Cite as: Aryal, P.: Modeling free surface thermal flow with relative motion of heat source and drop injector with respect to a liquid pool. In Proceedings of CFD with OpenSource Software, 2019, Edited by Nilsson. H., http://dx.doi.org/10.17196/OS_CFD#YEAR_2019

CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Modeling free surface thermal flow with relative motion of heat source and drop injector with respect to a liquid pool

Developed for OpenFOAM-3.0.x

Author:
Pradip Aryal
University West
pradip.aryal@hv.se

Peer reviewed by:
Muye Ge
Michael Bertsch

Disclaimer: This is a student project work, done as part of a course where OpenFOAM and some other OpenSource software are introduced to the students. Any reader should be aware that it might not be free of errors. Still, it might be useful for someone who would like learn some details similar to the ones presented in the report and in the accompanying files. The material has gone through a review process. The role of the reviewer is to go through the tutorial and make sure that it works, that it is possible to follow, and to some extent correct the writing. The reviewer has no responsibility for the contents.

January 20, 2020

Learning outcomes

The main goal of the tutorial is to learn how to use and modify interFoam solver by adding the energy conservation equation formulated with the temperature. The reader will also learn to derive the governing equations when considering a moving control volume case. The reader will also learn the implementation of newly formulated governing equations in a new solver. Additionally, the reader will learn to formulate and implement new boundary conditions that are time dependent.

The reader will learn:

How to use it:

- ...Description of the interFoam solver with an accompanying test case is presented. The emphasis is put on understanding the various types of BC at an inlet and at an atmospheric surface.

The theory of it:

- ...The theory of Reynolds transport theorem for a moving control volume is presented. The theory of existing BC and the necessity to develop new BC to model moving inlet BC is presented.

How it is implemented:

- ...The standard governing equations in a modified interFoam solver with energy equation for welding simulation is discussed. A moving heat source is modeled as the surface energy source term active on the liquid surface and treated explicitly in the energy conservation equation. A detailed derivation of the governing equations for moving reference frame is presented. The formulation and implementation of a new BC capable of handling periodical injection of droplet through moving inlet BC is discussed.

How to modify it:

- ...The modification is performed in two steps. Initially, a description of creating a user copy of an existing solver is presented. The interFoam solver is copied and modified with inclusion of energy conservation equation formulated with temperature.

  In the second step, the modified solver from the first step is further modified where governing equations are transformed to a reference frame attached to a moving heat source. The detailed derivation of new sets of governing equations in moving reference frame are derived and presented. Description of an additional term in the governing equations due to the moving reference frame is presented and implemented in the solver. Simulation of falling droplets from inlet boundary condition is performed with the two approaches and the results are compared and discussed.

Prerequisites

The reader is expected to know the following in order to get maximum benefit out of this report:

- Fundementals of Heat and mass Transfer , book by Frank. P Incropera
- Run standard tutorials like damBreak tutorial

Contents

Chapter 1

Introduction

Droplet injection on a pool has a wide range of applications such as inkjet printing, spray cooling, or welding. It is common to observe relative motion between drop injector and the pool in several of the engineering applications. When considering welding for instance, a heat source and a feeding wire moves with uniform speed relative to a base metal. Figure 1.1 shows a typical schematic of a welding process where the wire is moving and melting under the effect of heat source at the same time. The droplets formed by the melting of the wire are transferred into the melt pool. The movement of the wire relative to the base metal results in layers of deposited beads. To simulate such process, it is of common practice to simplify the physics and simply to inject liquid droplets in the computational domain [1] [2].

