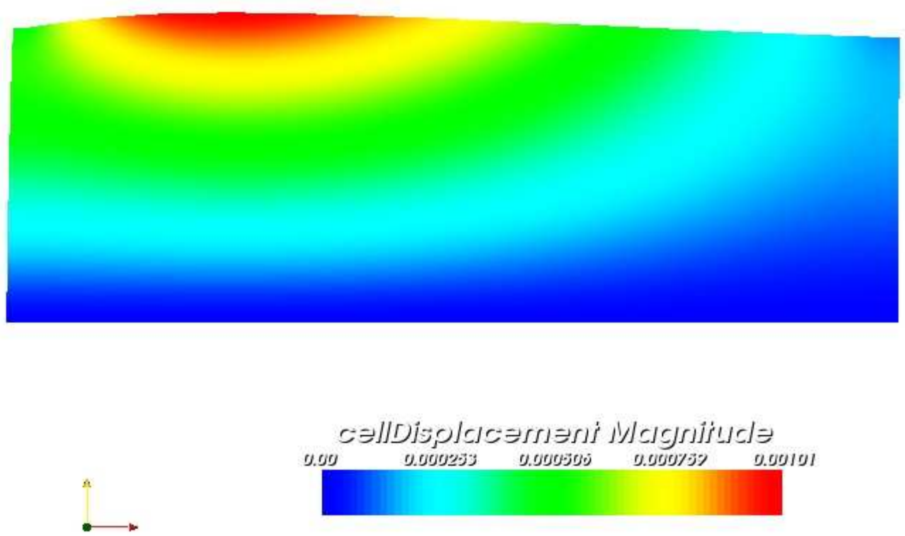
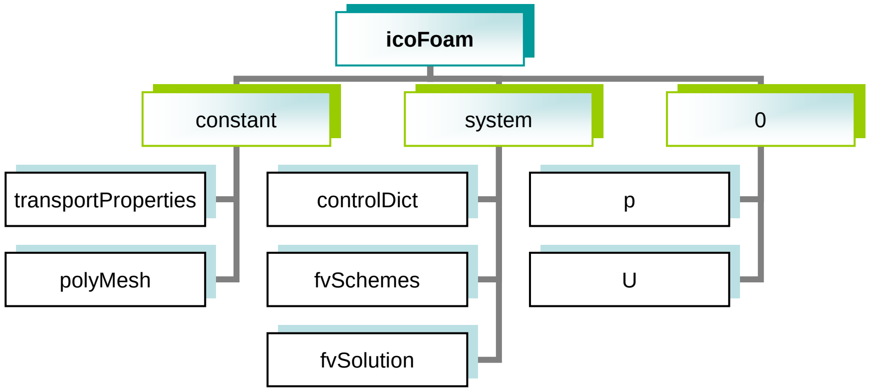
icoStructFoam

