Project work for the PhD course in OpenFOAM

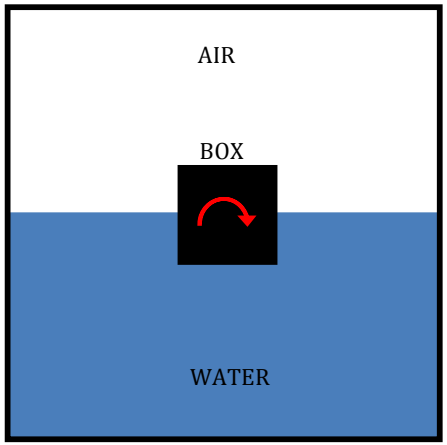
Roll Motion of a Box and Interaction with Free-Surface

Arash Eslamdoost

arash.eslamdoost@chalmers.se

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

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 computational domain.



Schematic presentation of the intended modeling

The most similar available tutorial to the rolling box case is “sloshingTank2D”, which could be found in the following directory:

$FOAM_TUTORIALS/multiphase/interDyMFoam/ras/sloshingTank2D

The case, which the whole computational domain is supposed to have an angular oscillation about a

defined axis, is called “boxSolidOscillation” and the case, which only the box is supposed to oscillate, is called “boxFreeOscillation”. These cases could be found in the OpenFOAM Course Homepage in the following address:

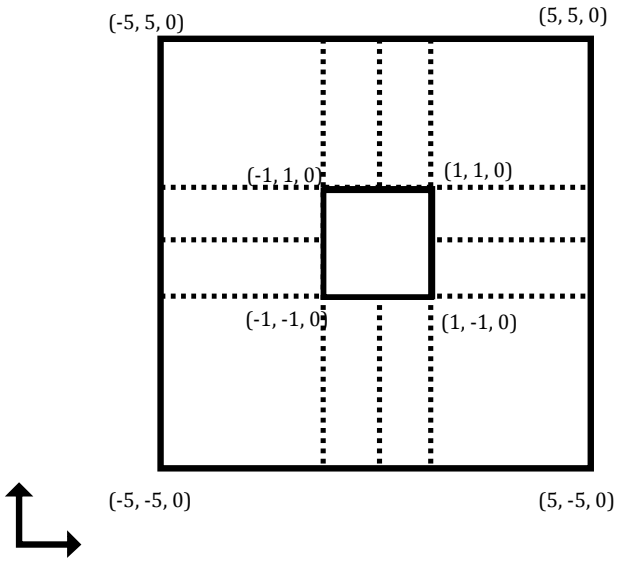
http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2009

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

```
cd $FOAM_RUN
wget  http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2009/arasheslamdoost/arashProjectFiles.tgz
tar xzf arashProjectFiles.tgz
cd boxSolidOscillation
```

boxSolidOscillation case

- Current case just deals with pure solid movement of whole computational domain without having any change in cells' volume.
- Grid applied in this case is produced applying blockMesh. To check the grid properties open "constant/polyMesh/blockMesh".



Dimensions of the generated grid

```
blockMesh
checkMesh
```

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

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

internalField   uniform (0 0 0);

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

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

dimensions      [1 -1 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    box
    {
        type            buoyantPressure;
        value           uniform 0;
    }
    tank
    {
        type            buoyantPressure;
        value           uniform 0;
    }
    front
    {
        type            empty;
    }
    back
    {
        type            empty;
    }
}
```

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      alpha1;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 0 0 0 0 0 0];
internalField   uniform 0;
boundaryField

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

fvSolution, the PISO sub-dictionary

```
PISO
{
    momentumPredictor           no;
    nCorrectors                 2;
    nNonOrthogonalCorrectors    0;
    nAlphaCorr                  1;
    nAlphaSubCycles             3;
    cAlpha                      1.5;
    correctPhi                  no;

    pRefPoint                   (0 -4 0);
    pRefValue                   1e5;
}
```

The nAlphaSubCycles and cAlpha keywords. nAlphaSubCycles represents the number of sub-cycles within the α_1 equation; sub-cycles are additional solutions to an equation within a given 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.

Phases (water and air) initial distribution 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**".

