Connecting OpenFOAM with MATLAB
Project presentation
CFD with OpenSource Software

Johannes Palm

Shipping and Marine Technology / Hydrodynamics,
Chalmers University of Technology,
Gothenburg, Sweden

2012-10-22

Running the case

Begin by copying the tutorial case directory to your run directory.

```
OF21x
cp -r  $FOAM_TUTORIALS/multiphase/interDyMFoam/ras/floatingObject $FOAM_RUN
cd $FOAM_RUN/floatingObject
```

Open the `system/controlDict` file and change `endTime` from 6s to 2s to save some time. Then run the case using the Allrun script.

```
./Allrun
```

Case setup description

The Allrun-script

`blockMesh` (util.)

`topoSet` (util.)

`subSetMesh` (util.)

`setFields` (util.)

`interDyMFoam` (solver)

Libraries needed

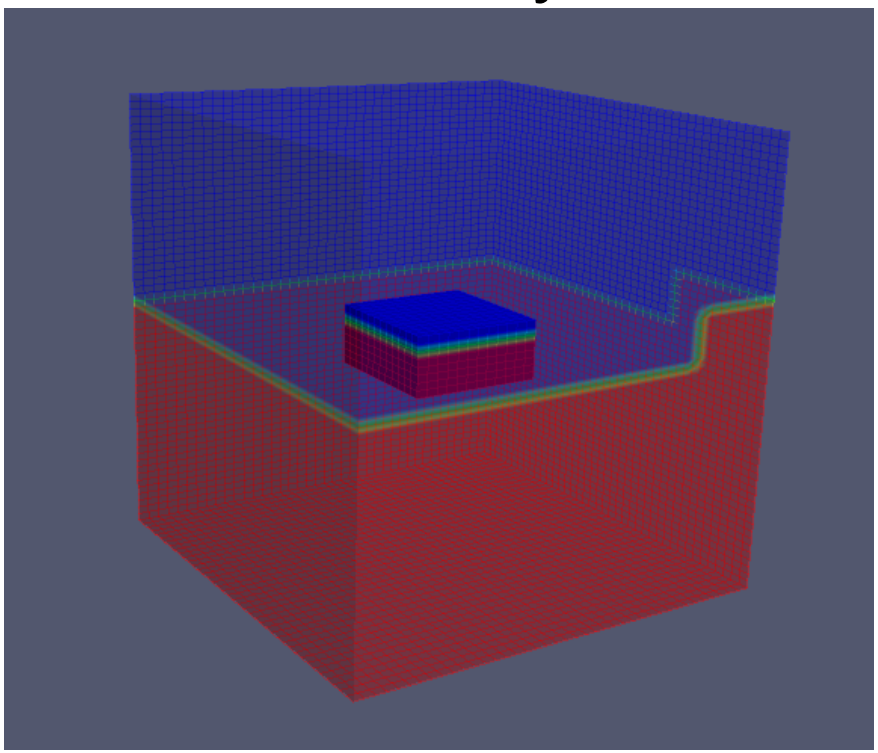
`libOpenFOAM.so`

`libincompressibleRASModels.so`

`libfvMotionSolvers.so`

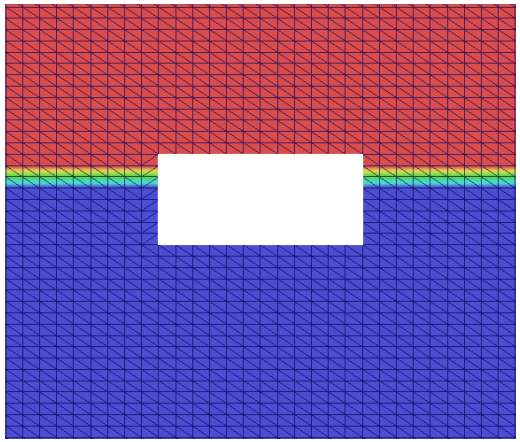
`libforces.so`

Geometry

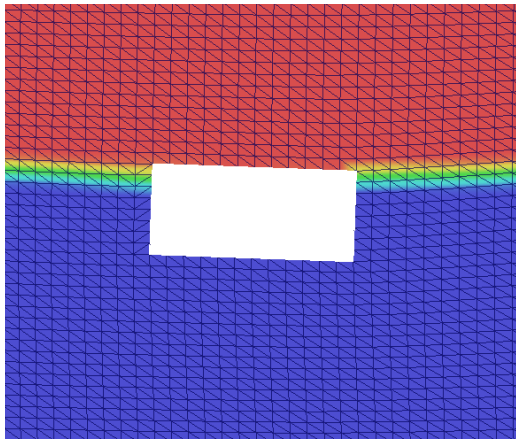


Case setup at t=0

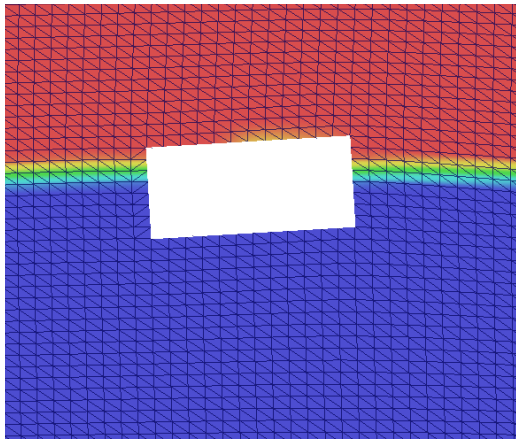
Results



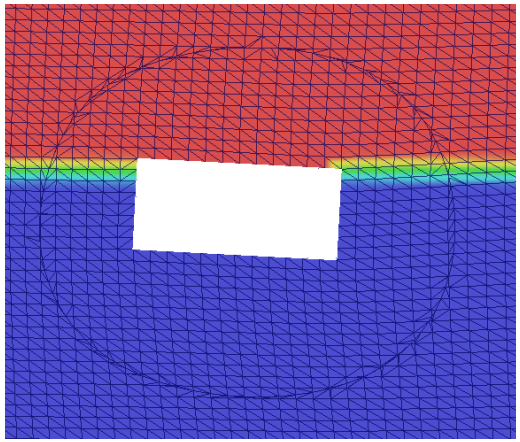
Solution in plane y=0, at t=0



Solution in plane y=0, at t=1s



Solution in plane y=0, at t=1.5s



Solution in plane y=0, at t=2.0s

Adding a restraint

The easiest way to insert a restraint in the sixDoFRigidBodyDisplacement function object is to find another tutorial where the restraint functionality is used.

```
pushd incompressible/pimpleDyMFoam/wingMotion/wingMotion2D_pimpleDyMFoam/
cd 0.org
#Open File pointDisplacement and copy restraints section#
popd
#Insert restraints in floatingObject part of 0.org/pointDisplacement#
```

Change the values of the `verticalSpring` to better suite this physical setting, and delete the rest of the restraints.