a fluid-structure interaction solver

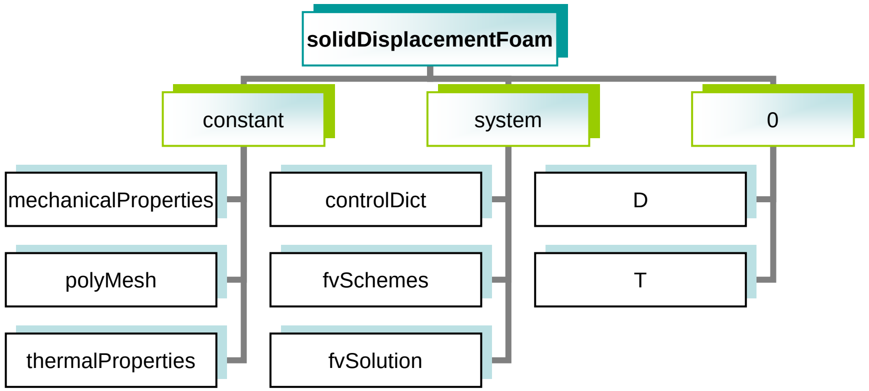
Philip Evegren



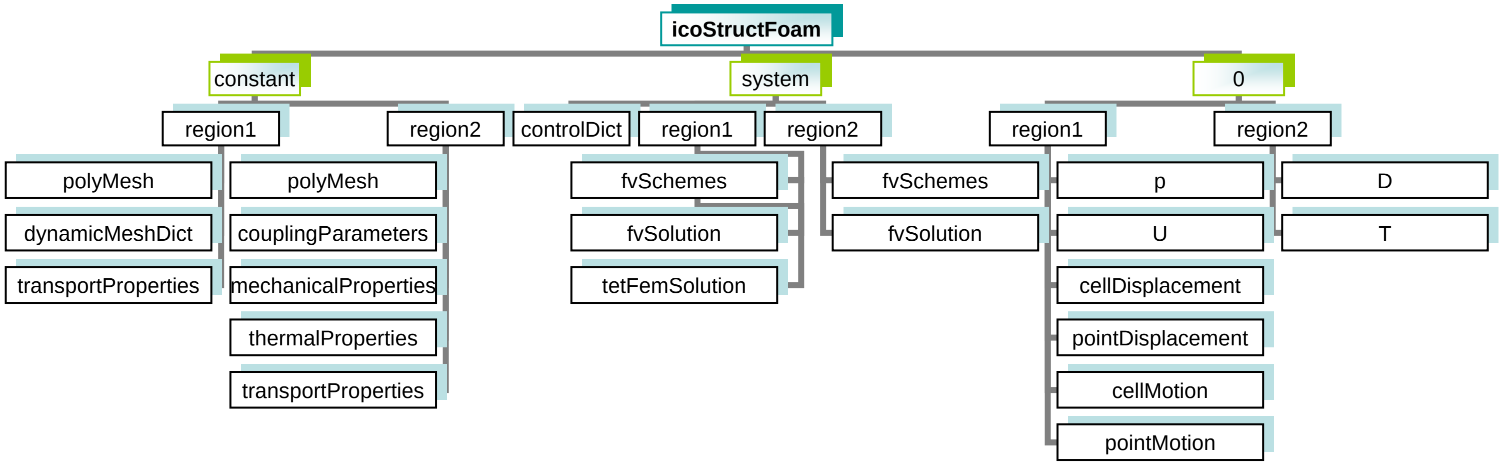
Cases



- icoFoam
  - constant
    - transportProperties
    - polyMesh
  - system
    - controlDict
    - fvSchemes
    - fvSolution
  - 0
    - p
    - U

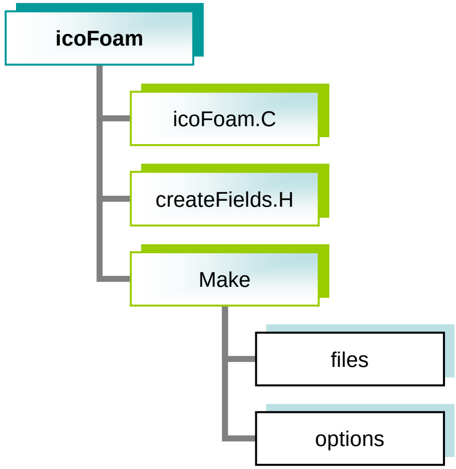


- solidDisplacementFoam
  - constant
    - mechanicalProperties
    - polyMesh
    - thermalProperties
  - system
    - controlDict
    - fvSchemes
    - fvSolution
  - 0
    - D
    - T

