CHALMERS UNIVERSITY OF TECHNOLOGY

Project work for the PhD course in
CFD WITH OPEN SOURCE SOFTWARE

Roll Motion of a Box and Interaction with Free-Surface

Author:

Arash Eslamdoost

arash.eslamdoost@chalmers.se

Sustainable Ship Propulsion
Shipping and Marin Technology Department
Chalmers University of Technology
Gothenburg, Sweden

Reviewers:

Håkan Nilsson
Karl Jacob Maus

January 2010

Table of Contents

1. Introduction

The objective of this tutorial is to investigate the capability of OpenFOAM-1.6.x in handling free-surface flows when there is some mesh deformation in the computational domain. This type of problem is often seen in ship motion modeling. To simplify the problem, a two-dimensional box located between the interface of water and air is considered. Moreover, to check the OpenFOAM's capability in handling dynamic mesh, the box will be set to roll in an oscillating mode which requires adapting the grid for each time step. The general presentation of the discussed modeling is seen in Figure 1.

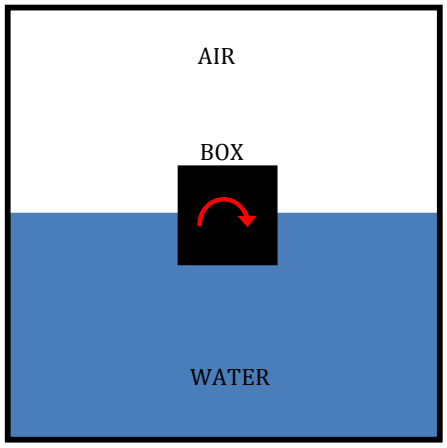

Figure 1. Schematic presentation of the intended modeling

It is common among OpenFOAM users to browse and find the available model among the existing OpenFOAM tutorials similar to the new intended model and base this new model on these previously linked solvers and libraries. The most similar available tutorial to the rolling box case is "sloshingTank2D", which is found in the following directory:

$FOAM_TUTORIALS/multiphase/interDyMFoam/ras/sloshingTank2D

In the "sloshingTank2D" tutorial there is a tank partially filled with water. It is possible to apply roll and other sorts of motions (e.g. sway and heave) to the whole tank together. Therefore, in this case, there is a moving mesh but actually there is not any change in cell volumes and arrays relative to each other. This case run with the "interDyMFoam" solver and according to the information provided in [1] this solver is applicable for two incompressible, isothermal immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach, with optional mesh motion and mesh topology changes including adaptive re-meshing. Therefore, "interDyMFoam" solver should be able to handle both interface tracking and mesh motion in the rolling box case.

As a first step to start the rolling box modeling, it has been tried to just change the geometry of the tank and replace it with the rolling box geometry. Now, it is required to generate a new grid and this is done with the "blockMesh" application. The detailed coordinate of the vertices, edges, patches, blocks and mesh adaption implemented in the blockMesh file is presented in appendix A. The generated mesh is rather coarse and in case that the accuracy of the solution is important this grid should be refined.

Throughout the rest of the document, the case inwhich the whole computational domain is supposed to have an angular oscillation about a defined axis is called "boxSolidOscillation" and the case, in which only the box is supposed to oscillate, is called "boxFreeOscillation". These cases can be found on the CFD with open source software course homepage at the following address:

http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2009/

Download and copy them to your run directory and then unpack the contents.

2. The boxSolidOscillation Case

The current case just deals with pure solid movement of the whole computational domain without any change in cells' volumes. It would be clearer to first discuss the SDA class, which takes the care of oscillating patches movement in this case, and then focus on the required settings to run the boxSolidOscillation case.

2.1.The SDA Class

Ship design analysis (SDA) mentioned in the "dynamicMeshDict" is a 3DoF motion function, which is applied to the entire computational domain. The SDA class comprises sinusoidal roll (rotation about x), heave (z-translation) and sway (y-translation) motions (see Figure 2) with changing amplitude and phase. It is found in the following directory:

$FOAM_SRC/dynamicFvMesh/solidBodyMotionFvMesh/solidBodyMotionFunctions

In "dynamicMeshDict" there are some coefficients required for defining the way that the computational domain is supposed to move. These coefficients are introduced in "SDA.H" as private data and are employed in motion equations presented in "SDA.C". Definition and dimension for each of these coefficients are presented in Table 1. Also, definitions for translational and rotational movement of a floating body are presented in Figure 2.

Table 1. Coefficients applied in "dynamicMeshDict"

Coefficient	Description	Type	Dimension
CofG	Center of gravity	vector	[m]
lamda	Model scale ratio	scalar	[-]
rollAmax	Max roll amplitude	scalar	[rad]
rollAmin	Min roll amplitude	scalar	[rad]
heaveA	Heave amplitude	scalar	[m]
swayA	Sway amplitude	scalar	[m]
Q	Damping Coefficient	scalar	[-]
Tp	Time Period for liquid	scalar	[sec]
Tpn	Natural Period of Ship	scalar	[sec]
Tpi	Current roll period	scalar	[sec]
dTi	Reference time step	scalar	[sec]
dTp	Increase in Tp per unit dTi	scalar	[-]

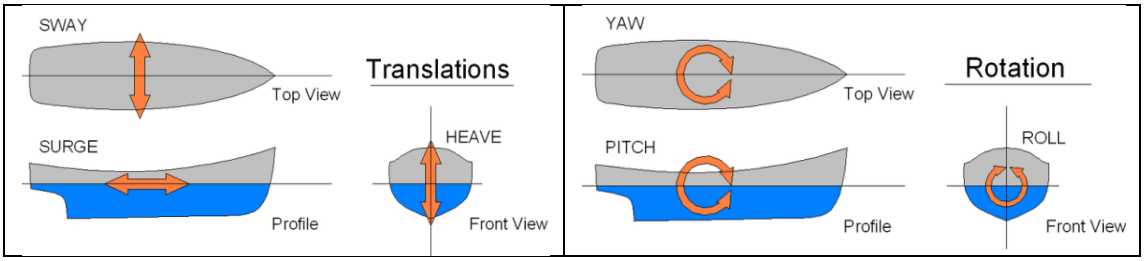

Figure 2. General presentation of a floating body movement.

