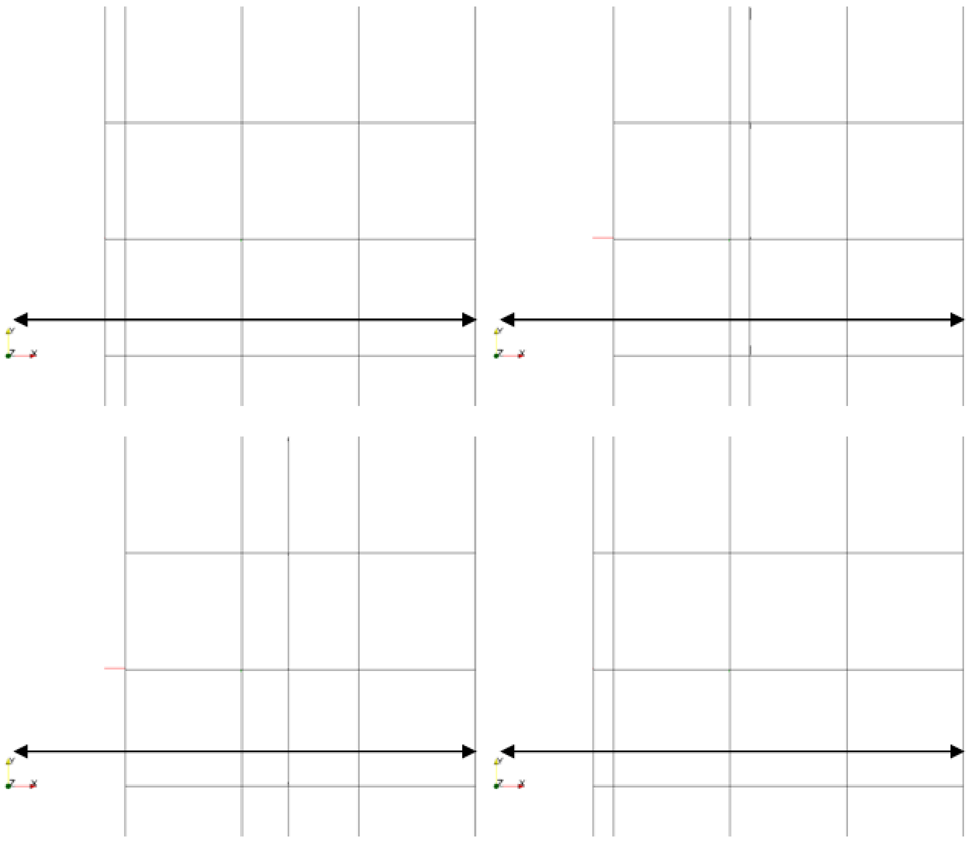

Figure 8: Mesh motion with 0.06m of amplitude, 2Hz of frequency and 0.006m of refPlaneX for t = 0.1, 0.2, 0.25 and 0.4s.

Observing the figures above and analyzing the equation of motion (Eq. 2), it can be seen that the three constants modify the motion in the following way:

- *amplitude*: varies the length the mesh is deformed in x direction. Looking at Figure 3 and 4, it can easily be seen how for the same time steps the position of the left side of the mesh has moved less; at t = 0.25s, when the maximum displacement takes place is the double in the Figure 3 because the *amplitude* value is the double.

- *frequency*: modifies the number of periods for the same total time and therefore varies the speed of change. Comparing Figure 3 and 5, it can be seen that with the same time (0.5s) the sinusoidal motion in Figure 5 has been done twice and that is because the *frequency* is the double that configuration. In order to see it easily, the equation (Eq. 1) has been analyzed; as the scaling_function is a cosinus function which is multiplied by 0.5 after 1 is substracted to it, therefore it goes from -1 to 0 which makes the motion to follow the sinus shape, as shown in the following figure:

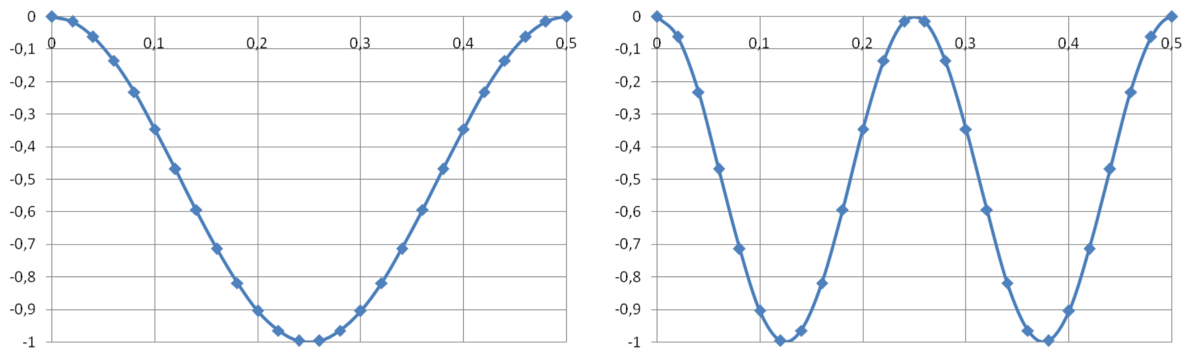

Figure 9: Scaling function for 2Hz and 4Hz of frequency from 0 to 0.5s.

- *refPlaneX*: from the equation (Eq. 2), it can be noted that the sinusoidal motion is with respect to *refPlaneX*. But it affects in different ways depending on the intervals where it is located, all considering that the mesh defined is going in x direction from -0.006 until 0:

- For *refPlaneX* $\in [-\infty,0]$, the values given by *pos* function are 1 for all the points of the mesh; therefore, the mesh motion will be the same for this interval of *refPlaneX* values.

- For *refPlaneX* $\in (0,0.006]$, the values given by *pos* function are 1 only for the points that has an x position smaller than –*refPlaneX*; therefore, only these points are moved, while the rest are kept in the initial position.

- For *refPlaneX* $\in (0.006,+\infty]$, the values given by *pos* function are 0 for all the points of the mesh; so there is no motion.

4.3 Modification of the class d*ynamicInkJetFvMesh* to *dynamicMyClassFvMesh*

In this part of the project it is going to be shown how to modify the class *dynamicInkJetFvMesh* with the purpose to define the desired motion.

The new class it is called *dynamicMyClassFvMesh* where a polynomial motion is going to be carried out; shown in Figure 5.

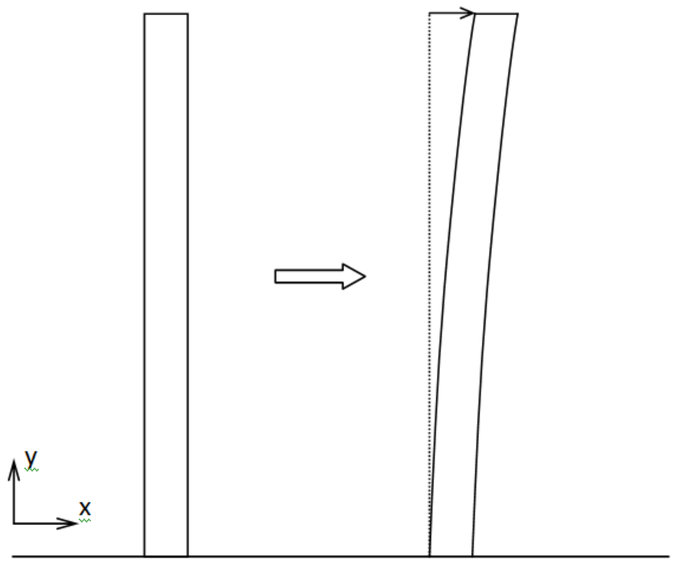

Figure 10: Motion schema for the new class.

