icoStructFoam
A Fluid-Structure Interaction Solver

Philip Evegren

April 14, 2008

icoStructFoam is a fluid-structure interaction (FSI) solver made for small deformations. As it is stated under the description paragraph of the source code file (icoStructFoam.C) it is a combination of the transient incompressible laminar solver icoFoam and the stress analysis solver solidDisplacementFoam, with coupling based on an idea from conjugateFoam. First the solver will be described and compared to the pre-existing solvers icoFoam and solidDisplacementFoam, the new added code, i.e. the coupling, will be emphasized. After that a simple case, taken from the Wiki, will be explained and run.

1 Solver

The solver thus combines two pre-existing solvers of openfoam for a new specific purpose, namely fluid-structure interaction. This is one of the basic ideas of openfoam, i.e. to taylor a new solver, from the pre-existing ones, that fulfills the specific purpose of the user. Combining different solvers may be more or less difficult depending on which solvers that should be combined and what the new solver should do (and off course depending on the previous C++ and OpenFOAM knowledge). In this case the tricky part is the coupling between the solid part of the mesh and the fluid part of the mesh.

As described in the introductory text the fluid part of the solver is based on icoFoam which is a solver for transient laminar flow, the solid part is based on solidDisplacementFoam. Example cases of both these solvers are found in the Users Guide under the Tutorial chapter. It will therefore be assumed that the cases of those two solvers are known before, also that the user have some knowledge about the content of the source files for those two cases.

As mentioned above the coupling is based on an idea from conjugateFoam which solves a diffusion equation in two different regions, with coupling between them at one boundary. The major contribution to icoStructFoam from conjugateFoam, as far as the author sees it, is how to treat the two regions

in the source files and in the case files. Technically two meshes are produced which are overlapping at certain patches, as the user wishes. These are then put in different polyMesh directories, one for region 1 and one for region 2, respectively. Further case details will be described in the next section.

1.1 The Code

The solver directory contains several additional files as compared to the directories of icoFoam and solidDisplacementFoam, which will be described or mentioned as they appear while going through the main code in **icoStrucFoam.C**. The code of this file is shown in appendix A.

The file starts with several include statements as all the other OpenFOAM solvers. The first two statements are also found in the old solvers, most other include files are new and do the following (with reservation for the misunderstanings of the author).

- **fixedValueFvPatchFields.H** type defining the fixedValue patches to scalar, tensor etc (included in fvCFD.H).
- **tractionDisplacement/tractionDisplacementFvPatchVectorField.H** declaration of the tractionDisplacement boundary condition.
- **fvMesh.H** Mesh data declarations (included in fvCFD.H).
- **primitivePatchInterpolation.H** declaration of interpolation functions from points to faces and vice verca.
- **motionSolver.H** declaration of mesh-motion solver.

Then comes the *main* part of the program, which is initiated by including several files into the code as follows. The first two files are found in both old solvers and are therefore omitted here.

- **createMeshes.H** creates meshes for fluid region and solid region respectively (same thing as createMesh.H).
- **readMechanicalProperties.H** reading the mechanicaleProperties dictionary and computing some parameters (same as for sDF).
- **readStressedFoamControls.H** reading "stressAnalysis" subdictionary of fvSolution in region 2 (solid region).
- **readThermalProperties.H** reading thermal properties and computing parameters (same as for sDF).
- **createIcoFields.H** create fields for fluid domain (same as createFields.H in iF).

- **createStructureFields.H** create fields for solid domain (same as createFields.H in sDF).
- **readCoupling.H** read "couplingParameters" file in constant/region2/. Identifying the coupled patches and checking that the meshes are next to each other. Set original face coordinates, *oldPoints*, for common fluid patch.
- **createMeshMotion.H** initializes mesh motion solver and if it is displacement based or not. Initial displacement is set.
- **initContinuityErrs.H** found in iF.