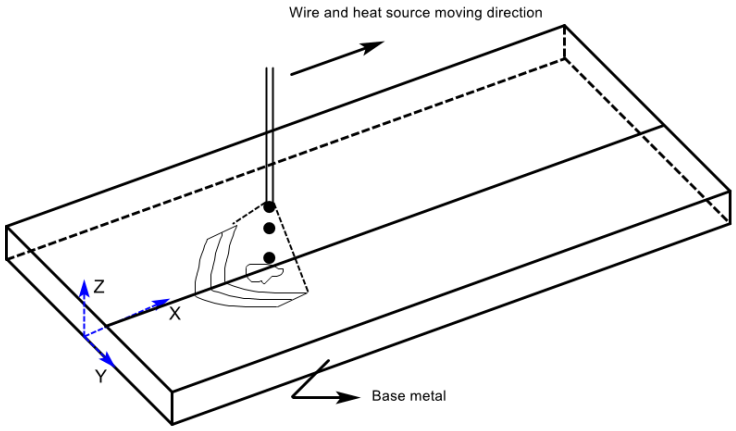

Figure 1.1: Schematic of a moving heat source relative to a base metal.

The problem addressed here is to model the relative motion between a drop injector attached to a heat source and a liquid pool. The relative motion is usually neglected while performing CFD simulations. Modeling the relative motion often requires the implementation of a complex boundary condition which could be computationally expensive. However, the interest in performing CFD simulation often lies in modeling relative motion between two bodies to capture real physics. While modeling relative motion in welding or AM for instance, the most common approach is to implement a moving heat source in a fixed coordinate system [3]. The other approach is to transform the governing equations to a reference frame attached to a moving heat source [4] [5]. The aim of this tutorial is to model relative motion between a drop injector fixed to a heat source and a liquid pool through both approaches.

As the liquid droplets are falling through a gas medium, the multiphase solver interFoam is selected for this purpose. The interFoam sovler is discussed in Chapter 2. As the studied problem is temperature dependent, the interFoam sovler is modified and a new solver is created by adding the energy equation, which is presented in Chapter 3. The incoming heat source is added as an energy source term. The modified interFoam solver with energy equation is further modified in Chapter 4 to create an additional solver for addressing the problem in moving reference frame. The governing equations in this solver are modified to a coordinate system moving with the heat source. The theoretical background to modify the governing equations is based on the Reynolds transport theorem for moving control volume. The modified form of equations are derived and the method to implement in OpenFOAM is discussed in Chapter 4. A new boundary condition capable of handling periodically falling and simultaneously moving droplet is formulated and implemented in Chapter 5. Two identical test cases with slightly different boundary conditions are set up in Chapter 6. Test case 1 with heat source and drop injector moving through the computational domain is solved with solver in fixed coordinate system. In test case 2, the location of the heat source and drop injector remains stationary and the test case is solved with the solver modified for moving reference frame. Results are discussed and analyzed in Chapter 7. Future work is suggested in Chapter 8.

Chapter 2

The interFoam solver

This chapter contains a description of the interFoam solver and the accompanying test case in OpenFOAM version 3.0.1. The solver is located at `FOAM_SOLVERS/multiphase/interFoam`. The interFoam solver is a transient solver for two incompressible isothermal immiscible fluids using the Volume of Fluid, VOF, method based on phase-fraction for interface capturing. In the VOF method implemented in the interFoam solver, the transport equation for volume fraction of one phase is solved simultaneously with the continuity and momentum equations. The thermodynamic and transport properties such as viscosity, density and specific heat are of mixture type and calculated as a weighted average based on the distribution of volume fraction.

2.1 Governing equations

The governing equations solved in the interFoam solver are the continuity equation, momentum equation and conservation equation for phase fraction using VOF method

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U) = 0, \tag{2.1}$$

$$\frac{\partial}{\partial t}\rho U + \nabla \cdot (\rho U U) = -\nabla p + \nabla \cdot (\mu_{eff}(\nabla U + (\nabla U)^T)) + \rho g + F_S, \tag{2.2}$$

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\alpha U) + \nabla \cdot ((1-\alpha)\alpha U_r) = 0. \tag{2.3}$$

where ρ is the density, U the velocity vector, t is the time, p is the pressure, μ_{eff} is the effective viscosity, g the gravitational acceleration, Fs is the surface tension force acting on the interface. Equation 2.3 is the transport equation of the scalar function α. The value of α is 1 in cells with only one phase, 0 in a cells with only the other phase and ranges between 0 and 1 in interface cells containing both of the phases. In Eq. 2.3, the third term on the left hand side is an additional artificial compression term with artificial compressive velocity U_r. The artificial term is introduced to allow necessary compression at the interface.

