Guide to rotationDeformationFvMesh solver for solids4Foam

Note:

This document describes the general workflow of the FSI solver, based on [solids4Foam](https://bitbucket.org/philip_cardiff/solids4foam-release/src/master/) tool. There are two main styles of writing:

One for general descriptions.

The other for codes, file names, locations.

Spacings, comments, and other details, which enhances code readability, were removed to reduce spent area. Irrelevant parts of the code were removed and marked with code function.

Table of Contents

[Solver architecture 2](#_Toc74594907)

[1. Starting solver (solids4Foam.C) 2](#_Toc74594908)

[2. Initialization of parameters (physicsModel.C) 4](#_Toc74594909)

[3. FSI loop (AitkenCouplingInterface.C) 6](#_Toc74594910)

[Implementing custom solver 8](#_Toc74594911)

[Start your FSI simulation 9](#_Toc74594912)

# Solver architecture

## Starting solver (solids4Foam.C)

The main solver is located at solids4Foam/applications/solvers/solids4Foam/. This code controls whole simulation process by initializing required classes and calling their methods. The solids4Foam.C file has following content:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

#include "fvCFD.H"

#include "physicsModel.H"

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

{

# include "setRootCase.H"

# include "createTime.H"

# include "solids4FoamWriteHeader.H"

**autoPtr<physicsModel> physics = physicsModel::New(runTime);**

while (runTime.run())

{

physics().setDeltaT(runTime);

runTime++;

Info<< "Time = " << runTime.timeName() << nl << endl;

**physics().evolve();**

physics().updateTotalFields();

if (runTime.outputTime())

{

physics().writeFields(runTime);

}

Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"

<< " ClockTime = " << runTime.elapsedClockTime() << " s"

<< nl << endl;

}

physics().end();

Info<< nl << "End" << nl << endl;

return(0);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

The main interest should be paid to two lines, highlighted with bold. The first line searches for dictionary files, located in a case folder, stores these parameters, and initializes required classes for specific simulation setup. This is an example output of this line of code:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

Selecting fluidSolidInterface method Aitken

Selecting fluidModel pimpleFluid

Selecting dynamicFvMesh rotationDeformationFvMesh

Selecting motion solver: velocityLaplacian

Selecting motion diffusion: quadratic

Selecting motion diffusion: inverseDistance

Turbomachine Mixer mesh:

origin: (0 0 0)

axis : (0 1 0)

g field not found in constant directory: initialising to zero

Selecting incompressible transport model Newtonian

Selecting turbulence model type RASModel

Selecting RAS turbulence model kOmegaSST

PIMPLE: Operating solver in PISO mode

Selecting solidModel linearGeometryTotalDisplacement

Selecting dynamicFvMesh solidBodyMotionFvMesh

Selecting solid-body motion function constantVelocity

Creating solidTraction boundary condition

limiter coefficient: 1

Creating fixedDisplacement boundary condition

Creating solidTraction boundary condition

limiter coefficient: 1

Creating fixedDisplacement boundary condition

under-relaxation method: fixed

Creating the mechanicalModel

Selecting mechanical law linearElastic

Selecting interfaceToInterfaceMapping GGI

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

Then, the while loop is started. Despite being a loop, it runs only once per whole simulation. All simulation loops are calculating during the second bold code.

## Initialization of parameters (physicsModel.C)

The first bold line of code from section 1, calls multiple methods with name new. In case of FSI simulations, there methods are located at:

Solids4Foam/src/solids4FoamModels/physicsModel/physicsModel.C

Solids4Foam/src/solids4FoamModels/fluidModels/fluidModel/ fluidModel.C

Solids4Foam/src/solids4FoamModels/solidModels/solidModel/solidModel.C

The content of new method from physicsModel.C is written below:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//Deleted. General class declaration.

Foam::autoPtr<Foam::physicsModel> Foam::physicsModel::New(Time& runTime)

{

word physicsTypeName;

{

**IOdictionary physicsProperties**

**(**

**IOobject**

**(**

**"physicsProperties",**

**runTime.constant(),**

**runTime,**

**IOobject::MUST\_READ,**

**IOobject::NO\_WRITE**

**)**

**);**

physicsProperties.lookup("type")

>> physicsTypeName;

}

Info<< "Selecting physicsModel " << physicsTypeName << endl;

if (physicsTypeName == "fluid")

{

//Deleted. Pure fluid simulation.

}

else if (physicsTypeName == "solid")

{

//Deleted. Pure solid simulation.

}

else **if (physicsTypeName == "fluidSolidInteraction")**

{

word fsiTypeName;

{

**IOdictionary fluidSolidInteractionProperties**

**(**

**IOobject**

**(**

**"fsiProperties",**

**runTime.constant(),**

**runTime,**

**IOobject::MUST\_READ,**

**IOobject::NO\_WRITE**

**)**

**);**

fluidSolidInteractionProperties.lookup("fluidSolidInterface")

>> fsiTypeName;

}

Info<< nl << "Selecting fluidSolidInterface method "

<< fsiTypeName << endl;

fluidSolidInteractionConstructorTable::iterator cstrIter =

fluidSolidInteractionConstructorTablePtr\_->find

(

fsiTypeName

);

if (cstrIter == fluidSolidInteractionConstructorTablePtr\_->end())

{

//Deleted. Error in case if no FSI solver.