```
restraints{
    verticalSpring
    {
        sixDoFRigidBodyMotionRestraint linearSpring;

        linearSpringCoeffs
        {
            anchor          (0.5 0.5 0.2);
            refAttachmentPt (0.5 0.5 0.45);
            stiffness       100;
            damping         0;
            restLength      0.25;
        }
    }
}
```

Now the case can be run again, with a linear spring attached to the floating object.

Introduction

One could use the MATLAB compiling command `mex` to generate the MATLAB code into a C/C++ files.
This functionality is not compatible with the latest version of the gcc compiler, at least not in my case, so another approach is presented here.

The MATLAB engine.h file is included in the program and compiled from within OpenFOAM using wmake.

Key syntax demonstration

The following program is a very simple example of a MATLAB connection, as it would look from within a solver.

```
#include<iostream>
#include<cmath>
#include "engine.h"
using namespace std;
int main(){
// Open a portal to MATLAB through a pointer to an Engine object //
   Engine *eMatlabPtr=engOpen(NULL);

// Create an empty mxArray of size [1,1] //
    mxArray *aMxArray = mxCreateDoubleMatrix(1,1,mxREAL);

// Get pointer to the actual value of the mxArray //
    double *aPtr = mxGetPr(aMxArray);

// Set value of mxArray//
    aPtr[0] = 5;

// Set the value of matlab parameter a to the value of aMxArray. //
    engPutVariable(eMatlabPtr,"a",aMxArray);

// Execute commands in MATLAB //
    engEvalString(eMatlabPtr,"b=a.^2; plot(0:0.1:2*pi,sin(0:0.1:2*pi)); pause(5);");
```

Script continued on next slide...