Knowing the definition of the parameters indicate in the Table 1 and Figure 2, one can follow the algorithm applied in the code to oscillate the moving patches.

$$T_{pi} = T_p + dT_p \times \left(time / dT_i\right), \qquad \text{Current roll period[sec]}$$

$$\omega_r = \frac{2 \times \pi}{T_{pi}}; \qquad \text{Current frequencty[sec]}$$

$$r = \frac{dT_p}{dT_i},$$

$$u = T_p + r \times time,$$

$$ph_r = \frac{2 \times \pi \times \left(\left(\left(\frac{T_p}{u} - 1\right) + \log|u| - \log\left(T_p\right)\right)\right)}{r}. \qquad \text{Current phase for roll[rad]}$$

$$ph_s = ph_r + \pi. \qquad \text{Current phase for sway [rad]}$$

$$ph_h = ph_r + \frac{\pi}{2}. \qquad \text{Current phase for Heave[rad]}$$

$$rollA = \max\left\{rollA\max \cdot e^{\frac{-\sqrt{T_{pi} - T_{pn}}}{2Q}}, rolA\min\right\}; \qquad \text{roll amplitude [rad];}$$

$$T = \begin{bmatrix} 0 \\ swayA \times \left(\sin\left(\omega_r \times time + ph_s\right) - \sin\left(ph_s\right)\right) \\ heaveA \times \left(\sin\left(\omega_r \times time + ph_h\right) - \sin\left(ph_h\right)\right) \end{bmatrix}.$$

$$R = \text{quaternion} \begin{bmatrix} \text{rollA} \times \sin\left(\omega_{\text{r}} \times time + ph_r\right) \\ 0 \\ 0 \end{bmatrix}; \qquad \text{rotation[rad]}$$

$$\text{T}R = \text{septernion}\left[septernion\left(Cof\ G + T\right) \times R \times septernion\left(-Cof\ G\right)\right] \qquad \text{transformation [m]}$$

Quaternion class used to perform rotations in 3D space. Septernion class used to perform translations and rotations in 3D space. It is composed of a translation vector and rotation quaternion and as such has seven components hence the name "septernion" from the Latin to be consistent with quaternion rather than "hepternion" derived from the Greek. Mathematical descriptions of the quaternion and septernion classes are discussed in more detail in [4].

2.2. Setting up the boxSolidOscillation Case

To get started with this case, move into the "boxSolidOscillation" directory.

```
cd boxSolidOscillation
```

The "boxSolidOscillation" folder contains three sub-directories; "0", "constant", and "system". Some of the main contents of these sub-directories will be discussed in a little more detail.

The grid used in this case is created by running "blockMesh'. To check the grid properties open "constant/polyMesh/blockMesh". Also, the "blockMesh" file content is available in appendix A. Run "blockMesh" to generate the mentioned structured grid and then to check the mesh quality the "checkMesh" command could be applied.

```
blockMesh
checkMesh
```

Open and investigate the generated grid in paraview. It can bee seen that the generated mesh is located on the xy-plane. Default "dynamicMesh" is set in a way that the axis of angular oscillation should be located along x-axis; but, current generated grid is located on the xy-plane and its axis of oscillation is along the z-axis. According to this, the grid should be adapted by interchanging the current array of the x and z-axis. Following command line will do this:

```
rotateMesh "(0 0 1)" "(1 0 0)"
```

Actually, this command line means that the whole global coordinate system is going to rotate in a way that the first indicated vector (here, (0 0 1)) finally stands in the place of second defined vector (here, (1 0 0)). Figure 3 shows how the "rotateMesh" command works.

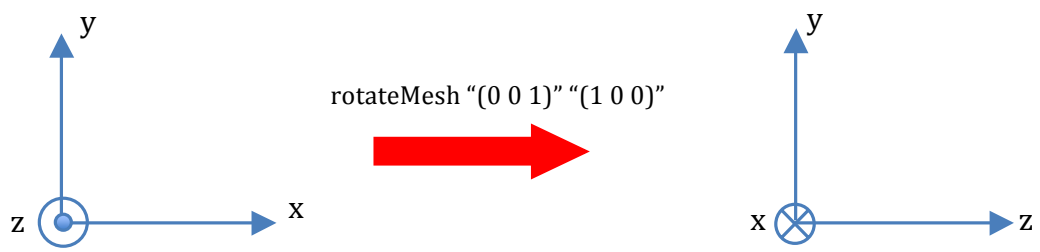

Figure 3. The left hand side coordinate system in the initial array and the right hand side coordinate system is the new array of this initial system after applying the rotateMesh command.

Now it is required to modify the files in the "0" directory for the current geometry and set the proper boundary conditions. Appendixes B, C, D, present the settings for velocity, pressure and alpha1[1] boundary condition. The boundaries of the box and tank are treated as non-moving walls. Pressure boundary conditions on walls are treated as "buoyantPressure[2]". Also, "zeroGradient" alpha1 has been set on boundaries.

To see how the initial phases distribution in the computational domain is defined open "system/setFields". Settings for the "setFields" application can be seen in Appendix E.

"In fvSolution, the PISO sub-dictionary contains elements that are specific to interFoam. There are the usual correctors to the momentum equation but also correctors to a PISO loop around the alpha1 phase equation. Of particular interest are the nAlphaSubCycles and cAlpha keywords. nAlphaSubCycles represents the number of sub-cycles within the alpha1 equation; sub-cycles are additional solutions to an equation within a given time step. It is used to enable the solution to be stable without reducing the time step and vastly increasing the solution time" [2]. Here 3 sub-cycles has been specified, which means that the alpha1equation is solved in 3×one-third-length time steps within each actual time step.