After those initilization rows we are going straight into the mesh motion and coupling part of the code which is new for this solver. The time counter is first started, then another for loop is found which check with the value specified in the previously mentioned "couplingParameter" file whether mesh motion should be started yet or not. If it sould be started the previously read or set *displace* value, of the solid side, is given to the variable *dispVals*. *maxDist* and *maxAway* are initialized to $-1 \cdot 10^{10}$. Then if the motion solver is displacement based we are going into the next if statement. There the boundary patches of the *cellDisplacement* file, of the fluid side, are given the reference variable *meshDisplacement*, whereafter it is checked if the fluid side boundary towards the solid boundary is movable.

The displacement of the fluid side of the common patch is given the reference name *mDisp*. A motion relaxation *factor* is then introduced, taken from the "couplingParameter" file again.

Then a loop is started that loops through all elements of the fluid side of the boundary patch towards the solid side. In the loop the solid face label is first exchanged to the corresponding fluid face label. The coordinates of face centers and initial face centers (*oldPoints*) of this patch are listed, whereafter the displacement of the corresponding solid patch is listed. The total displacement is then computed based on the solid patch displacement, the relaxation factor and the new and inital face center coordinates, whereafter the reference variable *mDisp* is given this value. In mathematical terms the displacement is computed as

$$D_{tot} = Df + (P - P_{old})(1 - f); \tag{1}$$

where D_{tot} is the total displacement, D is the solid side displacement, f is the relaxation factor, P is the new coordinate of the face center and P_{old} is the initial coordinate of the face center.

After the displacement is set the total maximum displacement and the current maximum displacement are computed.

Then follows an else statement, whereafter the same procedure is performed, but instead based on a motion based mesh motion solver. As far as

I understand only one of them is used and therefore I only describe one of them. The codes are very similar anyway.

After the else statement the new points are computed based on a laplacian mesh motion solver and then follows the regular icoFoam solver and the solidDisplacement solver. The icoFoam computations are made on mesh 1 and solidDisplacementFoam on mesh 2.

IcoFoam solves the Navier-Stoke's equations and couples the momentum field to the continuity equation and pressure by the PISO formulation. The Navier-Stokes equations and the conitunity equation for incompressible flow are given by (2).

$$\begin{gathered} \frac{\partial u_i}{\partial t} + \frac{\partial u_i u_j}{\partial x_j} - \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} \\ \frac{\partial u_i}{\partial x_i} = 0 \end{gathered} \tag{2}$$

solidDisplacementFoam solves the following form of Navier's equations for the solid region.

$$\frac{\partial^2 D_i}{\partial t^2} = \frac{\partial}{\partial x_j}\left(2\mu + \lambda\right)\frac{\partial D_i}{\partial x_j} + \frac{\partial \sigma_{ij}}{\partial x_j} \tag{3}$$

where $\mu = \frac{E}{2(1+\nu)}$ and $\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}$.

After the solidDisplacementFoam solver has been run the pressure from the fluid solver is written to the solid side of the boundary, before the next time step is initiated.

The solver may be concluded as doing the following in every time step:

1. The fluid mesh is deformed according to the displacement of the solid boundary.
2. The flow is solved for in the deformed fluid region.
3. The deformation of the solid region is solved for, based on the pressure distribution at the solid/fluid interface from previous timestep.
4. The pressure is transferred from the fluid to the solid region.

2 Case

This case is also downloaded from sourceforge and consists of a 2D deformable channel. The domain is shown in figure 1, where the upper green part is the solid region and the lower black part is the fluid region. The lower side of the fluid domain and the upper side of the solid domain are defined as stationary walls. The flow through the fluid domain is steady from the left to the right.