Open the main source file `interFoam.C` and have a look. The code in the main source file `interFoam.C` starts with several include files to set up a frame work for the finite volume simulation.

```
#include "fvCFD.H"
#include "CMULES.H"
#include "EulerDdtScheme.H"
#include "localEulerDdtScheme.H"
```

```
#include "CrankNicolsonDdtScheme.H"
#include "subCycle.H"
#include "immiscibleIncompressibleTwoPhaseMixture.H"
#include "turbulentTransportModel.H"
#include "pimpleControl.H"
#include "fvIOoptionList.H"
#include "CorrectPhi.H"
#include "fixedFluxPressureFvPatchScalarField.H"
#include "localEulerDdtScheme.H"
#include "fvcSmooth.H"
```

It is followed by the main function. It contains several classes for time and mesh control.

```
int main(int argc, char *argv[])
{
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"

    pimpleControl pimple(mesh);

    #include "createTimeControls.H"
    #include "createRDeltaT.H"
    #include "initContinuityErrs.H"
    #include "createFields.H"
    #include "createMRF.H"
    #include "createFvOptions.H"
    #include "correctPhi.H"

    if (!LTS)
    {
        #include "readTimeControls.H"
        #include "CourantNo.H"
        #include "setInitialDeltaT.H"
    }
```

The time loop is initiated with `{while (runTime.run())}`. The new time step is calculated in the time loop based on the Courant Number. The major concern in free surface simulation using VOF method is the conservation of phase fraction [6]. A stable solution of phase fraction, requires small time step. In order to achieve stable solution without significantly increasing simulation time, the time step is divided and the calculation of phase fraction is performed in a number of sub cycle in `\#include alphaEqnSubCycle.H`. Inside the time loop, pimple loop is initiated for pressure-velocity coupling.

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

    while (runTime.run())
    {
```

```
        #include "readTimeControls.H"

        if (LTS)
        {
            #include "setRDeltaT.H"
        }
        else
        {
            #include "CourantNo.H"
            #include "alphaCourantNo.H"
            #include "setDeltaT.H"
        }

        runTime++;

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

        // --- Pressure-velocity PIMPLE corrector loop
        while (pimple.loop())
        {
            #include "alphaControls.H"
            #include "alphaEqnSubCycle.H"

            mixture.correct();

            #include "UEqn.H"

            // --- Pressure corrector loop
            while (pimple.correct())
            {
                #include "pEqn.H"
            }

            if (pimple.turbCorr())
            {
                turbulence->correct();
            }
        }

        runTime.write();

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }

    Info<< "End\n" << endl;

    return 0;
}
```

`#include UEqn.H` initiates the solving of momentum equation mentioned in Eq. 2.2. In OpenFOAM language, it is written as

```
fvVectorMatrix UEqn
    (
        fvm::ddt(rho, U)
      + fvm::div(rhoPhi, U)
      + MRF.DDt(rho, U)
      + turbulence->divDevRhoReff(rho, U)
     ==
        fvOptions(rho, U)
    );

    UEqn.relax();
```

`#include UEqn.H` is followed by a pressure correction loop which contains `#include pEqn.H` that iteratively calculates and corrects the pressure value. Turbulence is also corrected in the same loop after the correction of pressure value. The time step finishes with runTime.write() which writes out the information about time and residuals. A new time step is calculateed again based on the Courant number. It continues until the criteria for convergence are met, or the maximum number of iterations is reached.

2.2 Tutorial

A comprehensive set of tutorials for multiphase simulation using the interFoam solver is available in OpenFOAM package. All the tutorials are accessed by environment variable `tut` in terminal window after the initialization of OpenFOAM. Go to

```
cd FOAM_TUTORIALS/multiphase/interFoam/laminar
ls
```

There are three test cases. We will look into damBreak tutorial. Copy the test case to the appropriate user directory.