The equation to define this motion is the one that follows:

$$x = a \cdot t \cdot y^2 + b \qquad \text{Eq. 3}$$

where x is the displacement in x direction and y is the position form the bottom part of the geometry. It is dependent on time to see the movement step by step, where time will be going from 0 to 1s. As the bottom part is fixed:

for $y = 0 \rightarrow x = 0 \rightarrow b = 0$

Defining a displacement in the top part, for example 10cm, when t = 1s:

for $y = 0.75 \rightarrow x = 0.1 \rightarrow a = 0.1778$

Then the coordinate x is updated with the next function:

$$x = x_{old} + a \cdot t \cdot y^2 \qquad \text{Eq. 4}$$

A scaling function has been added (Eq. 5) to provide a more complex motion giving then a displacement in x direction but in both sides. Therefore the updating function for the x coordinate is shown below:

$$\text{scaling\_function} = \cos(2\pi tf) \qquad \text{Eq. 5}$$

$$x = x_{old} + a \cdot t \cdot y^2 \cdot \text{scaling\_function} \qquad \text{Eq. 6}$$

In order to change the class, first the new class have to be created from the existing one:

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

At this point the new class has been created but only by changing the names from the original *dynamicInkJetFvMesh* class, therefore, it has to be compiled. To compile the *dynamicMyClassFvMesh* class, the *files* file and the *options* file have to be modified:

- *files*, rewritten as follows to only compile the *dynamicMyClassFvMesh* library:

```
dynamicMyClassFvMesh.C
LIB=$(FOAM_USER_LIBBIN)/libdynamicMyClassFvMesh
```

- *options*, the next line has been added to include the files from the original library:

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

Now the compilation can be done:

```
>> cd $WM_PROJECT_USER_DIR/dynamicMyClassFvMesh
>> wmake libso
```

When the compilation is done properly, then the modification can be done. This step defined just above, it is only done to ensure that the modification of name is done properly. To modify the motion equation, *dynamicMyClassFvMesh.C* has to be modified in the part where the equation is defined:

```
    a_(readScalar(dynamicMeshCoeffs_.lookup("a"))),
    frequency_(readScalar(dynamicMeshCoeffs_.lookup("frequency"))),
    // refPlaneX_(readScalar(dynamicMeshCoeffs_.lookup("refPlaneX"))),

        << "a: " << a_
        << " frequency: " << frequency_ << endl;
     //   << " refPlaneX: " << refPlaneX_ << endl;

    scalar scalingFunction =
        (::cos(2*mathematicalConstant::pi*frequency_*time().value());

    newPoints.replace
    (
        vector::X,
        stationaryPoints_.component(vector::X)+
        a_*time().value()*(stationaryPoints_.component(vector::Y))*
(stationaryPoints_.component(vector::Y))*scalingFunction
    );
```

The class has to be recompiled by typing:

```
>> wmake libso
```

4.4 Example of use of the *dynamicMyClassFvMesh* class

The example is the same that was done in part 4.2, but now it is going to show the motion of the mesh by using the new class.

The example defined in part 4.2 can be copied:

```
>> cp -r $WM_PROJECT_USER_DIR/myExample $WM_PROJECT_USER_DIR/myClassExample
```

But some changes have to be done in *controlDict* (where now the endTime is 1s, code attached in the Appendix) and *dynamicMeshDict*:

- reference to the new class library
- the needed coefficients (*a* and *frequency*) have to be redefined in the subdictionary *dynamicMyClassFvMeshCoeffs* inside the *dynamicMeshDict*. The values for the coefficients have been taken:

$$a = 0.4 \qquad frequency = 2$$

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

dynamicFvMeshLibs        ("libdynamicMyClassFvMesh.so");
dynamicFvMesh            dynamicMyClassFvMesh;
motionSolverLibs         ("libfvMotionSolvers.so");
dynamicMyClassFvMeshCoeffs
{
        a       0.4;
        frequency       2;
}

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

Now, the mesh can be moved and icoDyMFoamMesh is going to be used. By calling the solver, the mesh manipulation can be seen in the paraFoam:

```
>> icoDyMFoamMesh
>> paraFoam
```

As for the example shown for the original class, different values for the two constants are defined and the results achieved are shown in the figures below:

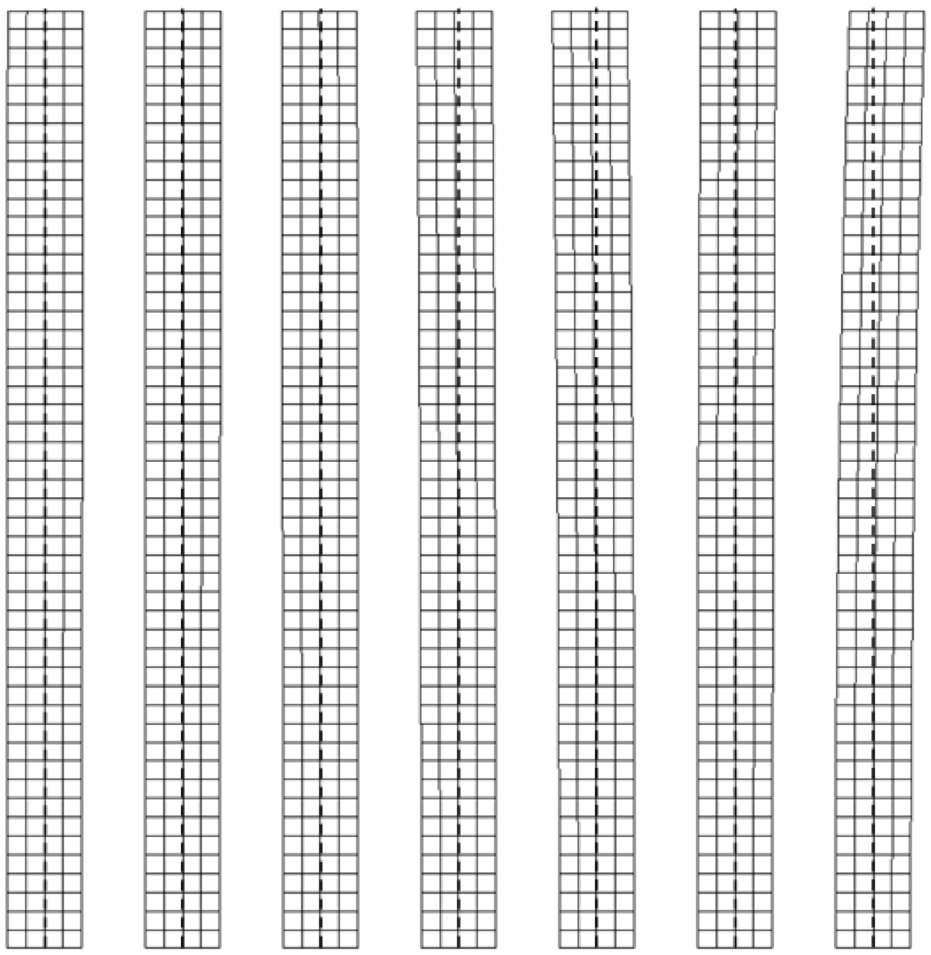
Figure 11: Mesh motion with 0.4m of amplitude and 2Hz of frequency for t = 0.05, 0.1, 0.15, 0.2, 0.25, 0.4 and 0.5s.