```
dynamicFvMesh   solidBodyMotionFvMesh;

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;
   }
}
```

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]
dTi	Reference time step	scalar	[sec]
dTp	Increase in Tp per unit dTi	scalar	[-]

Now, the case is ready to run applying "interDyMFoam" application.

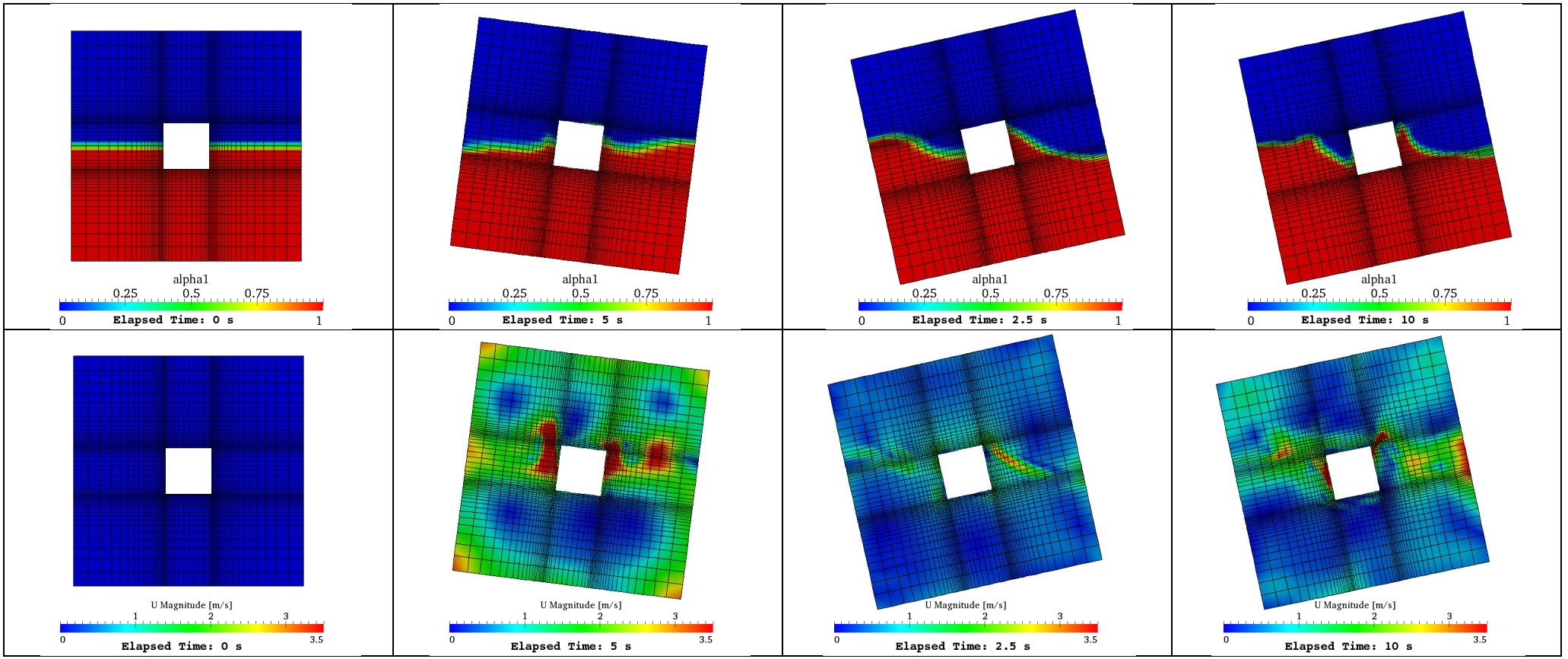
```
interDyMFoam
```

```
Create time
Create mesh for time = 0

Selecting dynamicFvMesh solidBodyMotionFvMesh
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

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 0 0)
(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
```

This application will start the solution first by reading the scheme of mesh motion and then setting the initial fields such as g, p, alpha1, U and boundary conditions. Consequently, the incompressible transport model and turbulence model are set.

Moreover, output of running this application is presented for one of the time steps. According to setting the nAlphaSubCycles to 3, it could be seen that three consequent iterations occurs for solving the multidimensional universal limiter for explicit solution (MULES).

Result for boxSolidOscillation



boxFreeOscillation

In this section, just the inner box is going to have an angular oscillation. At this stage, difference between "boxFreeOscillation" case and "boxSolidOscillating" is between the settings of "dynamicMeshDict" and consequently the required boundary motion input data in "pointMotionU". Moreover, it is required to modify the "fvSolution" dictionary and define the cell motion solver.

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 could be found in the following path:

$FOAM_SRC/fvMotionSolver/pointPatchFields/derived/

In the current case, "angularOscillatingVelocity" library is going to be used.

angularOscillatingVelocity

The source files for this library could be found in "dynamicMeshDict" folder of the provided materials at OpenFOAM Course 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 not to make some distinction among 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. Finally, the "files" and "options" files should be the same as following ones.

files

