Implementation of Continuous forcing IBM in OpenFOAM

Divyaprakash

July 18, 2025

1 Introduction

The implementation is in the ESI version of openfoam (openfoam2012).

The idea is to modify the *icofoam* solver in OpenFOAM (OF) to enable it to run continuous forcing Immersed Boundary Method simulations. The equations that are solved in *icofoam* solver are as follows.

```
// Momentum predictor
        fvVectorMatrix UEqn
        (
            fvm::ddt(U)
          + fvm::div(phi, U)
          - fvm::laplacian(nu, U)
        );
```

The very first step would be to add a source term to the above equation that comes in because of the forces that are exerted by the immersed boundary. To keep the equations dimensionally homogeneous the units of F must be $\mathrm{m\,s^{-2}}$. Thus the force applied must be per unit volume, which is taken care of by the 3D discrete dirac delta interpolation function (δ_d) and per unit density. Thus the modified code is given below.

```
// Momentum predictor
        fvVectorMatrix UEqn
        (
            fvm::ddt(U)
          + fvm::div(phi, U)
          - fvm::laplacian(nu, U)
          - F // IBM Source Term
        );
```

In order for OF to recognize F, which is a volumetric vector field, it's entry should be added to the source code in the file *createFields.H*.

```
Info<< "Reading field F\n" << endl;
volVectorField F
(
    IOobject
    (
        "F",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
```

With the above changes one can then compile his own custom solver. This link provides an introduction on that topic. Make sure to place your custom solver in the path provided by `$(FOAM_USER_APPBIN)`

2 Implementation: Basic

At the very least, one can implement the forcing due to a point force in the domain. To do this one needs to know the location of the point where it needs to apply the force and also the location of the center of all

cells in the mesh. The cell centers can be generated by the following post-processing utility provided by OF as follows.

```
    postProcess -func writeCellCentres
```

Now one can loop over these cell centers applying the δ_d function to spread the point force in the domain assigning values to each cell. This data can then be written to a file which is then inputted a internal field to the *F* file in the *0/* directory as follows. We perform these calculations using a MATLAB script.

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

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

internalField   nonuniform List<vector>
400
(
...
(0.000000 0.000000 0.000000)
(0.062500 0.000000 0.000000)
(0.125000 0.000000 0.000000)
(0.062500 0.000000 0.000000)
...
(0.000000 0.000000 0.000000)
(0.000000 0.000000 0.000000)
(0.125000 0.000000 0.000000)
(0.250000 0.000000 0.000000)
(0.125000 0.000000 0.000000)
...
(0.000000 0.000000 0.000000)
(0.000000 0.000000 0.000000)
(0.062500 0.000000 0.000000)
(0.125000 0.000000 0.000000)
(0.062500 0.000000 0.000000)
...
(0.000000 0.000000 0.000000)
(0.000000 0.000000 0.000000)
);
```

The compiled solver is then run and the spread of the point force is visualized in Figure 1.

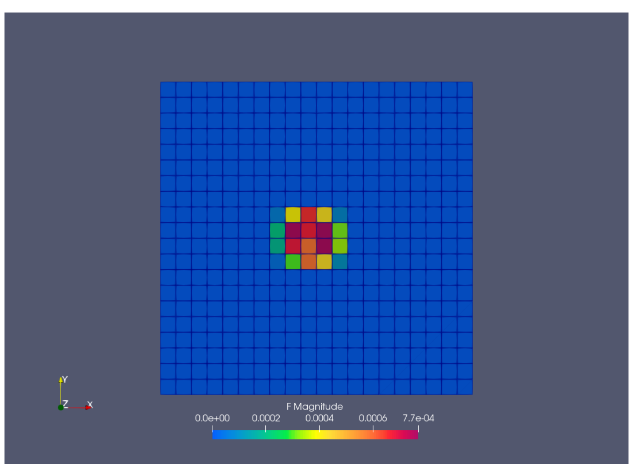

Figure 1: The point force at a Lagrangian point spread to the Eulerian grid by the δ_d function

Now this can be internal field can then be updated at every time step depending on the changing location of the point force. However this is very tedious and an alternative way is outlined below.

3 Implementation: Advanced

A structural solver is needed which given deformations calculates the force at every point in the structure. We already have this solver written in the Fortran programming language. A separate fortran program is written which acts as the interface between the solid and the fluid (OF) solver. This program is compiled as a shared library and linked to the OF solver. This shared library has function calls which enable the following.

1. Fetches immersed boundary (IB) point locations from the solid solver
2. Fetches forces at each of these IB points
3. Obtains velocity information at each of the IB points from the OF solver and updates the IB points locations in the solid solver