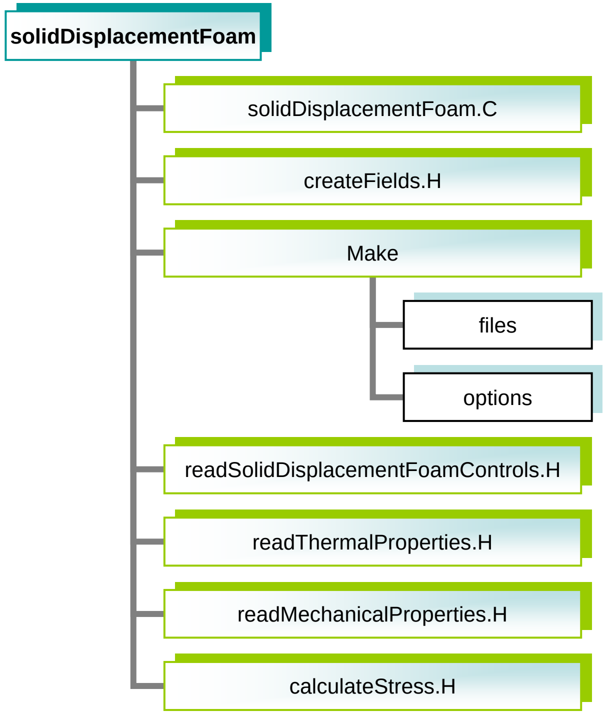


- icoStructFoam
  - constant
    - region1
      - polyMesh
      - dynamicMeshDict
      - transportProperties
    - region2
      - polyMesh
      - couplingParameters
      - mechanicalProperties
      - thermalProperties
      - transportProperties
  - system
    - controlDict
    - region1
      - fvSchemes
      - fvSolution
      - tetFemSolution
    - region2
      - fvSchemes
      - fvSolution
  - 0
    - region1
      - p
      - U
      - cellDisplacement
      - pointDisplacement
      - cellMotion
      - pointMotion
    - region2
      - D
      - T

Solvers

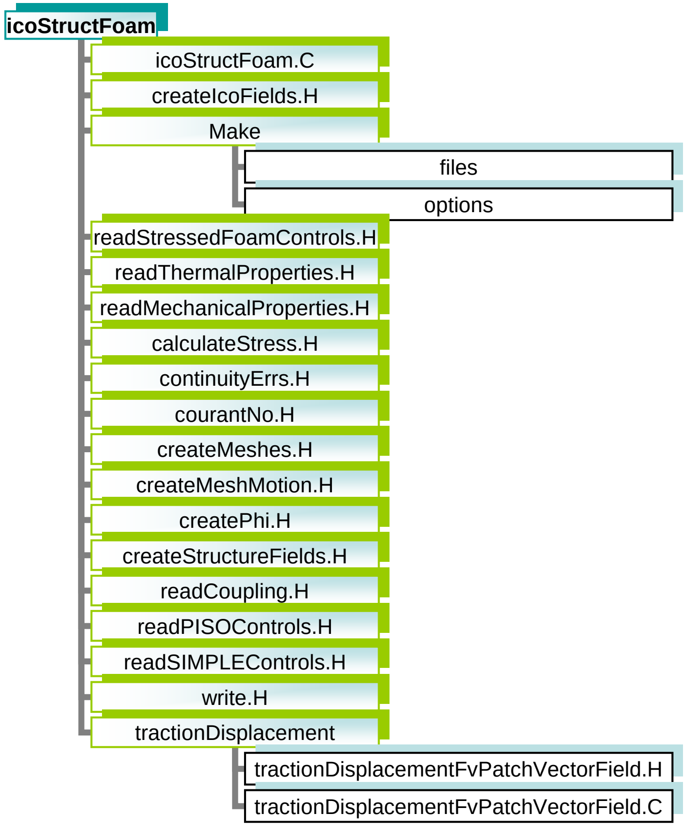


- **icoFoam**
  - icoFoam.C
  - createFields.H
  - Make
    - files
    - options



- **solidDisplacementFoam**
  - solidDisplacementFoam.C
  - createFields.H
  - Make
    - files
    - options
  - readSolidDisplacementFoamControls.H
  - readThermalProperties.H
  - readMechanicalProperties.H
  - calculateStress.H