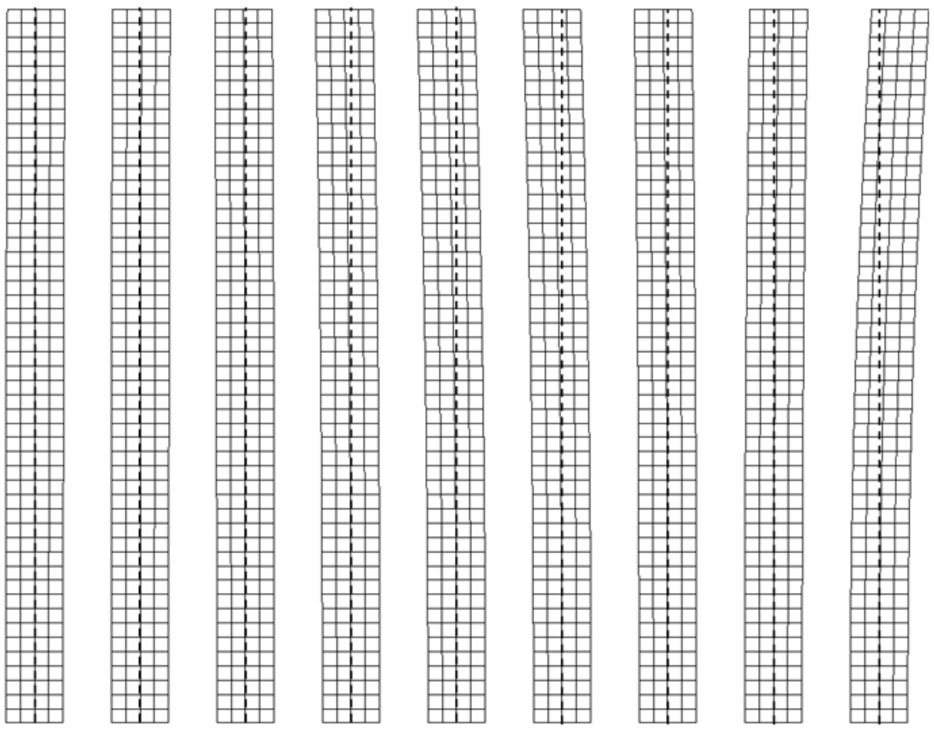
Figure 12: Mesh motion with 0.8m of amplitude and 2Hz of frequency for t = 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4 and 0.5s.

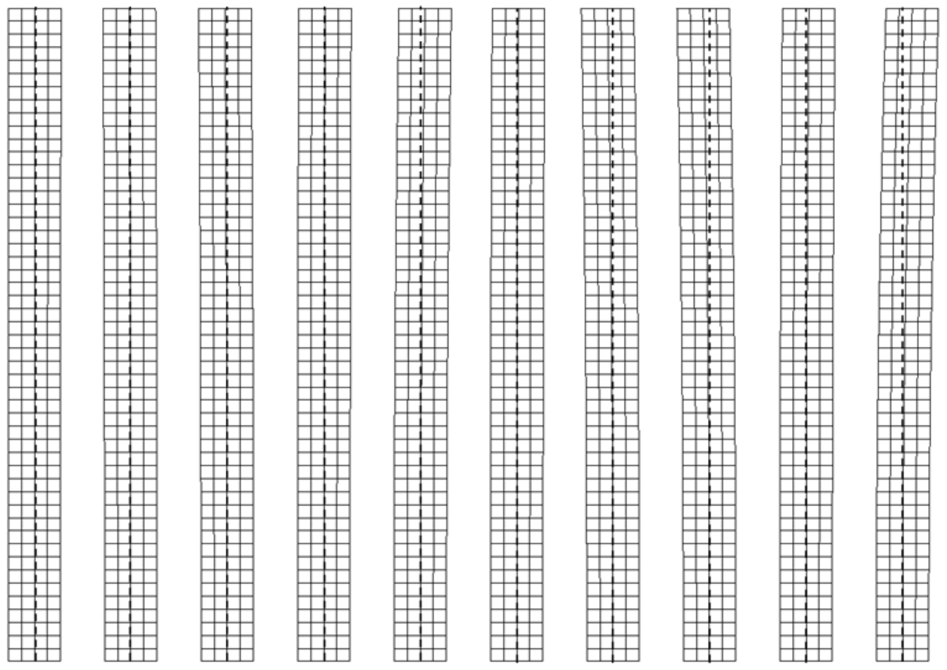

Figure 13: Mesh motion with 0.4m of amplitude and 4Hz of frequency for t = 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45 and 0.5s.

It can be seen that the two constants modify the motion in the following way:

- *a*: varies the total displacement in x direction. Comparing Figure 11 and 12, it can be observed how the displacement of the top part of the mesh in the last time step (0.5s) for *a* equal to 0.8 is bigger than the one for *a* with a value of 0.4.