This part of the tutorial will show one example how to implement the previously described solver, including treatment of multiple meshes with OpenFOAM. Again, it is presumed that the user is allready familiar with the

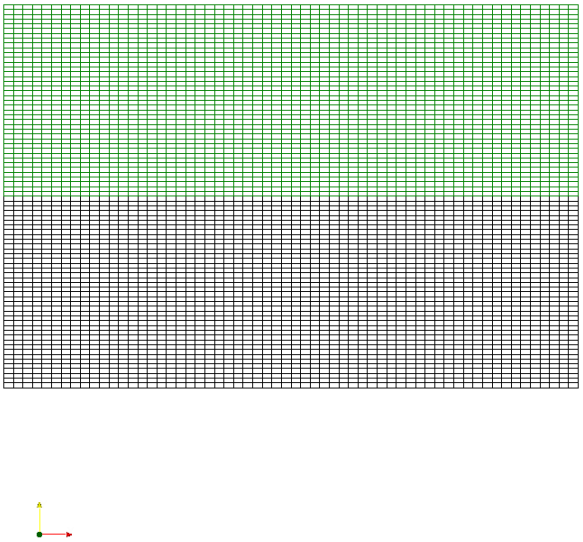

Figure 1: The computational mesh, where the upper part is for the solid part and the lower is for the fluid part.

two tutorials covering the icoFoam and the solidDisplacementFoam solvers in the Users Guide.

2.1 Pre-processing

The solver and the case files are downloaded from sorceforge using the following statement: *svn checkout https://openfoam-extend.svn.sourceforge.net/svnroot/openfoam-extend/trunk/Breeder/solvers/other/IcoStructFoam/*. This version of the solver is made for OpenFOAM-1.4.1, there is another older version for OpenFoam-1.3 on the OpenFOAM Wiki, which also contains some information cencerning multiple meshes and the case described here.

After the code and case files have been downloaded the code needs to be compiled in order for it to run. This is done by going to the icoStructFoam solver directory, where the Make directory is, and typing *wmake*.

2.1.1 Mesh generation

After the solver and the Case called *icoStructFoamTest* have been downloaded the meshes should be generated. However, as this case contains multiple meshes the standard blockMesh utility can not be used as usual. The reason is that a polyMesh directory is not found under the constant directory, but the the two polyMesh directories are found under *constant/region1/*

and *constant/region2/*, for the fluid and solid regions respectively. In order then to trick the **blockMesh** utility, soft links are created, whereafter the **blockMesh** utility is run. The code reads:

ln -s region1/polyMesh/ icoStructFoamTest/constant/polyMesh
blockMesh . icoStructFoamTest
rm icoStructFoamTest/constant/polyMesh
ln -s region2/polyMesh/ icoStructFoamTest/constant/polyMesh
blockMesh . icoStructFoamTest
rm icoStructFoamTest/constant/polyMesh

The *blockMeshDict* file describes the vertices, blocks, edges and patches, as for the previous tutorials. Also, as for the previous tutorials this is a 2D case, so the front and back are defined as empty. The other patches are called inlet, outlet, top and bottom, for both mesh parts.

2.1.2 Boundary & initial conditions

The initial and boundary conditions are located under *0/region1/* and *0/region2/* for the two regions. Region 1 is the fluid region and region 2 the solid. For the fluid region the pressure and velocity field needs to be specified, but also the initial displacement or motion of the fluid region needs to be specified. The pressure is set in file *p* to zeroGradient for bottom, top and inlet, to empty for frontandback, and to fixedValue Uniform 0 for the outlet. InternalField is set to uniform 0. The velocity is set in file *U* to fixedValue 0 for the bottom and top (no slip condition), zeroGradient for outlet, fixedValue uniform (0.005 0 0) at the inlet, and empty for frontandback. Internal field is again set to uniform (0 0 0).

Now, the motion solver will either be motion driven or diplacement driven, and as the code was described before it will here be displacement driven. The *cellMotionU* and *pointMotionU* files are therefore not needed and will not be described any further. The *cellDisplacement* file describes the displacement of the cell centers and *pointDisplacement* describes the motion of the vertices. In both files the inlet and outlet are defined as zeroGradient, the bottom is defined as fixedValue uniform (0 0 0). The top is defined as zeroGradient for the points and fixedValue uniform (0 0 0) for the cell centers. The frontandback are again empty.