```
cd FOAM_RUN
cp -r $FOAM_TUTORIALS/multiphase/interFoam/laminar/damBreak .
cd damBreak
```

The test case folder contains three sub-directories, they are: *0*, *constant* and *system*. First, we will have a look at the *constant* directory.

```
cd constant
```

The folder contains the properties of the fluid and the parameters that remains constant during the simulation. We see that the sub-directories contain four files namely `dynamicMeshDict`, `g`, `transportProperties` and `turbulenceProperties`. The `g` file sets the value of the gravitational acceleration and its direction. The `dynamicMeshDict` file specifies whether the mesh is static or dynamic. The `turbulenceProperties` file specifies the turbulence model to be used in the simulation. Since no turbulence model is used in this test case, it is set to `laminar`. The `transportProperties` file contains information about the properties of the fluids. The fluid properties in the test case are kinematic viscosity (ν) and density (ρ) of air and water specified by the keyword *nu* and *rho* respectively. Additionally, the surface tension (σ) of water is also specified by the keyword *sigma*.

The *system* sub-directory contains the dictionary `blockMeshDict` where the information about the mesh such as vertices, blocks, name and type of boundary surface is stored. You can generate the

mesh by using `blockMesh` utility. The other dictionaries in the system-directory are related to simulation settings. The `controlDict` contains the parameters for simulation control. The `fvSchemes` dictionaries consist of the numerical discretization scheme for each term. The `fvSolution` dictionary consists of all the setting concerning solver, relaxation factors, maximum residuals and pressure-velocity correction loop.The `decomposeParDict` dictionary contains information to decompose the test case to run in parallel mode. The `setFiedlsDict` dictionary contains information to initialize specific fields in a certain region of the domain.

The *0* sub-directory contains the individual files of the fields e.g. velocity and pressure. In each file, the value of the variables at the initial condition and the boundary condition are specified. In the *damBreak* tutorial, there are three variables `alpha.water.org`, `p_rgh` and `U`.

Open `alpha.water.org` file with the editor of your choice.

`0/alpha.water.org`

The BC for alpha.water on all the walls is set to `zeroGradient`. The BC on the atmospheric surface is set to `inletOutlet`. `inletOutlet` BC is identical to `zeroGradient` BC when the flow goes out of the computational domain. However, in case of backward flow, when the velocity vector next to the patch of atmospheric surface is directed into the domain, `inletOutlet` BC changes to `fixedValue` type BC. The value of the `fixedValue` is given in this case by `inletValue`.

Open p_rgh file.

`0/p_rgh`

The BC for `p\_rgh` on all the walls is set to `fixedFluxPressure`. This boundary condition precedes the generally used `zeroGradient` BC for `p_rgh` when the solution domain contains body forces such as gravity and surface tension. The BC on the atmosphere surface is set to `totalPressure`.

Open U file.

`0/U`

The BC for `U` on all the walls is set to `fixedValue` type with 0 velocity in each direction. The BC on the atmosphere surface is set to `pressureInletOutletVelocity`. The `pressureInletOutletVelocity` BC is identical to `zeroGradient` BC for outflow. For inflow, the velocity is obtained from the the patch-face normal component of the internal-cell value.

In the damBreak tutorial, the computational domain is 2D and a column of water is initialized in a rectangular field near left wall. Copy the original file in 0 sub-directory before running setFields utility

```
cp 0/alphawater.org 0/alphawater
```

Now, run setFields utility by typing `setFields`.

Run the test case and write the log-file