"The cAlpha keyword is a factor that controls the compression of the interface where: 0 corresponds to no compression; 1 corresponds to conservative compression; and, anything larger than 1, relates to enhanced compression of the interface" [2].

```
PISO
{
momentumPredictor no;
nCorrectors2;
nNonOrthogonalCorrectors0;
nAlphaCorr1;
nAlphaSubCycles3;
cAlpha1.5;
correctPhino;

pRefPoint  (0 -4 0);
pRefValue1e5;
}
```

[1] alpha1 is the phase fraction of phase 1 in a multi-phase system, alpha in the control dictionaries refers to any phase. The reason for the "inconsistency" is to ensure that the discretisation schemes are the same for all phases.

[2] buoyantPressureFvPatchScalarField, the new buoyancy pressure boundary condition in OF1.6 now supports p (static pressure) and pd for backward compatibility. pd replaced by static pressure p. All solvers in which buoyancy affects might be strong have been converted from using pd to p with improved numerics to give equally good accuracy and stability. This change is prompted by the need to remove the confusion surrounding the meaning and purpose of pd. In the code ,buoyantPressureFvPatchScalarField , if the pressure boundary condition is set to be "buoyantPressure", the pressure on that boundary, pd, computes applying the following equation.

pd = p-rho*g*h

which p is the static pressure, rho is the fluid density and the height of the fluid column on this boundary patch.

Initial distribution of phases (water and air) is fixed by applying the “setFields” command.

```
cp 0/alpha1.org 0/alpha1
SetFields
```

Mesh movement in this case is applied through the available dictionary in “constant” directory, which is called “dynamicMeshDict”.

```
dynamicFvMeshsolidBodyMotionFvMesh;

solidBodyMotionFvMeshCoeffs
{
solidBodyMotionFunction SDA;
SDACoeffs
   {
CofG          ( 0 0 0 );
Lamda          50;
rollAmax      0.22654;
rollAmin      0.10472;
heaveA         0;
swayA          0;
Q              2;
Tp             13.93;
Tpn            11.93;
dTi            0.059;
dTp           -0.001;
   }
}
```

In the current case just the roll motion of the computational domain is of interest and therefore the heave and sway coefficients are set to zero.

The case is now ready to run with the “interDyMFoam” solver.

```
interDyMFoam
```

This application will start the solution by reading the scheme of mesh motion and then reading the initial fields such as g, p, alpha1, U and boundary conditions. Consequently, the incompressible transport model and the turbulence model are set. The presented lines in the following gray box are the report of this procedure that appears on the screen after issuing the “interDyMFoam” command. Moreover, output from this application is presented for one time step. According to the “nAlphaSubCycles” value of 3, it can be seen that three consequent iterations occurs when solving the multidimensional universal limiter for explicit solution (MULES).

```
Create time
Create mesh for time = 0

Selecting dynamicFvMeshsolidBodyMotionFvMesh
Selecting solid-body motion function SDA

Reading g
Reading field p

Reading field alpha1

Reading field U

Reading/calculating face flux field phi

Reading transportProperties

Selecting incompressible transport model Newtonian
Selecting turbulence model type laminar
time step continuity errors : sum local = 0, global = 0, cumulative = 0
GAMGPCG:  Solving for pcorr, Initial residual = 1, Final residual = 2.2321e-13, No Iterations 1
time step continuity errors : sum local = 1.38355e-11, global = 5.61669e-30, cumulative = 5.61669e-30
Courant Number mean: 1.3569e-11 max: 1.72618e-10

Starting time loop
.
Courant Number mean: 0.0930747 max: 0.472539
deltaT = 0.00714286
Time = 0.128571

solidBodyMotionFunctions::SDA::transformation(): Time = 0.128571 transformation: ((0 00) (0.999235
(0.0391004 0 0)))
Execution time for mesh.update() = 0.01 s
MULES: Solving for alpha1
Liquid phase volume fraction = 0.5  Min(alpha1) = -4.02279e-20  Max(alpha1) = 1
MULES: Solving for alpha1
Liquid phase volume fraction = 0.5  Min(alpha1) = -2.78346e-21  Max(alpha1) = 1
MULES: Solving for alpha1
Liquid phase volume fraction = 0.5  Min(alpha1) = -6.47083e-22  Max(alpha1) = 1
GAMG:  Solving for p, Initial residual = 0.0681279, Final residual = 0.000220229, No Iterations 2
time step continuity errors : sum local = 1.0173e-06, global = 1.50675e-17, cumulative = -1.83543e-16
GAMGPCG:  Solving for p, Initial residual = 0.00106785, Final residual = 1.71041e-09, No Iterations 7
time step continuity errors : sum local = 7.4766e-12, global = 1.50911e-17, cumulative = -1.68452e-16
ExecutionTime = 1.18 s  ClockTime = 1 s
```

The result of the computation is presented in two sets of phase distribution and velocity field plots in Figure 4.