- *frequency*: varies the speed of change; depending on the value it can be needed to change the write interval in the controlDict file to see the results of the change in the paraFoam. Comparing Figure 11 and 13, where Figure 13 has a *frequency* which is the double of the one for the case solved in Figure 11, it can be observed that the mesh is making more fluctuations around the vertical axis for the same time. The *frequency* constant is used inside the scaling function (Eq. 5) and it is similar to the one used for the *dynamicInkJetFvMesh* class but in this case it is going from -1 to 1, therefore, plotting the x position without considering the dependency on the y position the following picture is gotten:

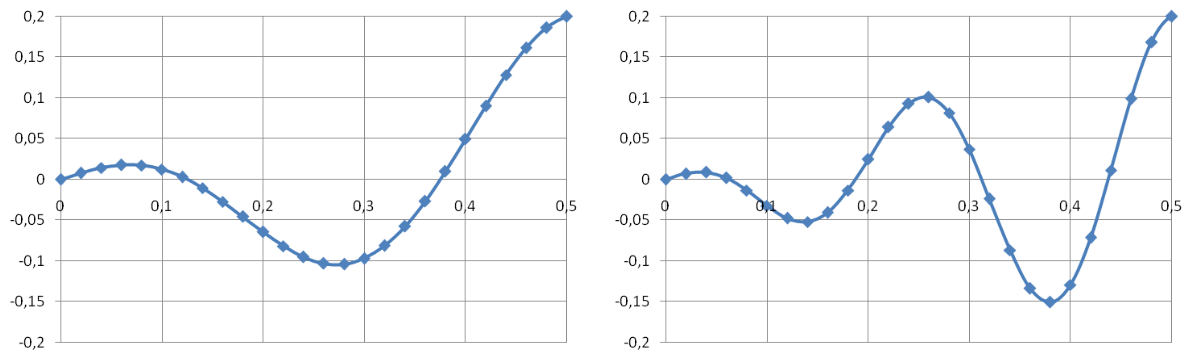

Figure 14: X position for 2Hz and 4Hz of frequency and time from 0 to 1s.

From Figure 14, it can be seen the motion that is applied to the mesh, which is a sinusoidal movement which is amplitude is increasing with time. As the motion is dependent on y, the displacement for the x coordinates shown in Figure 14 is multplied by the y coordinate squared therefore for the bottom points the movement is zero while for the top points the motion is the highest one.

5 CONCLUSIONS

It can be concluded that in order to have a mesh with motion there are two main ways two follow:

- The automatic mesh motion (*dynamicFvMesh*) with which the mesh topology does not change.
- The topological changes in the mesh (*topoChangerFvMesh*).

Inside this two options of mesh motion manipulation, there are different classes with different motions specified.

It has to be added that *dynamicInkJetFvMesh* defines a movement based on harmonic motion around a reference plane solved internally in the class, which means that a solver is not needed. The modification of this class, *dynamicMyClassFvMesh*, defines another motion, a sinusoidal one along y direction depending on y position.

Finally, as it has been seen with the *dynamicInkJetFvMesh* from the *dynamicFvMesh*, when the motion specified originally is not the demanded by the user, the class can be modified in order to define the desired motion.

6 REFERENCES

Håkan Nilsson (2009-09). *PhD course in CFD with OpenSource software, 2009*. Slides from the homepage of the course, which is given at Chalmers TH (Göteborg, Sweden).

OpenFOAM website – The Open Source CFD Toolbox. Retrieved November 2009 from: www.opencfd.co.uk/openfoam/

OpenFOAM Wiki. Retrieved November 2009 from:
www.openfoamwiki.net

CFD Online Forums about OpenFOAM. Retrieved November 2009 from:
www.cfd-online.com

Hrvoje Jasak and Henrik Rusche (2009-06). *Dynamic Mesh Handling in OpenFOAM*. Slides from the 4th OpenFOAM workshop (Montreal, Canada).

Pirooz Moradnia (2008). *A tutorial on how to use Dynamic Mesh solver IcoDyMFoam*. Report for the PhD course in OpenFOAM at Chalmers TH (Göteborg, Sverige).

Olivier Petit (2008). *Different ways to treat rotating geometries*. Report for the PhD course in OpenFOAM at Chalmers TH (Göteborg, Sverige).

Erik Bjerklund (2009). *A modification of the movingConeTopoFvMesh library*. Report for the PhD course in OpenFOAM at Chalmers TH (Göteborg, Sverige).