```
interFoam >& log.damBreak
```

when the simulation is complete, visualize the results in paraview by running `paraFoam`.

Chapter 3

Free surface thermal flow

The energy conservation equation formulated with temperature is included for free surface thermal flow analysis in the `interFoam` solver. The modified solver is named `interThermalFoam`. The modified solver is used to simulate temperature distribution, droplet injection and its impact on the surface of the liquid pool. The energy conservation equation in CFD in terms of partial derivatives is written as

$$\rho Cp\frac{\partial T}{\partial t} + \nabla \cdot (\rho CpUT) = \nabla \cdot (K\nabla T) + S_T, \tag{3.1}$$

where Cp is the specific heat, T is the temperature, K the thermal conductivity and S_T is the heat source term. The heat source term in the welding application for instance could be in the form of laser, arc, electron beam etc. The intensity of these heat sources in CFD simulation are usually assumed to have Gaussian [4], top-hat [7], and double-ellipsoid [8], [9] distribution. The heat source term with a Gaussian distribution is assumed in this study and given as

$$Qhs = \frac{\eta\dot{Q}}{\pi {r_L}^2} exp\left(-\frac{\eta r2}{{r_L}^2}\right), \tag{3.2}$$

where η is the efficiency, $\dot{Q}$ is the power, r is the radial coordinate (distance between a point and the centre of the heat source in x and y directions) and r_L is the effective radius.

3.1 The energy equation

The `interFoam` solver in OpenFOAM version 3.0.1 is modified to include the energy conservation equation mentioned in Eq. 3.1. The first step is to copy the `interFoam` solver to the user defined directory for solvers and change the name of the solver directory and main solver file.

```
mkdir $WM_PROJECT_USER_DIR/applications/solvers
cd $WM_PROJECT_USER_DIR/applications/solvers/
cp -r $FOAM_SOLVERS/multiphase/interFoam/ .
mv interFoam interThermalFoam
cd interThermalFoam
mv interFoam.C interThermalFoam.C
```

Now, update *Make/files* either by opening via text editor and manually modifying or by running the commands

```
sed -i s/"interFoam"/"interThermalFoam"/g Make/files
sed -i s/"FOAM_APPBIN"/"FOAM_USER_APPBIN"/g Make/files
```

Once again make sure the code in *Make/files* looks like

```
interThermalFoam.C
EXE = $(FOAM_USER_APPBIN)/interThermalFoam
```

Create a file named `TEqn.H` and in it write the following lines of code to implement energy conservation equation

```
surfaceScalarField alpha1f = min(max(fvc::interpolate(alpha1), scalar(0)), scalar(1));
kappaf = alpha1f*kappa1 + (1.0 - alpha1f)*kappa2;

fvScalarMatrix TEqn
    (
          fvm::ddt(rhoCp, T)
         +fvm::div(rhoPhiCpf, T)
         -fvm::laplacian(kappaf, T)
    );

TEqn.relax();

solve ( TEqn == Qhs*mag(fvc::grad(alpha1))*factor );
```

In the above energy conservation equation written in OpenFOAM language, *Qhs* is the heat source term with Gaussian distribution given in Eq. 3.2. The heat source term is explicitly coupled with the energy equation. The definition of the parameters and initialization of the heat source is discussed in section 3.2.

The variable `rhoCp` depends on alpha and it is updated with every sub cycle of the alpha equation. Open `alphaEqnSubCycle.H` with the editor of your choice and add the following line of code at the end.

```
rhoCp == alpha1*rho1*cp1 + alpha2*rho2*cp2;
```

The variable `rhoPhiCpf` is updated with every alpha iteration. Open `alphaEqn.H` and add the following line of code after the equation of `rhoPhi` at two instances.

```
rhoPhiCpf = alphaPhi*(rho1*cp1 - rho2*cp2) + phiCN*rho2*cp2;
```

The density ratio between liquid pool and gas phase can be very high. A high density ratio generates spurious current at the interface due to smearing of volume forces. Therefore, a redistribution term, called "factor" is used to redistribute the forces to the phase with higher density. The approach of redistribution term was first proposed and used by Brackbill [10]. The equation for factor is given as

$$factor = \frac{2\rho}{\rho 1 + \rho 2}. \tag{3.3}$$

The variable `factor` is updated with the update of $\rho 1$ and $\rho 2$ in every subcycle of alpha equation. Open `alphaEqnSubCycle.H` and add the following line of code between the equation of `rho` and `rhoCp`.