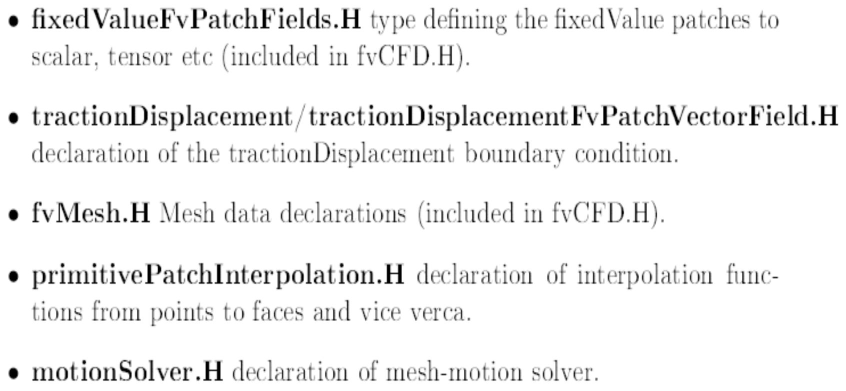
Solvers



icoStructFoam
icoStructFoam.C
createIcoFields.H
Make
files
options
readStressedFoamControls.H
readThermalProperties.H
readMechanicalProperties.H
calculateStress.H
continuityErrs.H
courantNo.H
createMeshes.H
createMeshMotion.H
createPhi.H
createStructureFields.H
readCoupling.H
readPISOControls.H
readSIMPLEControls.H
write.H
tractionDisplacement
tractionDisplacementFvPatchVectorField.H
tractionDisplacementFvPatchVectorField.C

Source code

```
/*---------------------------------------------------------------------------*\
 ##   ####  ######     |
 ##  ##     ##         | 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"
```

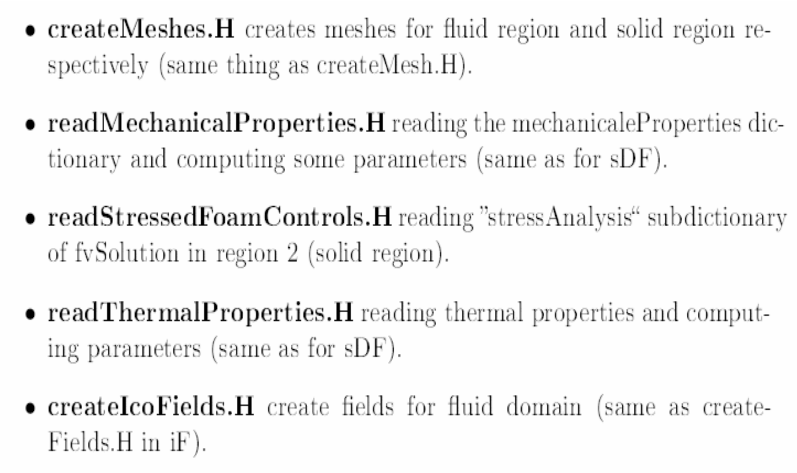


- **fixedValueFvPatchFields.H** type defining the fixedValue patches to scalar, tensor etc (included in fvCFD.H).
- **tractionDisplacement/tractionDisplacementFvPatchVectorField.H** declaration of the tractionDisplacement boundary condition.
- **fvMesh.H** Mesh data declarations (included in fvCFD.H).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

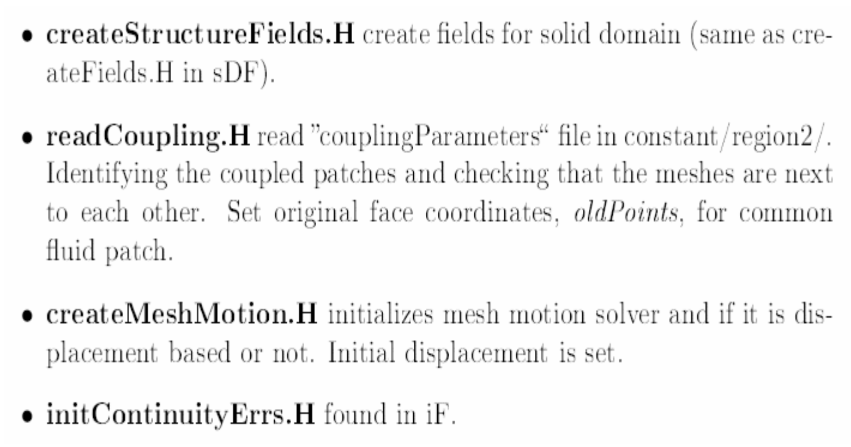
Source code

```
#   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



- **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.

Source code

displacement based
mesh motion

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