The OpenFOAM – The Open Source CFD Toolbox. Retrieved November 2009 from: http://foam.sourceforge.net

The 'SfR Fresh' Software Archive. Retrieved November 2009 from:
www.sfr-fresh.com

APPENDIX

The most important and used codes for this project are presented now:

dynamicInkJetFvMesh.H

```
#ifndef dynamicInkJetFvMesh_H
#define dynamicInkJetFvMesh_H
#include "dynamicFvMesh.H"
#include "dictionary.H"
#include "pointIOField.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
 namespace Foam
{
                           Class dynamicInkJetFvMesh Declaration
\*---------------------------------------------------------------------------*/
class dynamicInkJetFvMesh
:
    public dynamicFvMesh
{
    // Private data
        dictionary dynamicMeshCoeffs_;
        scalar amplitude_;
        scalar frequency_;
        scalar refPlaneX_;
        pointIOField stationaryPoints_;
    // Private Member Functions
        //- Disallow default bitwise copy construct
        dynamicInkJetFvMesh(const dynamicInkJetFvMesh&);
        //- Disallow default bitwise assignment
        void operator=(const dynamicInkJetFvMesh&);
public:
    //- Runtime type information
    TypeName("dynamicInkJetFvMesh");
    // Constructors
        //- Construct from IOobject
        dynamicInkJetFvMesh(const IOobject& io);
    // Destructor
        ~dynamicInkJetFvMesh();
    // Member Functions
        //- Update the mesh for both mesh motion and topology change
        virtual bool update();
};
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
} // End namespace Foam
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#endif
```

blockMeshDict

The code where the mesh is defined:

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

convertToMeters 0.1;
```

```
vertices
(
    (-0.006 0 0)
    (0 0 0)
    (0 0.075 0)
    (-0.006 0.075 0)
    (-0.006 0 0.001)
    (0 0 0.001)
    (0 0.075 0.001)
    (-0.006 0.075 0.001)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (4 50 1) simpleGrading (1 1 1)
);

edges
(
);

patches
(
    wall movingWall
    (
        (3 7 6 2)
        (0 4 7 3)
        (1 2 6 5)
    )
    wall fixedWalls
    (
        (1 5 4 0)
    )
    empty frontAndBack
    (
        (0 3 2 1)
        (4 5 6 7)
    )
);

mergePatchPairs
(
);

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

transportProperties

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

nu              nu [ 0 2 -1 0 0 0 0 ] 0.01;
```

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

system folder codes

controlDict

For myExample:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     icoFoam;
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         0.5;
deltaT          0.005;
adjustTimeStep  no; //added
maxCo           0.5;
writeControl    timeStep;
writeInterval   5;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression uncompressed;
timeFormat      general;
timePrecision   6;
runTimeModifiable yes;

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

For *myClassExample*, *endTime* is 1.

fvSchemes

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

gradSchemes
{
    default         Gauss linear;
    grad(p)         Gauss linear;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss linear;
}

laplacianSchemes
{
    default         none;
    laplacian(nu,U) Gauss linear corrected;
    laplacian((1|A(U)),p) Gauss linear corrected;
}
// ************************************************************************* //
```

fvSolution

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

PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}

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

0 folder codes

p

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

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

internalField   uniform 0;

boundaryField
{
    movingWall
    {
        type            zeroGradient;
    }

    fixedWalls
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            zeroGradient;
    }

}

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

U

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

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

internalField   uniform (0 0 0);

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

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

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

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

dynamicMeshDict

The code where the mesh class and the library for the class is specified. For myExample:

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

dynamicFvMesh dynamicInkJetFvMesh;

motionSolverLibs ("libfvMotionSolvers.so");

dynamicInkJetFvMeshCoeffs
{
        amplitude       0.06;
        frequency       2;
        refPlaneX       0;
}

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

And the code for myClassExample:

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

dynamicFvMeshLibs        ("libdynamicMyClassFvMesh.so");

dynamicFvMesh            dynamicMyClassFvMesh;

motionSolverLibs         ("libfvMotionSolvers.so");

dynamicMyClassFvMeshCoeffs
{
        a       0.4;
        frequency       2;
}

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

dynamicMyClassFvMesh.C

```
#include "dynamicMyClassFvMesh.H"
#include "addToRunTimeSelectionTable.H"
#include "volFields.H"
#include "mathematicalConstants.H"

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

namespace Foam
{
    defineTypeNameAndDebug(dynamicMyClassFvMesh, 0);
    addToRunTimeSelectionTable(dynamicFvMesh, dynamicMyClassFvMesh, IOobject);
}


// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //

Foam::dynamicMyClassFvMesh::dynamicMyClassFvMesh(const IOobject& io)
:
    dynamicFvMesh(io),
    dynamicMeshCoeffs_
    (
        IOdictionary
        (
            IOobject
            (
                "dynamicMeshDict",
                io.time().constant(),
                *this,
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            )
        ).subDict(typeName + "Coeffs")
    ),
    a_(readScalar(dynamicMeshCoeffs_.lookup("a"))),
    frequency_(readScalar(dynamicMeshCoeffs_.lookup("frequency"))),
    stationaryPoints_
    (
        IOobject
        (
            "points",
            io.time().constant(),
            meshSubDir,
            *this,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    )
{
    Info<< "Performing a dynamic mesh calculation: " << endl
        << "a: " << a_
        << " frequency: " << frequency_ << endl;
}

// * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * //

Foam::dynamicMyClassFvMesh::~dynamicMyClassFvMesh()
{}

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

bool Foam::dynamicMyClassFvMesh::update()
{
    scalar scalingFunction =
        (::cos(2*mathematicalConstant::pi*frequency_*time().value());

    Info<< "Mesh scaling. Time = " << time().value() << " scaling: "
        << scalingFunction << endl;

    pointField newPoints = stationaryPoints_;

    newPoints.replace
    (
        vector::X,
        stationaryPoints_.component(vector::X)+
        a_*time().values()*(stationaryPoints_.component(vector::Y))*
(stationaryPoints_.component(vector::Y))*scalingFunction
    );

    fvMesh::movePoints(newPoints);

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

    return true;
}

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

dynamicMyClassFvMesh.H

```
#ifndef dynamicMyClassFvMesh_H
#define dynamicMyClassFvMesh_H

#include "dynamicFvMesh.H"
#include "dictionary.H"
#include "pointIOField.H"

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

namespace Foam
{

/*---------------------------------------------------------------------------*\
                           Class dynamicMyClassFvMesh Declaration
\*---------------------------------------------------------------------------*/

class dynamicMyClassFvMesh
:
    public dynamicFvMesh
{
    // Private data

        dictionary dynamicMeshCoeffs_;

        scalar a_;
        scalar frequency_;

        pointIOField stationaryPoints_;


    // Private Member Functions

        //- Disallow default bitwise copy construct
        dynamicMyClassFvMesh(const dynamicMyClassFvMesh&);

        //- Disallow default bitwise assignment
        void operator=(const dynamicMyClassFvMesh&);

public:

    //- Runtime type information
    TypeName("dynamicMyClassFvMesh");

    // Constructors

        //- Construct from IOobject
        dynamicMyClassFvMesh(const IOobject& io);

    // Destructor

        ~dynamicMyClassFvMesh();

    // Member Functions

        //- Update the mesh for both mesh motion and topology change
        virtual bool update();
};


// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
} // End namespace Foam

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

#endif

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

icoDyMFoamMesh.C

The code of the solver modified in order to solve only the mesh motion:

```
#include "fvCFD.H"
#include "dynamicFvMesh.H"

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

int main(int argc, char *argv[])
{
#   include "setRootCase.H"

#   include "createTime.H"
#   include "createDynamicFvMesh.H"
#   include "readPISOControls.H"
#   include "initContinuityErrs.H"
#   include "createFields.H"
#   include "readTimeControls.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
    Info<< "\nStarting time loop\n" << endl;

    while (runTime.run())
    {
#       include "readControls.H"
#       include "CourantNo.H"

#       include "setDeltaT.H"

        runTime++;

        Info<< "Time = " << runTime.timeName() << nl << endl;

        mesh.update();

        runTime.write();

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }

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

    return(0);
}


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