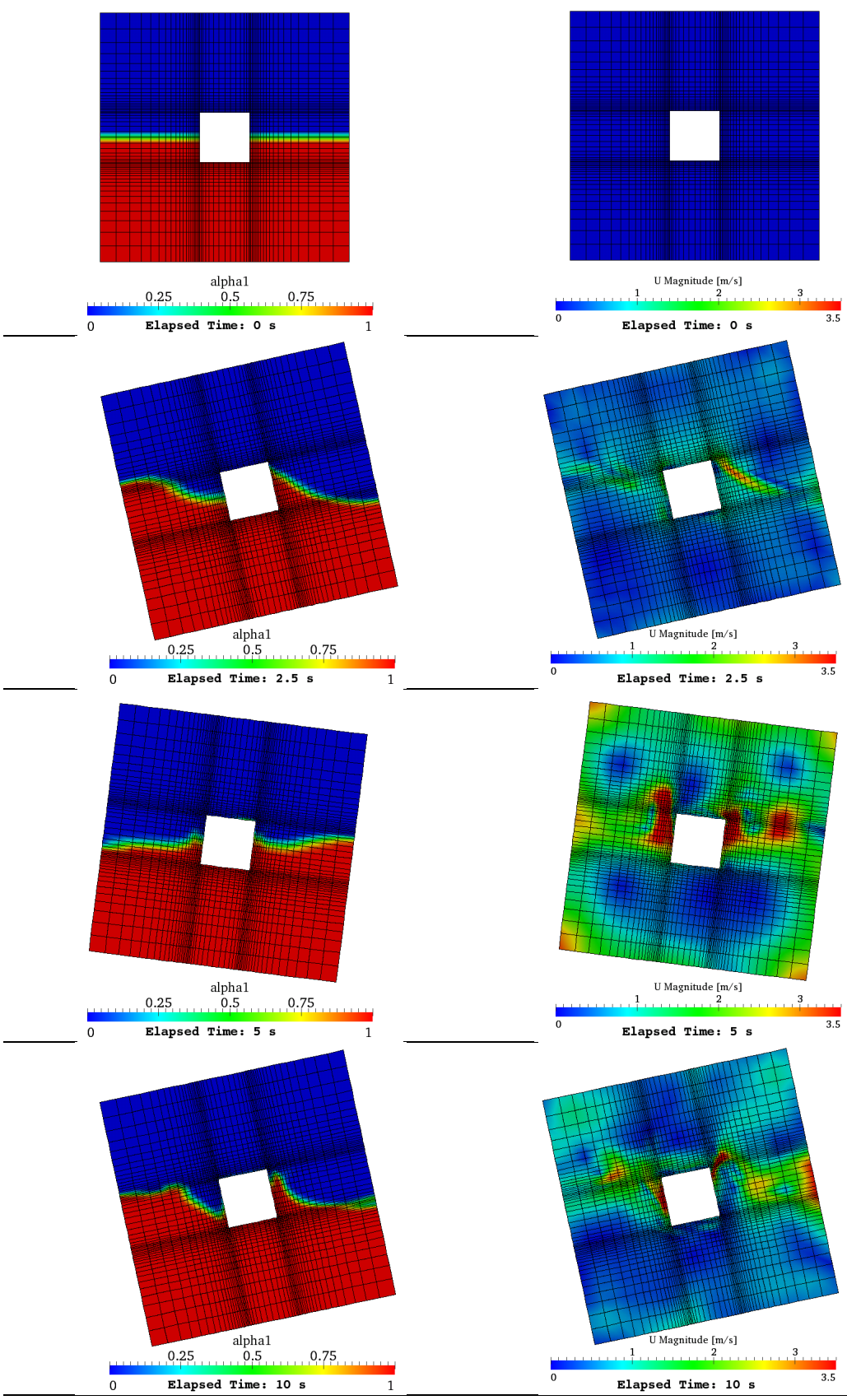

Figure 4. **Column A presents thealpha1 distribution and column B shows the velocity field for the corresponding time**

3. The boxFreeOscillation

In this section, in contrast with the previous one, instead of moving the whole computational grid as a solid body just the inner box is going to have an angular oscillation. So, it is required to set up another application for the dynamic mesh treatment. At this stage, the difference between the boxFreeOscillation case and the boxSolidOscillation case is the settings in "dynamicMeshDict" located in the "constant" directory and the required boundary motion input data in the "pointMotionU" file located in the "0" directory, as well. The "pointMotionU" file contains some input values that control the movement of the oscillating patches applied in the oscillation function discussed in 3.1.1. Moreover, as the next step, it is required to modify the "fvSolution" dictionary and define the cell motion solver. These modification and settings are going to be discussed in the following sections.

3.1. meshMotionLib

At the moment, there are two available libraries in OpenFoam-1.6.x, which can produce the desired oscillating for specified patches. They are called "angularOscillatingVelocity" and "angularOscillatingDisplacement" and both can be found in the following path:

$FOAM_SRC/fvMotionSolver/pointPatchFields/derived/

Both of these libraries are applicable for the current case based on user demand. Here, the "angularOscillatingVelocity" library is going to be used. This library is attached to the files available in the "CFD with open source software" homepage in "meshMotionLib" directory. To be able to make some modification in this library and prepare the library to a specific type of motion it would be better to edit the "files" and "option" files and recompile it to the user library. A general description to "angularOscillatingVelocity" is presented in the following paragraphs.

3.1.1. angularOscillatingVelocity

The source files for this library can be found in "dynamicMeshDict" folder of the provided files in the "CFD with open source software" homepage.

```
cd dynamicMeshDict
```

The original ".H" and ".C" files copied from the "$FOAM_SRC/fvMotionSolver/pointPatchFields/derived/angularOscillatingVelocity" path were named "libOscillatingVelocityPointPatchVectorField.H" and "libOscillatingVelocityPointPatchVectorField.C" but in order to make a distinction between the original library and this one the string "libOscillatingVelocity" in the names of ".H" and ".C" files has been replaced with "libMyOscillatingVelocity". This replacement has to be done inside these source files as well as "files" and "options" in the "Make" directory. The resulting "files" and "options" files are shown in the following gray boxes.

files

```
libMyOscillatingVelocityPointPatchVectorField.C

LIB = $(FOAM_USER_LIBBIN)/libMyOscillatingVelocityPointPatchVectorField
```