For region 2 the displacement is set in file *D*. The internal field is set to uniform (0 0 0), the other patches are defined as

bottom
{
type tractionDisplacement;
traction uniform (0 0 0);
pressure uniform 0;
value uniform (0 0 0);
}
top
{
type fixedValue;
value uniform (0 0 0);
}
frontAndBack
{
type empty;
}
inlet
{
// type tractionDisplacement;
// traction uniform (0 0 0);
// pressure uniform 0;
type fixedValue;
value uniform (0 0 0);
}
outlet
{
type tractionDisplacement;
traction uniform (0 0 0);
pressure uniform 0;
value uniform (0 0 0);
}

Note that the top patch is changed from the original setting, otherwise the solver will crash. The other file, *T*, is where the temperature details are specified, however, here the case is isothermal at 300 K, and therefore the temperature field is not very interesting. Still the temperature has to be set to 300 K in the *T* file.

2.1.3 Physical properties

The physical properties for the fluid region is only for the kinematic viscosity which is set to $\nu = 10^{-4}\ m^2/s$. For the solid region the density, $\rho = 7854\ kg/m^3$, Poisson ratio, $\nu = 0.3$ and the bulk modulus, $E = 10^{-2}\ kg/(ms^2)$

are set in the *mechanicalProperties* file. Here it is also specified that the plane stress assumption should be used. In the *thermalProperties* file thermal properties are specified, but those are not used here and will not be described. Also the *transportproperties* in the *region2* directory do not seem to be used.

2.1.4 Control

In the *controlDict* file the time step deltaT is set to 0.001, end time to 1 and the write control keyword to runTime and the writeInterval to 0.01.

2.1.5 Descretization & linear solver settings

As for the other tutorials these settings are not discussed in detail, but in addition to previous solvers the files here contain details about the mesh motion variables, and may be viewed in the corresponding directories, respectively.

2.1.6 Coupling settings

Under *constant/region1/* the file *dynamicMeshDict* is found. This file specifies that we have 2D-motion and that the mesh motion should be solved using a Laplacian solver with uniform diffusivity. The corresponding directory for region 2 contains the file *couplingParameters*, which specifies which patches that should be coupled, at what time point mesh motion should be taken into account and if a relaxation parameter should be specified for the mesh motion.

2.2 Running an application

The solver with the set up case is then run in the same way as before using *<solver> <path> <case>*, or with some other additional commands. If, for example, the case is called *isf* and we are standing in the parent directory of the case directory, the command would be *icoStructFoam . isf*.

2.3 Post-processing

The multi-mesh results can not either be viewed with paraFoam as the previous cases. Therefore OpenFOAM needs to be tricked again in order to make the results viewable with paraview. Use the following commands:

foamToVTK . icoStructFoamTest -mesh region1

foamToVTK . icoStructFoamTest -mesh region2

(Before typing the command for region2, the boundary condition along the top wall for T needs to be changed from *symmetryPlane* to *zeroGradient*, otherwise the utility will not work.)

Now, VTK-files have been written into the VTK directory. The data are then viewed by launching *paraview* and open the wanted files. Figure 2 show

the displacement by color of the solid mesh. In order to deform (warp) the mesh it seems like point data are needed, which was not saved. Therefore only cell values can be viewed, and the mesh in its undeformed shape.

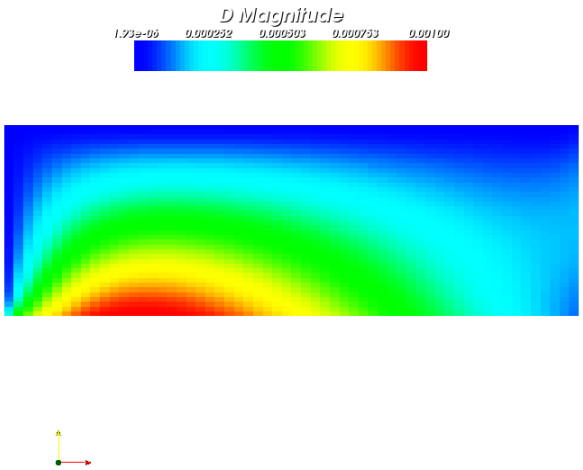

Figure 2: Displacement of the solid mesh shown by the color field.