```
                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);
         }
```

motion based
mesh motion

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

           fvVectorMatrix UEqn
```

icoFoam

Source code

```
      (
       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();
      }

  }

  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)
              );
            }
```

icoFoam

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

$$\frac{\partial u_i}{\partial x_i} = 0$$

solidDisplacementFoam

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

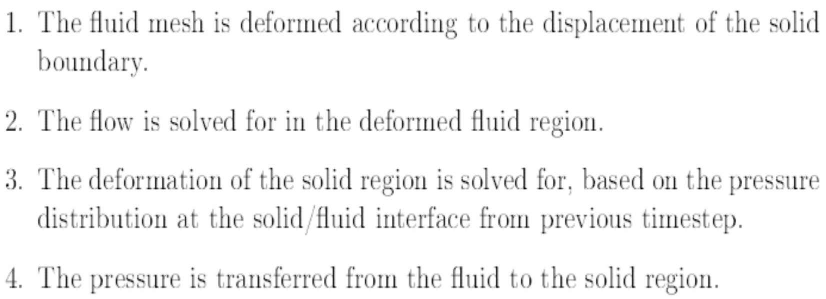
Source code

```
            {
                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;
```

solidDisplacementFoam

```
          }

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

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

          scalarField &solidP = displace.pressure();

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

#       include "write.H"
```

coupling pressure

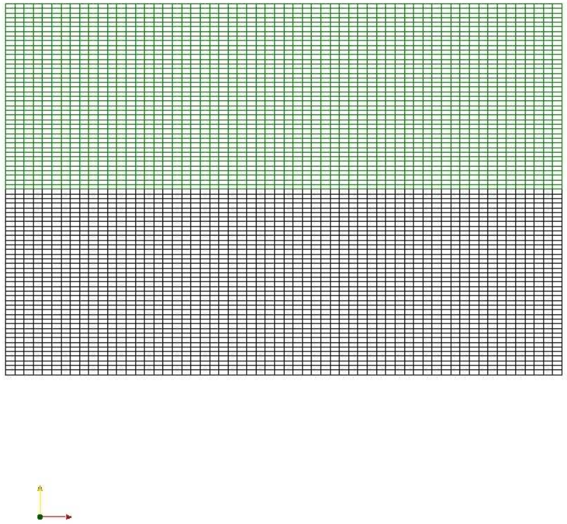


1. The fluid mesh is deformed according to the displacement of the solid boundary.
2. The flow is solved for in the deformed fluid region.
3. The deformation of the solid region is solved for, based on the pressure distribution at the solid/fluid interface from previous timestep.
4. The pressure is transferred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

Running a case

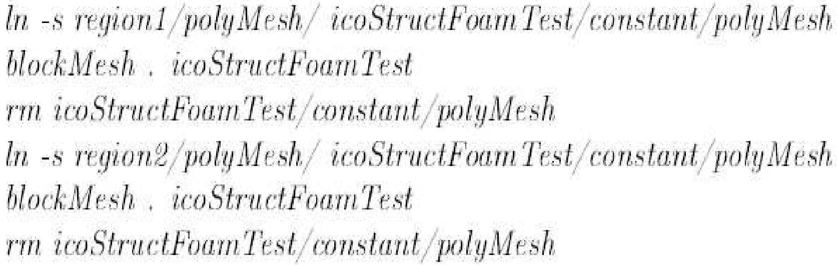
- **Download the solver using:**

svn checkout https://openfoam-extend.svn.sourceforge.net/svnroot/openfoam-extend/trunk/Breeder/solvers/other/IcoStructFoam/