options

```
EXE_INC = \
    -I$(LIB_SRC)/triSurface/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/dynamicMesh/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/fvMotionSolver/lnInclude

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

The main part of this library which defines the movement of the patches is presented as a member function in "libOscillatingVelocityPointPatchVectorField.C". Here, the velocity of each point on a specific patch is calculated for each time step. One may define a new function to move the patches by editing the presented function in this file and if required the constructors in ".H" file. The mechanism of the oscillating velocity function can be seen in Figure 5. In the specific case that the origin of rotation is located in the center of the box the result of the motion should be a pure roll around box's center axis. The function of this oscillation for patches, which exists as a member function, is presented in the following gray box.

libOscillatingVelocityPointPatchVectorField.C

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

voidlibMyOscillatingVelocityPointPatchVectorField::updateCoeffs()
{
if (this->updated())
    {
return;
    }

constpolyMesh& mesh = this->dimensionedInternalField().mesh()();
const Time& t = mesh.time();
constpointPatch& p = this->patch();

scalar angle = angle0_ + amplitude_*sin(omega_*t.value());
vectoraxisHat = 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()
    );

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

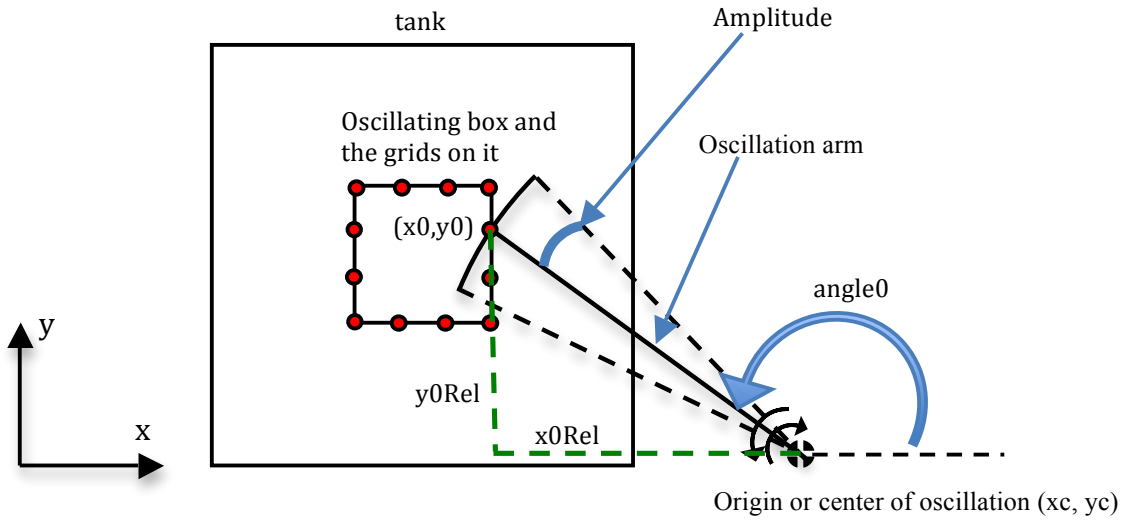

Figure 5. Mechanism of oscillating velocity

In this function, first, according to the given angular oscillation amplitude and frequency, a new angle for the oscillation arm (an arm which connects the grids on oscillating body to the center of oscillation, Figure 5) is computed in each time step employing the following sinusoidal equation.

$$angle = angle0 + ampliude \cdot \sin(\omega \cdot t);$$

where angle is the new calculated angle of the oscillation arm; angle0 is the initial angle of the oscillation arm which oscillation occurs about this angle; amplitude is the amplitude of the angular oscillation and actually acts similar to a weight factor and controls the new angle in each time step; omega (ω) is frequency of the angular oscillation and *t* is the actual run time of the simulation.

Consequently, after computing the new angle of oscillation arm, updated position of the grids on the moving body is computed as following:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \frac{1}{deltaT}\left[\begin{bmatrix} x0 \\ y0 \\ z0 \end{bmatrix} + \begin{bmatrix} x0_{rel} \\ y0_{rel} \\ z0_{rel} \end{bmatrix}\left(\cos(angle)-1\right) + \hat{A} \times \begin{bmatrix} x0_{rel} \\ y0_{rel} \\ z0_{rel} \end{bmatrix}\sin(angle) + \left(\hat{A} \cdot \begin{bmatrix} x0_{rel} \\ y0_{rel} \\ z0_{rel} \end{bmatrix}\right)\left(1-\cos(angle)\right)\hat{A} - p.localPoints()\right]$$

and

$$\hat{A} = \begin{bmatrix} x_{axis} \\ y_{axis} \\ z_{axis} \end{bmatrix} \Big/ \left\| \begin{bmatrix} x_{axis} \\ y_{axis} \\ z_{axis} \end{bmatrix} \right\|, \qquad p0Rel = \begin{bmatrix} x0_{rel} \\ y0_{rel} \\ z0_{rel} \end{bmatrix} - \begin{bmatrix} xc \\ yc \\ zc \end{bmatrix};$$

where, p0 is the coordinate for the oscillating grid point, (x0, y0, z0); p0Rel is the relative coordinate for the grids on the moving body in according to the center of oscillation, (xc, yc, zc); axisHat is a unit vector which defines the axis of oscillation.

It can be seen that, the updated positions of the moving grids on oscillating body are divided by the time step, deltaT, which means that the output of this function is a velocity. Actually, output of this function is the main difference between the libraries "angularOscillatingVelocity" and "angularOscillatingDisplacement" introduced in the section 3.1. Indeed, applying the "angularOscillatingVelocity" library (current), user defines the oscillation velocity of each of the moving points on the oscillating boundaries, but when using "angularOscillatingDisplacement" the displacement of the moving grids on oscillating boundaries are controlled.

To make the current library available for the other applications, it should be compiled through the following command.

```
wmake libso
```

There should be a link to this new library in the "controlDict" file to let the employed applications know about it. Inserting the following line to the "controlDict" does this.

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

3.2. Setting up the boxFreeOscillation Case

After preparing the meshMotion library, it is required to do some settings in the "fvSoloution" dictionary and define the solver for mesh motion application. This is done by adding the following lines to the "solvers" sub-directory in the "fvSolution" file.

fvSolution

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

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

"solvers" 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[1].

Table 2. Solver Options

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

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[1].

Table 3. Preconditioner options

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

The other distinction in the "boxFreeOscillation" case compared to the "boxSolidOscillation" case is in the "dynamicMeshDic" dictionary located in the "constant" directory. The new dictionary is presented in the following gray box.

dynamicMeshDict