}

if (fsiTypeName == "oneWayCoupling" && runTime.value() > 0.0)

{

//Deleted. One-way FSI declaration

}

**return autoPtr<physicsModel>(cstrIter()(runTime));**

}

else

{

//Deleted. Error if nothing was specified.

}

fluidConstructorTable::iterator cstrIter;

return autoPtr<physicsModel>(cstrIter()(runTime));

}

void Foam::physicsModel::writeFields(const Time& runTime)

{

runTime.write();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

This method firstly creates dictionary named physicsProperties and stores in it the content of file physicsProperties, located at constant folder. Then, it searches for parameter named physicsTypeName, and if it was set to FSI, then the program creates new dictionary with FSI properties. As the final step, this method return pointer to the physics class with FSI parameters. This forces to run more new methods from fluidModel.C and solidModel.C. These classes for fluid and solid solvers, dynamic mesh solvers, transport properties, mechanical properties etc.

## FSI loop (AitkenCouplingInterface.C)

In the section 2, the physics class got the FSI interpolation type from fsiProperties. In our case, the FSI interpolation type was set to Aitken, which is located at:

Solids4Foam/src/solids4FoamModels/fluidSolidInterfaces/AitkenCouplingInterface.C

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

bool AitkenCouplingInterface::evolve()

{

initializeFields();

updateInterpolatorAndGlobalPatches();

scalar residualNorm = 0;

if (predictSolid\_)

{

updateForce();

solid().evolve();

residualNorm =

updateResidual();

}

do

{

**outerCorr()++;**

**updateDisplacement();**

**moveFluidMesh();**

**fluid().evolve();**

**updateForce();**

**solid().evolve();**

**residualNorm = updateResidual();**

if (writeResidualsToFile() && Pstream::master())

{

residualFile()

<< runTime().value() << " "

<< outerCorr() << " "

<< residualNorm << endl;

}

}

while (residualNorm > outerCorrTolerance() && outerCorr() < nOuterCorr());

solid().updateTotalFields();

if (additionalMeshCorrection())

{

forAll(fluid().globalPatches(), interfaceI)

{

fluidZonesPointsDisplsPrev()[interfaceI] =

fluidZonesPointsDispls()[interfaceI];

fluidZonesPointsDispls()[interfaceI] += residuals()[interfaceI];

}

moveFluidMesh();

}

return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

In this evolve method, the FSI loop is calculated. The main calculations are highlighted in bold. It starts with incrementing loop. Then, the forces are transferred from solid to fluid. After it, the mesh fluid is deformed. The next step, solving fluid mesh, then transferring forces to the solid domain. The last step is to run the solid solver and calculate residuals. Each fluid().evolve() and solid().evolve() run fluid and solid loops.

# Implementing custom solver

The main problem with rotating and deforming mesh at the same timestep comes from FSI loop. From section 3 of General Concepts, mesh deformation and mesh rotation are happening in moveFluidMesh()and influid().evolve()respectively. Each command calls dynamicMesh solver and expects it to have specific methods. The examples of classes, containing required methods are:

For deformation: $FOAM\_SRC/dynamicMesh/dynamicFvMesh/dynamicMotionSolverFvMesh

For rotation: $FOAM\_SRC/dynamicMesh/dynamicFvMesh/turboFvMesh/

To run FSI simulation with rotation and deformation, the dynamicMesh solver should contain methods from both dynamicMotionSolverFvMesh and turboFvMesh.

The combined class could be found at the following git link:

<https://github.com/MDO-WT-Team/RotationDeformationFvMesh>

Following instructions, the solver should be placed in the following directory:

$FOAM\_SRC/dynamicMesh/dynamicFvMesh/

Then, the solver should be built by running Allwmake file in the parent directory.

# Start your FSI simulation

The first step would be to copy existing tutorial case. You can find them in the tutorial folder. Then, the setup is similar to every OpenFOAM simulations setup:

* Creating fluid and solid meshes
* Setting up FSI interpolation as Aitken
* Changing boundary conditions
* If GGI interpolation is used for sliding mesh, then cell zones should be specified
* Change controlDict, fvScheme, and fvSolution files to match required simulation.

The main requirement is to specify rotating cell zone for the fluid mesh. This cell zone should be specified in new dynamicMeshDict for fluid mesh:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

FoamFile

{

version 2.0;

format ascii;

class dictionary;

object dynamicMeshDict;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

dynamicFvMesh rotationDeformationFvMesh;

solver velocityLaplacian;

diffusivity quadratic inverseDistance (blade);

rotationDeformationFvMeshCoeffs

{

coordinateSystem

{

type cylindrical;

origin (0 0 0);

axis (0 1 0);

direction (1 0 0);

}

rpm

{

rotatingZone -12;

}

slider

{

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//