On the other hand, for the fluid mesh the displacement can be viewed and is shown in figure 3, where again the color shows the displacement.

Comparing the pressure field, figure 4, to the magnitude of the stress tensor, figure 5, of the solid domain confirms the relation between the two.

The last figure, figure 6, shows the velocity magnitude in the fluid domain.

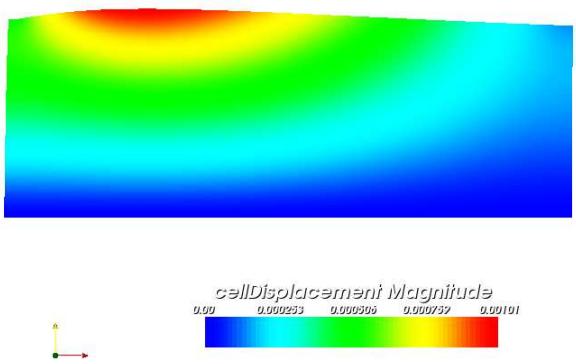

Figure 3: Displacement of the solid mesh shown by the color field.

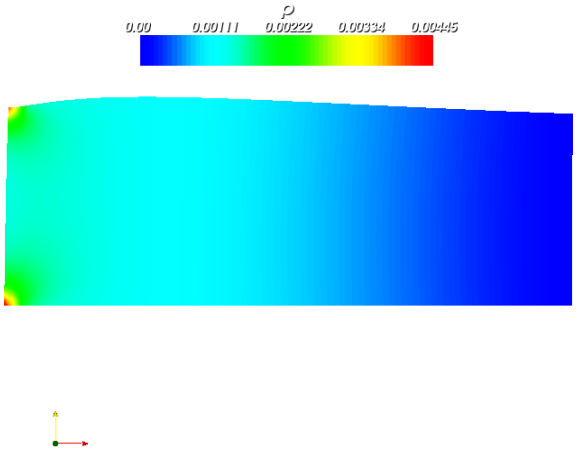

Figure 4: Pressure field of the fluid domain.

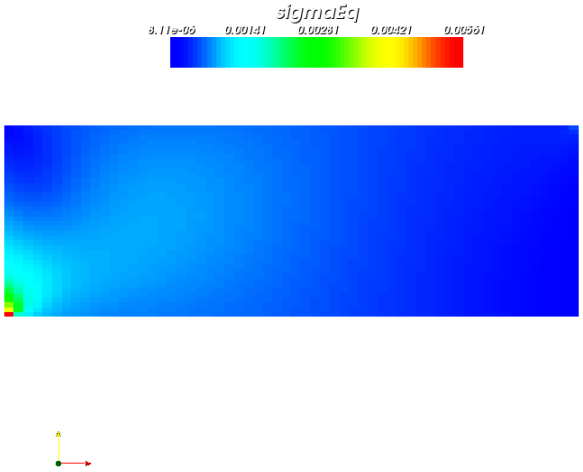

Figure 5: Magnitude of the stress tensor of the solid domain.

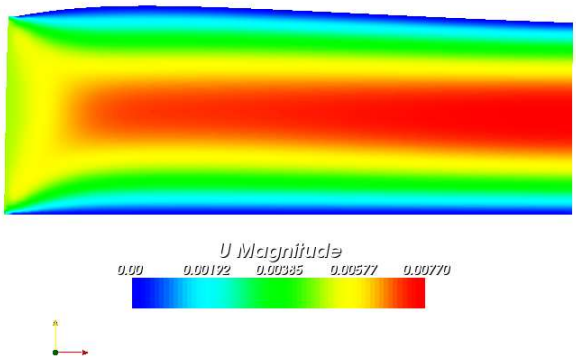

Figure 6: Velocity magnitude of the fluid domain.