```
FoamFile
{
version    2.0;
formatascii;
classdictionary;
location    "constant";
objectmotionProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dynamicFvMeshdynamicMotionSolverFvMesh;
motionSolverLibs ("libfvMotionSolvers.so");
solvervelocityLaplacian;

diffusivity uniform;
```

It is apparent from the "dynamicMeshDict" file that there are two parameters that are important in dynamic mesh manipulation which are "solver" and "diffusivity" scheme. Available models for the "solver" are:

- displacementLaplacian
- velocityLaplacian
- SBRStress

which in the "boxFreeOscillation" case is set to be velocityLaplacian. Available options for diffusivity models are presented in Table 4. Further information on the "solver" and "diffusivity" models can be found in [5].

Table 4. Diffusivity Models

quality-based methods	distance-based methods*
• uniform	• linear
• directional	• quadratic
• motionDirectional	• exponential
• inverseDistance	

*These models are used with "inverseDistance" method

Now it is required to define the motion of the moving patches (box) in the "pointMotionU" file located in the "0" directory. The "pointMotionU" file, which is appeared in the following gray box, contains some input values that control the movement of the oscillating patches applied in the oscillation function discussed in the section 3.1.1.These parameters are presented in Table 5 and in Figure 5 as well. Axis of rotation in this Figure is z-axis. Origin is the coordinate of the center of oscillation and angle0 is the angle of oscillation arm (Figure 5), which is considered as a reference angle for the oscillation, and oscillation occurs about this reference angle. Amplitude and omega are amplitude and frequency of the angular oscillation about the reference angle, angle0.

Table 5. Motion control Parameters in "pointMotionU"

Parameter	Type	Description	Dimension
axis	Vector	Axis of Rotation	m
origin	Vector	Center of Rotation	m
angle0	Scalar	Oscillation Occurs about this reference angle	rad
amplitude	Scalar	Amplitude of Angular Oscillation	rad
omega	Scalar	Angular Oscillation Frequency	rad/sec

pointMotionU

```
FoamFile
{
version 2.0;
formatascii;
classpointVectorField;
objectpointMotionU;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions     [0 1 -1 0 000];
internalField   uniform (0 00);
boundaryField
{
   tank
   {
      typefixedValue;
       value         uniform (0 00);
   }
   box
   {
      typeibMyOscillatingVelocityPointPatchVectorField;
      axis          (1 0 0);
      origin        (0 100 0);
      angle0        0;
      amplitude  0.01;
      omega         5;
      value uniform (0 00);
   }
    front
   {
      type          empty;
   }
    back
   {
      type          empty;
   }
}
```

After selecting a dynamic mesh solver for the case, it is time for setting the initial

phase (alpha) distribution in the computational domain. This step is totally similar to the "setFields" in previous case. Therefore, "setFieldsDict" dictionary settings should be similar to what is shown in appendix D, and then, running the following commands sets the phase distribution.

```
cp 0/alpha1.org 0/alpha1
setFields
```

Now the needed settings are done and the case is ready to be simulated applying interDyMFoam.