- **Compile the solver using wmake**

- **Run the blockMesh utility**



```
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
```

Running a case

- **Boundary & initial conditions**

region1

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

internalField   uniform (0 0 0);

boundaryField
{

    bottom
    {
        type            fixedValue;
        value            uniform (0 0 0);
    }

    top
    {
        type            fixedValue;
        value            uniform (0 0 0);
    }

    frontAndBack
    {
        type            empty;
    }

    inlet
    {
        type            fixedValue;
        value            uniform (0.005 0 0);
    }

    outlet
    {
                          type
zeroGradient;
    }
}
```

U

p

```
dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{

    bottom
    {
        type            zeroGradient;
    }

    top
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            empty;
    }

    inlet
    {
        type            zeroGradient;
    }

    outlet
    {
        type            fixedValue;
                       value
                       uniform 0;
    }
}
```

Running a case

- **Boundary & initial conditions region1**

cD

```
dimensions      [0 1 0 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{

    bottom
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    top
    {
//        type            zeroGradient;
        type            fixedValue;
        value           uniform (0 0 0);
    }

    frontAndBack
    {
        type            empty;
    }

    inlet
    {
        type            zeroGradient;
    }

    outlet
    {
                        type            zeroGradient;
    }
}
```

pD

```
dimensions      [0 1 0 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{

    bottom
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    top
    {
        type            zeroGradient;
//        type            fixedValue;
//        value           uniform (0 0 0);
    }

    frontAndBack
    {
        type            empty;
    }

    inlet
    {
        type            zeroGradient;
    }

    outlet
    {
                        type            zeroGradient;
    }
}
```

Running a case

- **Boundary & initial conditions**

region2

```
dimensions      [0 1 0 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{

    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);
    }
}
```

← **D**

couplingParameters

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

fluidSide top;
solidSide bottom;
startMeshMotion time [0 0 1 0 0 0 0] 0.1;
motionRelaxation iTime  [0 0 -1 0 0 0 0] 5;


// *************************************************************************** //
```

Running a case

- **Other properties**

dynamicMeshDict

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

twoDMotion      yes;

//solver          laplaceTetDecomposition;

//diffusion       quadratic patchEnhanced;

//frozenDiffusion off;

//distancePatches
//(
//);
dynamicFvMesh dynamicMotionSolverFvMesh;

motionSolverLibs ("libfvMotionSolvers.so");
// motionSolverLibs ("libfvMotionSolvers.dylib");

solver displacementLaplacian;

diffusivity  uniform;


// ************************************************************************* //
```

controlDict

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

applicationClass laplacianFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         1;

deltaT          0.001;

writeControl    runTime;

writeInterval   0.01;

cycleWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable yes;


//
*******************************************************************
****** //
```

Running a case

- **Run by typing:**

```
<solver> <path> <case>
```

- **Post-processing**

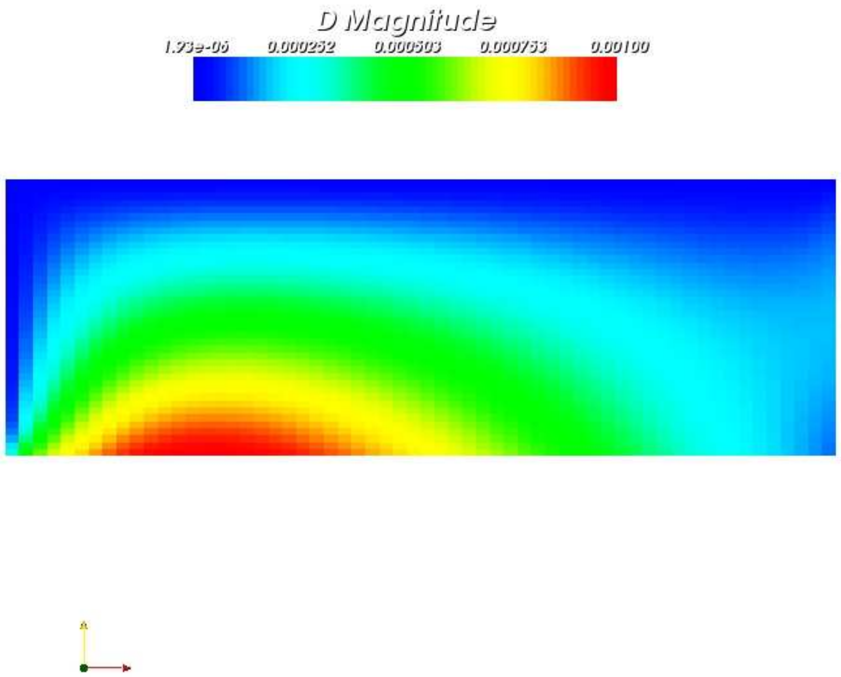
foamToVTK . icoStructFoamTest -mesh region1