A Main Code of icoStructFoam

```
/*---------------------------------------------------------------------------*\
 ##   ####  ######     |
 ## ##    ##         | Copyright: ICE Stroemungsfoschungs GmbH
 ## ##    ####       |
 ## ##    ##         | http://www.ice-sf.at
 ##   ####  ######     |
-------------------------------------------------------------------------------
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 1991-2005 OpenCFD Ltd.
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is based on OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Application
    icoStructFoam

Description
    Transient solver for incompressible, laminar flow of Newtonian fluids
    coupled with structural mechanics

    Based on icoFoam and solidDisplacementFoam
    Coupling after an idea by conjugateFoam

 ICE Revision: $Id:
/local/openfoam/branches/WikiVersions/FluidStructCoupling/icoStructFoam/icoStructFoam.C 1906 2007-08-
28T16:16:19.392553Z bgschaid  $
\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "Switch.H"
#include "fixedValueFvPatchFields.H"
#include "tractionDisplacement/tractionDisplacementFvPatchVectorField.H"
#include "fvMesh.H"
#include "PrimitivePatchInterpolation.H"
#include "motionSolver.H"

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

int main(int argc, char *argv[])
{

#   include "setRootCase.H"

#   include "createTime.H"
```

```
#   include "createMeshes.H"

#   include "readMechanicalProperties.H"
#   include "readStressedFoamControls.H"
#   include "readThermalProperties.H"

#   include "createIcoFields.H"
#   include "createStructureFields.H"

#   include "readCoupling.H"

#   include "createMeshMotion.H"

#   include "initContinuityErrs.H"

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

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

    for (runTime++; !runTime.end(); runTime++)
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

          if(runTime.value()>=startMeshMotion.value()) {
            Info << "\nMoving mesh\n" << endl;

          // Make the fluxes absolute
          //          fvc::makeAbsolute(phi, U);

          //           dispVals=inter.faceToPointInterpolate(displace);
          dispVals=displace;

            scalar maxDist=-1e10;
            scalar maxAway=-1e10;

          if(displacementMotionSolver) {
             volVectorField::GeometricBoundaryField &meshDisplacement=

const_cast<volVectorField&>(mesh1.objectRegistry::lookupObject<volVectorField>("cellDisplacement")).bou
ndaryField();

             if(typeid(meshDisplacement[fluidSideI])!=typeid(fixedValueFvPatchField<vector>)) {
                FatalErrorIn("Coupled Solver ") << "Fluid side not movable" << exit(FatalError);
             }

             vectorField &mDisp=refCast<vectorField>(meshDisplacement[fluidSideI]);

             scalar factor=1/motionRelaxation.value();

             forAll(fluidMesh,fluidI) {
                label solidI=exchange[fluidI];

                //      vector here=mesh1.points()[fluidPoints[fluidI]];
                vector here=fluidMesh.faceCentres()[fluidI];
                vector old =oldPoints[fluidI];
                vector disp=dispVals[solidI];
                vector neu=disp*factor+(here-old)*(1-factor);
                vector move=neu;

                mDisp[fluidI]=move;
```

displacement based mesh motion

```
            if(mag(here-old)>maxDist) {
                maxDist=mag(here-old);
            }
            if(mag(neu-here)>maxAway) {
                maxAway=mag(here-neu);
            }
        }

    } else {
        volVectorField::GeometricBoundaryField &motionU=

const_cast<volVectorField&>(mesh1.objectRegistry::lookupObject<volVectorField>("cellMotionU")).boundary
Field();

        if(typeid(motionU[fluidSideI])!=typeid(fixedValueFvPatchField<vector>)) {
            FatalErrorIn("Coupled Solver ") << "Fluid side not movable" << exit(FatalError);
        }

        vectorField &patchU=refCast<vectorField>(motionU[fluidSideI]);

        //      tetPointVectorField neu=inter.faceToPointInterpolate(patchU);
        scalar factor=1/(runTime.deltaT().value()*motionRelaxation.value());
        //          const labelList& fluidPoints=fluidPatch.meshPoints();

        //      forAll(fluidPoints,fluidI) {
        forAll(fluidMesh,fluidI) {
            label solidI=exchange[fluidI];

            //      vector here=mesh1.points()[fluidPoints[fluidI]];
            vector here=fluidMesh.faceCentres()[fluidI];
            vector old =oldPoints[fluidI];
            vector disp=dispVals[solidI];
            vector neu=disp+old;
            vector move=factor*(neu-here);

            patchU[fluidI]=move;

            if(mag(here-old)>maxDist) {
                maxDist=mag(here-old);
            }
            if(mag(neu-here)>maxAway) {
                maxAway=mag(here-neu);
            }
        }
    }
      mesh1.movePoints(motionPtr->newPoints());
    //      U.correctBoundaryConditions();

      Info << "\nBiggest movement: " << maxDist << " Bigges divergence " << maxAway <<  endl;

    // Make the fluxes relative
    //      fvc::makeRelative(phi, U);
     }

     Info << "Solving flow in mesh1\n" << endl;
     {
#    include "readPISOControls.H"
#    include "CourantNo.H"

      fvVectorMatrix UEqn
```