```
interDyMFoam
```

Results of the "boxFreeOscillating" case with presented settings are depicted in Figure 6. Paying attention to the phase distribution graphs one can see that the maximum amount of alpha1 is greater than 1, which is not physical and should be identical to 1. This problem appears to originate from the MULES implicit solver for the interDyMFoam and should be fixed. There is a thread in the OpenFOAM forum discusses exactly the same problem [6]. One of the users has modified the MULES to solve the problem. The modified code is available in this thread.

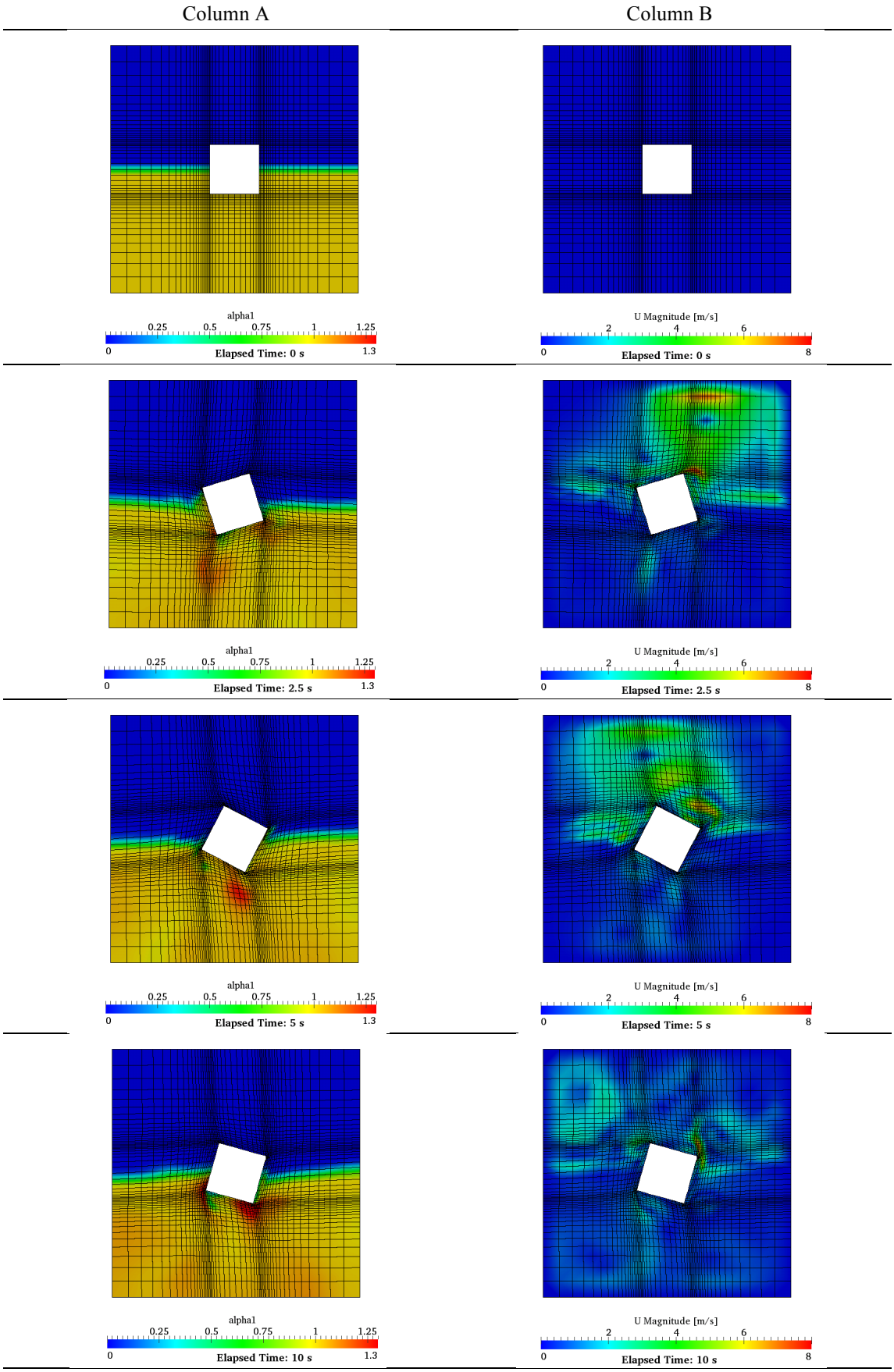

Figure 6. Column A presents the alpha1 distribution and column B shows the velocity field for the corresponding time

References:

1. *http://www.cfd-online.com/Forums/openfoam/*

2. http://www.openfoam.com/docs/user/damBreak.php

3. EysteinnHelgason, *Point-wise deformation of mesh patches*, Project work for the PhD course "CFD with Open Source Software", Chalmers University of Technology, Gothenburg, Sweden, Spring 2009.

4. Erik Ekedahl, *6-DOF VOF-solver without Damping in OpenFOAM*, Project work for the PhD course "CFD with Open Source Software", Chalmers University of Technology, Gothenburg, Sweden, Winter 2008.

5. PiroozMoradnia, *A tutorial on how to use Dynamic Mesh solver IcoDyMFOAM*, Project work for the PhD course "CFD with Open Source Software", Chalmers University of Technology, Gothenburg, Sweden, Spring 2008.

6. http://www.cfd-online.com/Forums/openfoam-solving/58021-interdymfoam-gives-strange-gamme-if-cell-volume-changes-due-mesh-motion.html

Appendix A: Grid Generation Applying blockMesh

Location of this file in the code:
boxSolidOscillation /constant / polyMesh /blockMesh
and also in
boxFreeOscillation /constant / polyMesh /blockMesh

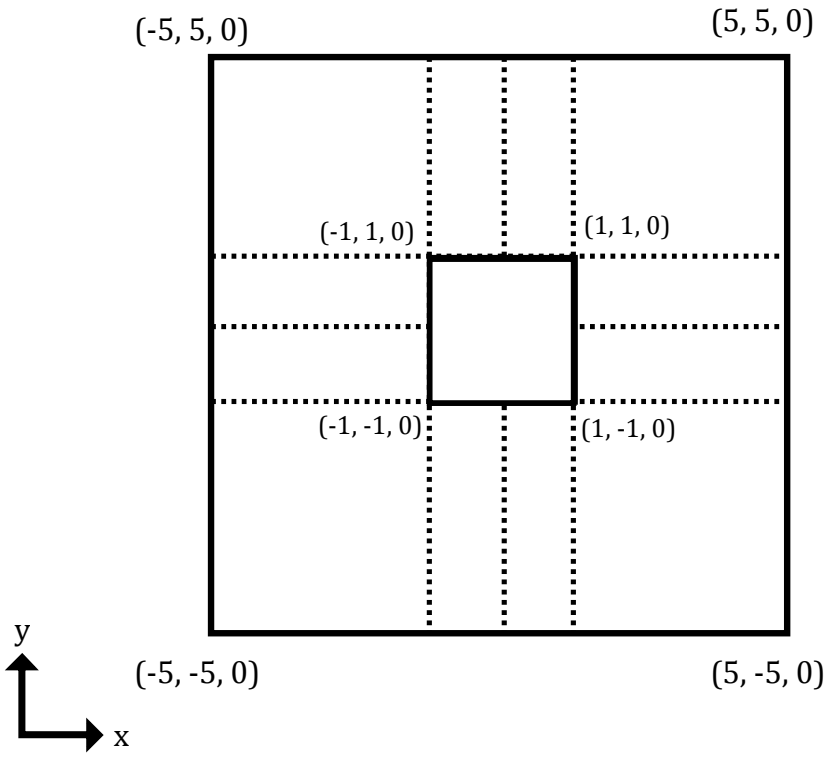

Figure A. 1: Dimensions of the generated grid

The following lines of the codedescribe the grids generated and employed in this tutorial.

```
// Parametric description
vertices
(
// Back Vertexes /////////
   (-5 -5 -0.5)   //Vertex No. = 0
   (-1 -5 -0.5)   //Vertex No. = 1
   (0 -5 -0.5)    //Vertex No. = 2
   (1 -5 -0.5)    //Vertex No. = 3
   (5 -5 -0.5)    //Vertex No. = 4

   (-5 -1 -0.5)   //Vertex No. = 5
   (-1 -1 -0.5)   //Vertex No. = 6
   (0 -1 -0.5)    //Vertex No. = 7
```

```
    (1 -1 -0.5)   //Vertex No. = 8
    (5 -1 -0.5)   //Vertex No. = 9

    (-5 0 -0.5)   //Vertex No. = 10
    (-1 0 -0.5)   //Vertex No. = 11
    (1 0 -0.5)    //Vertex No. = 12
    (5 0 -0.5)    //Vertex No. = 13

    (-5 1 -0.5)   //Vertex No. = 14
    (-1 1 -0.5)   //Vertex No. = 15
    (0 1 -0.5)    //Vertex No. = 16
    (1 1 -0.5)    //Vertex No. = 17
    (5 1 -0.5)    //Vertex No. = 18

    (-5 5 -0.5)   //Vertex No. = 19
    (-1 5 -0.5)   //Vertex No. = 20
    (0 5 -0.5)    //Vertex No. = 21
    (1 5 -0.5)    //Vertex No. = 22
    (5 5 -0.5)    //Vertex No. = 23

//Front Vertexes ///////
    (-5 -5 0.5)   //Vertex No. = 24
    (-1 -5 0.5)   //Vertex No. = 25
    (0 -5 0.5)    //Vertex No. = 26
    (1 -5 0.5)    //Vertex No. = 27
    (5 -5 0.5)    //Vertex No. = 28