```
libMyOscillatingVelocityPointPatchVectorField.C

LIB = $(FOAM_USER_LIBBIN)/
libMyOscillatingVelocityPointPatchVectorField
```

options

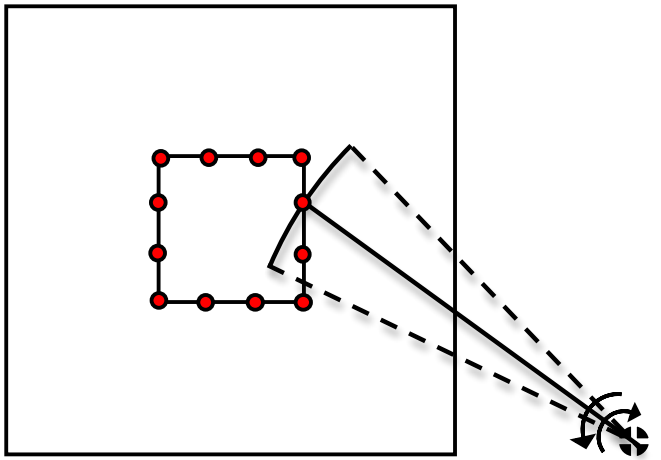
```
EXE_INC = \
    -I$(LIB_SRC)/triSurface/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/dynamicMesh/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/fvMotionSolver/lnInclude

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

angularOscillatingVelocity

The main part of this library which defines the movement of the patches is presented as 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.

libOscillatingVelocityPointPatchVectorField.C

```
void angularOscillatingVelocityPointPatchVectorField::updateCoeffs()
  if (this->updated())
  {
     return;
  }
  const polyMesh& mesh = this->dimensionedInternalField().mesh()();
  const Time& t = mesh.time();
  const pointPatch& p = this->patch();
  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()
  );
  fixedValuePointPatchField<vector>::updateCoeffs();
}
```



In a 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.

angularOscillatingVelocity

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

```
wmake libso
```

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

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

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 purpose is done by adding the following lines to the solvers in the "fvSolution".

fvSolution

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

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

angularOscillatingVelocity

dynamicMeshDict

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

dynamicFvMesh   dynamicMotionSolverFvMesh;
motionSolverLibs ("libfvMotionSolvers.so");
solver velocityLaplacian;

diffusivity uniform;
```

Available models for the "solver"

- displacementLaplacian
- velocityLaplacian
- SBRStress

Available "diffusivity" models

Quality-based methods	Distance-based methods*
• uniform	• linear
• directional	• quadratic
• motionDirectional	• exponential
• inverseDistance	

*These models are used with "inverseDistance" method

angularOscillatingVelocity

Now it is required to define the motion of the moving patches (box) in "pointMotionU" file located in "0" directory. This input file is required to calculate the coordinate of each point on a specified moving patch during time variation.

pointMotionU

```
dimensions      [0 1 -1 0 0 0];
internalField   uniform (0 0 0);

boundaryField
{
  tank
  {
     type          fixedValue;
     value         uniform (0 0 0);
  }
  box
  {
     type ibMyOscillatingVelocityPointPatchVectorField;
     axis (1 0 0);
     origin (0 100 0);
     angle0 0;
     amplitude 0.01;
     omega 5;
     value uniform (0 0 0);
  }

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

Setting the phase field

After setting a dynamic mesh solver for the case, it is the time for setting the 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 the appendix 4 then it is possible to run the following command:

```
Cp 0/alpha1.org 0/alpha1
setFields
```

At the moment needed setting are done and the case is ready to start the calculations applying interDyMFoam.

```
interDyMFoam
```

Paying attention to the phase distribution graphs one can see that the maximum amount of alpha1 is greater than 1, which is not practical and should be identical to 1. This problem appears to be originated from the MULES implicit solver in the interPhaseChangeFoam,

Result for boxFreeOscillation