displacement based mesh motion

motion based mesh motion

icoFoam

```
        (
         fvm::ddt(U)
         + fvm::div(phi, U)
         - fvm::laplacian(nu, U)
         );

        solve(UEqn == -fvc::grad(p));

        // --- PISO loop

        for (int corr=0; corr<nCorr; corr++)
         {
           volScalarField rUA = 1.0/UEqn.A();

           U = rUA*UEqn.H();
           phi = (fvc::interpolate(U) & mesh1.Sf())
           + fvc::ddtPhiCorr(rUA, U, phi);

           adjustPhi(phi, U, p);

           for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
               {
                 fvScalarMatrix pEqn
                  (
                   fvm::laplacian(rUA, p) == fvc::div(phi)
                   );

                 pEqn.setReference(pRefCell, pRefValue);
                 pEqn.solve();

                 if (nonOrth == nNonOrthCorr)
                  {
                    phi -= pEqn.flux();
                  }
               }

#          include "continuityErrs.H"

           U -= rUA*fvc::grad(p);
           U.correctBoundaryConditions();
         }

       }
```

icoFoam

```
       Info << "\nSolving structure in mesh2\n" << endl;

       {
#       include "readStressedFoamControls.H"
         int iCorr = 0;
         scalar initialResidual = 0;

         do
          {
           if (thermalStress)
               {
                volScalarField& T = Tptr();
                solve
                   (
                       fvm::ddt(T) == fvm::laplacian(DT, T)
                   );
               }
```

solidDisplacementFoam

```
        {
            fvVectorMatrix DEqn
            (
                fvm::d2dt2(D)
                ==
                fvm::laplacian(2*mu + lambda, D, "laplacian(DD,D)")
                + divSigmaExp
            );

            if (thermalStress)
            {
                const volScalarField& T = Tptr();
                DEqn += threeKalpha*fvc::grad(T);
            }

            //UEqn.setComponentReference(1, 0, vector::X, 0);
            //UEqn.setComponentReference(1, 0, vector::Z, 0);

            initialResidual = DEqn.solve().initialResidual();

            if (!compactNormalStress)
            {
                divSigmaExp = fvc::div(DEqn.flux());
            }
        }
        {
            volTensorField gradD = fvc::grad(D);
            sigmaD = mu*twoSymm(gradD) + (lambda*I)*tr(gradD);

            if (compactNormalStress)
            {
                divSigmaExp =
                    fvc::div(sigmaD - (2*mu + lambda)*gradD, "div(sigmaD)");
            }
            else
            {
                divSigmaExp += fvc::div(sigmaD);
            }
        }

        } while (initialResidual > convergenceTolerance && ++iCorr < nCorr);

#   include "calculateStress.H"

        Info << "\nMaximum Displacement: " << max(mag(D)).value() << endl;

    }

    Info << "\nCoupling the solutions\n" << endl;

    scalarField & fluidP = p.boundaryField()[fluidSideI];

    scalarField &solidP = displace.pressure();

    forAll(fluidP,fl) {
      solidP[exchange[fl]]=-fluidP[fl];
    }

#   include "write.H"
```

solidDisplacementFoam

coupling pressure