    (-5 -1 0.5)   //Vertex No. = 29
    (-1 -1 0.5)   //Vertex No. = 30
    (0 -1 0.5)    //Vertex No. = 31
    (1 -1 0.5)    //Vertex No. = 32
    (5 -1 0.5)    //Vertex No. = 33

    (-5 0 0.5)    //Vertex No. = 34
    (-1 0 0.5)    //Vertex No. = 35
    (1 0 0.5)     //Vertex No. = 36
    (5 0 0.5)     //Vertex No. = 37

    (-5 1 0.5)    //Vertex No. = 38
    (-1 1 0.5)    //Vertex No. = 39
    (0 1 0.5)     //Vertex No. = 40
    (1 1 0.5)     //Vertex No. = 41
    (5 1 0.5)     //Vertex No. = 42

    (-5 5 0.5)    //Vertex No. = 43
    (-1 5 0.5)    //Vertex No. = 44
    (0 5 0.5)     //Vertex No. = 45
    (1 5 0.5)     //Vertex No. = 46
    (5 5 0.5)     //Vertex No. = 47
);

//Blocks//////
blocks
(
        hex (0 1 6 5 24 25 30 29)    (20 20 1)  simpleGrading  (0.2 0.2 1)   //Block No. = 0
        hex (1 2 7 6 25 26 31 30)    (10 20 1)  simpleGrading  (3 0.2 1)       //Block No. = 1
        hex (2 3 8 7 26 27 32 31)    (10 20 1)  simpleGrading  (0.3333 0.2 1)   //Block No. = 2
```

```
        hex (3 4 9 8 27 28 33 32)     (20 20 1)  simpleGrading  (5 0.2 1)       //Block No. = 3
        hex (5 6 11 10 29 30 35 34)   (20 10 1)  simpleGrading  (0.2 3 1)   //Block No. = 4
        hex (8 9 13 12 32 33 37 36)   (20 10 1)  simpleGrading  (5 3 1)       //Block No. = 5
        hex (10 11 15 14 34 35 39 38) (20 10 1)  simpleGrading  (0.2 0.3333 1)   //Block No. = 6
        hex (12 13 18 17 36 37 42 41) (20 10 1)  simpleGrading  (5 0.3333 1)       //Block No. = 7
        hex (14 15 20 19 38 39 44 43) (20 20 1)  simpleGrading  (0.2 5 1)   //Block No. = 8
        hex (15 16 21 20 39 40 45 44) (10 20 1)  simpleGrading  (3 5 1)       //Block No. = 9
        hex (16 17 22 21 40 41 46 45) (10 20 1)  simpleGrading  (0.3333 5 1)   //Block No. = 10
        hex (17 18 23 22 41 42 47 46) (20 20 1)  simpleGrading  (5 5 1)       //Block No. = 11
);
//Edges////////
edges
(
);
//Patches///////
patches
(
wall box
   (
      (6 30 35 11)
      (11 35 39 15)
      (15 39 40 16)
      (16 40 41 17)
      (17 41 36 12)
      (12 36 32 8)
      (8 32 31 7)
      (7 31 30 6)
   )


wall tank
   (
      (0 24 29 5)
      (5 29 34 10)
      (10 34 38 14)
      (14 38 43 19)
      (19 43 44 20)
      (20 44 45 21)
      (21 45 46 22)
      (22 46 47 23)
      (23 47 42 18)
      (18 42 37 13)
      (13 37 33 9)
      (9 33 28 4)
      (4 28 27 3)
      (3 27 26 2)
      (2 26 25 1)
      (1 25 24 0)
   )

empty front
   (
      (24 25 30 29)
      (25 26 31 30)
      (26 27 32 31)
      (27 28 33 32)
      (29 30 35 34)
```

```
        (32 33 37 36)
        (34 35 39 38)
        (36 37 42 41)
        (38 39 44 43)
        (39 40 45 44)
        (40 41 46 45)
        (41 42 47 46)
    )

empty back
    (
            (0 5 6 1)
        (1 6 7 2)
        (2 7 8 3)
        (3 8 9 4)
        (5 10 11 6)
        (8 12 13 9)
        (10 14 15 11)
        (12 17 18 13)
        (14 19 20 15)
        (15 20 21 16)
        (16 21 22 17)
        (17 22 23 18)
    )
);
// ********************************************************************** //
```

Appendix B: Velocity setting on boundaries at time 0

Location of this file in the code: boxFreeOscillation / 0 / U

```
FoamFile
{
version     2.0;
formatascii;
classvolVectorField;
object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 000];

internalField   uniform (0 00);

boundaryField
{
front
   {
type          empty;
   }
back
   {
type          empty;
   }
box
   {
typemovingWallVelocity;
value          uniform (0 00);
   }
tank
   {
typemovingWallVelocity;
value          uniform (0 00);
   }
}
```

Appendix C: Pressure setting on boundaries at time 0

Location of this file in the code: boxFreeOscillation / 0 / p

```
FoamFile
{
version     2.0;
formatascii;
classvolScalarField;
object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 0 000];

internalField   uniform 0;

boundaryField
{
box
   {
typebuoyantPressure;
value          uniform 0;
   }

tank
   {
typebuoyantPressure;
value          uniform 0;
   }

front
   {
type          empty;
   }
back
   {
type          empty;
   }
}
```

Appendix D: alpha1 setting on boundaries at time 0

Location of this file in the code: boxFreeOscillation / 0/ alpha1

```
FoamFile
{
version     2.0;
formatascii;
classvolScalarField;
object      alpha1;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 000000];
internalField   uniform 0;
boundaryField

{
box
{
        typezeroGradient;
}
tank
{
        typezeroGradient;
}
front
{
        type         empty;
}
back
        {
        type         empty;
        }
}
```

Appendix E: alpha1 field setting using “setFields” application

Location of this file in the code: boxFreeOscillation / system/ setFields

```
defaultFieldValues
(
volScalarFieldValue alpha1 0
);

regions
(
boxToCell
   {
box ( -0.5 -5 -5 ) ( 0.5 5 0 );
fieldValues
      (
volScalarFieldValue alpha1 1
      );
   }
);
```