Key syntax demonstration continued

```
// Collect the result from MATLAB back to the C++ code //
    mxArray *bMxArray = engGetVariable(eMatlabPtr,"b");
    double *bPtr = mxGetPr(bMxArray);

// Print the result //
    cout << "5*5 = " << bPtr[0] << endl;

// Close the pipe to Matlab //
    engClose(eMatlabPtr);
    return 0;
}
```

Compiling with wmake

Creating the Make directory, the following must be included in the Make/files and Make/options files.

Make/options

```
EXE_INC = \
  -Wl,-rpath,/chalmers/sw/sup64/matlab-2011b/bin/glnxa64 \
       -I/chalmers/sw/sup64/matlab-2011b/extern/include

EXE_LIBS = \
  -L/chalmers/sw/sup64/matlab-2011b/bin/glnxa64 \
  -leng \
  -lmx
```

Make/files

```
testMatlabScript.C

EXE = myExecutableMatlabScript
```

Creating a dynamic library

In order to use this effectively within OpenFOAM, the MATLAB pipe will be established using a dynamic library. Create the following directory:

```
mkdir $WM_PROJECT_USER_DIR/src/externalPipe
```

Locate the supplied directories named:

`myFirstMatlabPipe`

`Make`

Copy these and place them both in the externalPipe directory.
Let's look at the files and then compile the library.

```
cd $WM_PROJECT_USER_DIR/src/externalPipe
wmake libso
```

Creating a new restraint type

Now, in this setting, MATLAB will be called from a restraint of the floatingObject. This means that a modified restraint needs to be created. We will use the linearSpring restraint as a demonstration.

```
foam
cp -r --parents src/postProcessing/functionObjects/forces/\
pointPatchFields/derived/sixDoFRigidBodyMotion/sixDoFRigidBodyMotionRestraint/\
linearSpring $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR/src/postProcessing/functionObjects/forces/\
pointPatchFields/derived/sixDoFRigidBodyMotion/sixDoFRigidBodyMotionRestraint/
```

On the course homepage there is also a folder called mylinearSpring. Copy this into the current directory and remove the folder linearSpring.
Let's look at the files and then compile the library.

```
wmake libso
```

Case file modification

Now we proceed as earlier in the course when we added a modified boundary condition library. The new library `mylinearSpring.so` is added to the library list of system/controlDict.

```
run
cd floatingObject/
#insert "mylinearSpring.so" in the library list of the system/controlDict file #
```

We must also change the `0.org/pointDisplacement` file so that we specify that the new restraint will be used.

```
restraints{
    verticalSpring{
        sixDoFRigidBodyMotionRestraint mylinearSpring;

        mylinearSpringCoeffs
        {
            anchor          (0.5 0.5 0.2);
            refAttachmentPt (0.5 0.5 0.45);
            stiffness       100;
            damping         0;
            restLength      0.25;
        }
    }
}
```

New case files needed

In this case there is only one last detail left, and that is to create the matlab scripts needed by mylinearSpring.

mooringScript.m

```
%----- Calculate effective extension -----%
if inputFromCpp <=0
    outputToCpp = 0;
else
   outputToCpp = 50*inputFromCpp;
end

%----- Plot the results runtime -----%
plot(ii,inputFromCpp,'k.',ii,outputToCpp,'rs');
hold on
if ii==1
  title('Evolution of actual and effective extension');
  legend('actual extension','used extension');
  xlabel('Number of time steps [-]')
  ylabel('Spring extension [m]');
end
```

saveInfoScript.m

```
zHeave=[zHeave;inputFromCpp];

save("matlabSession.mat","ii","zHeave");
```

Now the case can be run again using the Allclean and Allrun scripts.

Conclusion, Discussion and Questions

I would like to point out that the MATLAB functionality can be incorporated into OpenFOAM in a more general way than has been shown in this tutorial.

The matlabPipeClass object needs to have individual member functions depending on what data type that should be sent and returned. But, for simple calculations that send double arrays and returns double arrays, only a single member function in the matlabPipeClass is needed, and the rest can supposedly be controlled from your case-specific m-files and the application library (in this case the mylinearSpring restraint).

Thank you for your attention!