foamToVTK . icoStructFoamTest -mesh region2

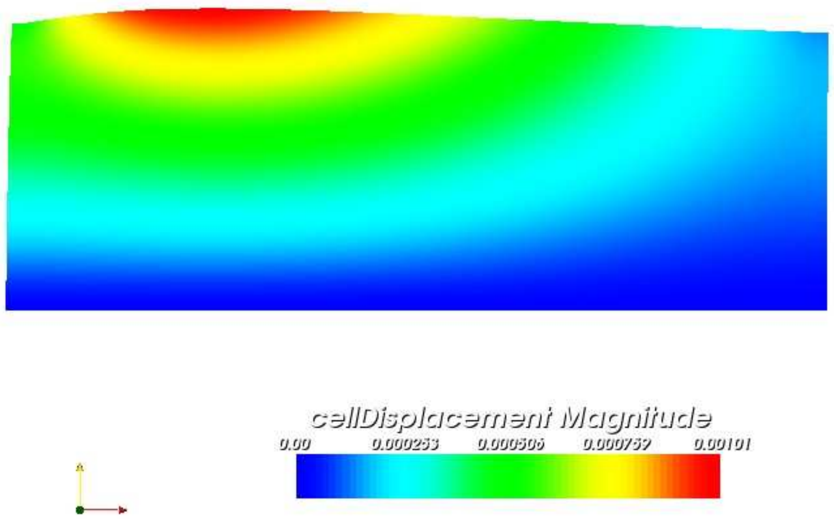
Launch paraview

Running a case

Displacement of solid region

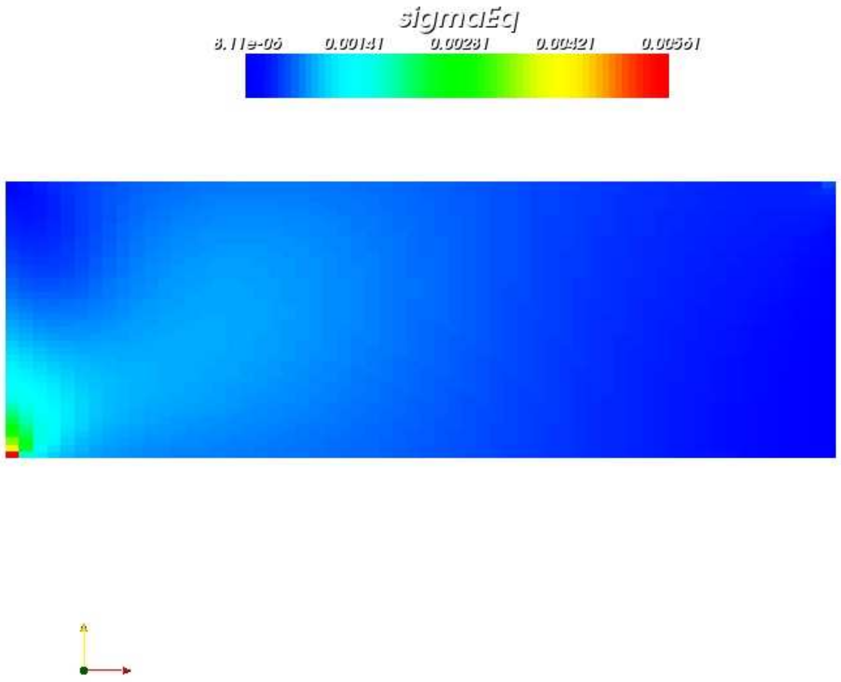


Displacement of fluid region