```
    // Declare the external subroutine
extern "C" {
    void getpositions(double *pposx, double *pposy, double *pposz, int *noelpts);
    // subroutine getpositions(XC,YC,ZC) bind(C)
    void calculateforces(double *pfx, double *pfy, double *pfz, int *noelpts);
    // subroutine calculateforces(FXC,FYC,FZC) bind(C)
    void updatepositions(double *pvx, double *pvy, double *pvz, double *dt, int *noelpts);
    // subroutine updatepositions(U,V,W,dt) bind(C)
}
```

We would want to implement the IBM in such a way that the OF solver should ask for location of all the points that constitute an immersed boundary. The solver then goes through each of these points in a loop and creates a list of neighbours that are contained within the support of the δ_d function. Once a list of neighbours for all the IB points have been generated. The solver enquires for the forces at each of these points. Then using the δ_d function it spreads these forces to it's Eulerian fluid mesh. The solver then sends back the velocity information at each of the IB points back to the interface.

3.1 Cell's Neighbours

It becomes essential to create a list of cells in the neighbourhood of a cell on which the δ_d function will have non-zero values. Thus it reduces the conputation time which would be spent in looping over all the cells of the mesh. The neighbours are found and stored in a list as shown below.

```
label cellIndex = mesh.findCell(rr);
DynamicList<Foam::label> appended;
appended.append(cellIndex);

// Get the neighbouring cells 4 times
for (int l = 0; l < 4; ++l) {
    int nn = appended.size();
    for (int m = 0; m < nn; ++m) {
        labelList neighbors = mesh.cellCells()[appended[m]];
        // appended.append(neighbors);
        appended.append(neighbors);
    }
    // Get rid of repeating values and also sort the array
    Foam::inplaceUniqueSort(appended);
}
```

3.2 Technicalities

3.2.1 Interoperability

The solid solve is in fortran and the fluid solver is in C++ programming language. However, data interoperability by Fortran is supported and thus it is possible to call subroutines in Fortran from C++.

3.2.2 Shared library

```
gfortran -c -fPIC fem3d.f90 -o fem3d.o
gfortran -c -fPIC soft_particles.f90 -o soft_particles.o
gfortran -shared -o libsoftparticles3D.so soft_particles.o fem3d.o
```

The above creates the shared library which is linked to OF solver in the `make\options` file as given below.

```
EXE_LIBS = \
    -lfiniteVolume \
    -lmeshTools \
    -L/home/divyaprakash/my_solvers/ofcil3D/fem2d -lsoftparticles3D
```

It should be noted that the shared library path must be exported to the environment every time a new shell is launched to run the program as given below.

```
    export LD_LIBRARY_PATH=/home/divyaprakash/my_solvers/ofcil3D/fem2d:$LD_LIBRARY_PATH
```

4 Post-processing

The solid solver writes the files containing the coordinates of the points that make up the solid. The data is in the form of x,y and z coordinates of all the points. We need to create vtk files using this data so that it can be opened alongside the fluid flow data in paraview. A matlab script is available on matlab file exchange website. We can use it's functions to generate vtk files for visualization. Sometimes the files created by the solid solver are not synchronous with the OF files. In that case remember to use the Temporal shift scale filter in paraview.

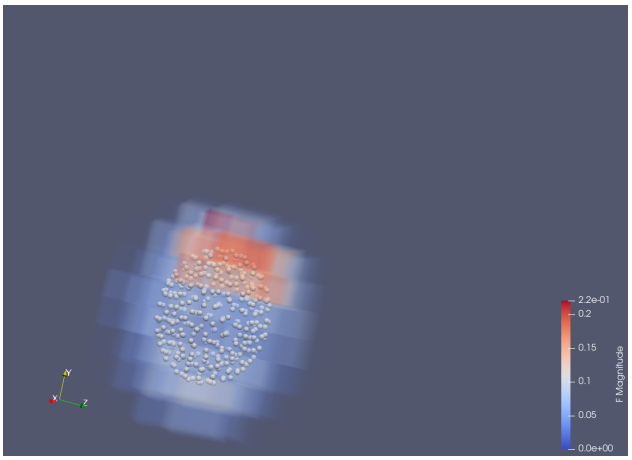

Figure 2: Caption