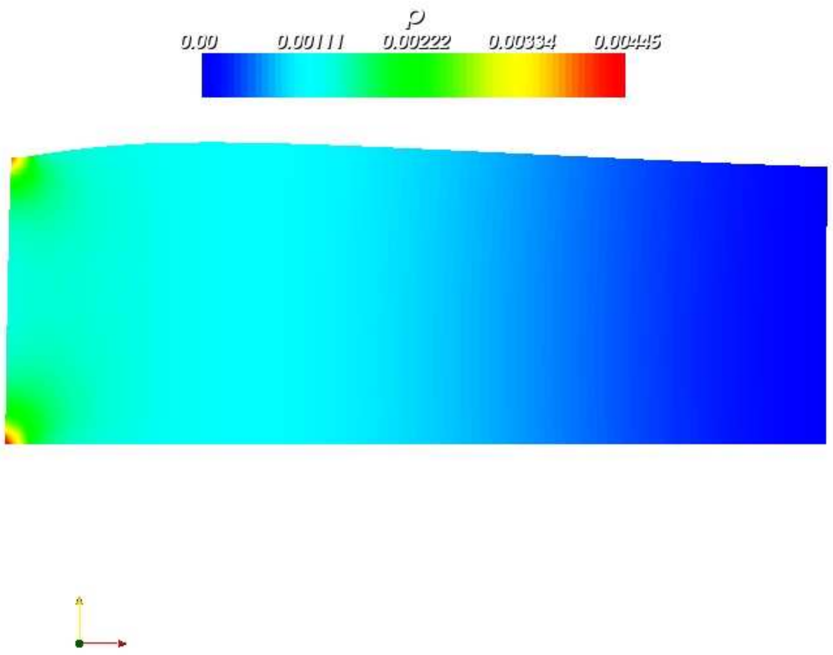


Running a case

Stress field of solid region

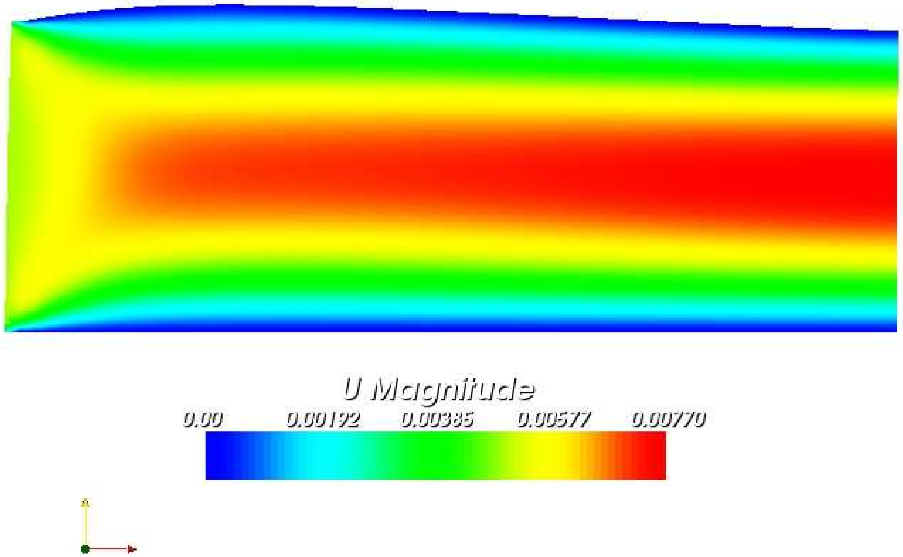


Pressure field of fluid region



Running a case

Velocity field of fluid region


U Magnitude