```
factor = scalar(2)*rho/(rho1+rho2);
```

Now, we should declare and initialize the new fields written in `TEqn.H`. We need three fields of type `volScalarField` (`T`, `rhoCp`, and `factor`), three of type `surfaceScalarField` (i.e. `alpha1f`, `kappaf`, and `rhoPhiCpf`) and four of type `dimensionedScalar` (i.e. `cp1`, `cp2`, `kappa1`, and `kappa2`). Open `createFields.H` and add the following lines of code after the end of `volScalarField rho` (i.e. `rho.oldTime();`) and before `surfaceScalarField rhoPhi`.

```
// for fields specific to the extension of interFoam to interThermalFoam
Info<< "Reading field T\n" << endl;
volScalarField T
(
       IOobject
       (
           "T",
           runTime.timeName(),
           mesh,
           IOobject::MUST_READ,
           IOobject::AUTO_WRITE
       ),
       mesh
);

Info<< "Reading thermophysicalProperties of phase 1\n" << endl;
IOdictionary thermophysicalPropertiesPhase1
(
       IOobject
       (
           "thermophysicalPropertiesPhase1",
           runTime.constant(),
           mesh,
           IOobject::MUST_READ,
           IOobject::NO_WRITE
        )
);

Info << "Creating pi \n" << endl;
dimensionedScalar pi = constant::mathematical::pi;

// specific heat of phase 1
Info<< "Reading cp1l \n" << cp1 << endl;
const dimensionedScalar& cp1 = thermophysicalPropertiesPhase1.lookup("cp1");

// thermal conductivity of phase 1
const dimensionedScalar& kappa1 = thermophysicalPropertiesPhase1.lookup("kappa1");
Info<< "Reading kappa1 \n" << kappa1 << endl;

Info<< "Reading thermophysicalProperties of phase 2\n" << endl;
IOdictionary thermophysicalPropertiesPhase2
(
       IOobject
       (
           "thermophysicalPropertiesPhase2",
           runTime.constant(),
           mesh,
           IOobject::MUST_READ,
           IOobject::NO_WRITE
       )
);
```

```
// specific heat of phase 2
Info<< "Reading cp2 \n" << cp2 << endl;
const dimensionedScalar& cp2 = thermophysicalPropertiesPhase2.lookup("cp2");

// thermal conductivity of phase 2
const dimensionedScalar& kappa2 = thermophysicalPropertiesPhase2.lookup("kappa2");
Info<< "Reading kappa2 \n" << kappa2 << endl;

Info<< "Reading / initializing kappaf\n" <<endl;
surfaceScalarField kappaf
(
       IOobject
       (
           "kappaf",
           runTime.timeName(),
           mesh,
           IOobject::NO_READ,
           IOobject::NO_WRITE
        ),
       mesh,
       dimensionedScalar("kappaf",dimensionSet(1,1,-3,-1,0,0,0), 0.0)
);


Info<< "Reading / calculating rho*cp\n" <<endl;
volScalarField rhoCp
(
        IOobject
        (
            "rho*cp",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::NO_WRITE
        ),
        alpha1*rho1*cp1+alpha2*rho2*cp2,
        alpha1.boundaryField().types()
);
rhoCp.oldTime();


Info<< "Reading / calculating rhoPhi*cp\n" <<endl;
surfaceScalarField rhoPhiCpf
(
     IOobject
        (
            "rhoPhiCpf",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::NO_WRITE
```

```
        ),
        fvc::interpolate(rhoCp)*phi
);

volScalarField factor
(
     IOobject
        (
            "factor",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::NO_WRITE
     ),
     scalar(2)*rho/(rho1+rho2)
);
```

3.2 Creating fields

The next step is to create fields for the heat source and define and declare the variables required for initialization and calculation of heat source mentioned in Eq. 3.2. Create a new file named `createLaserFields.H` and add

```
Info<< "Reading the laser beam heat source properties\n" << endl;
IOdictionary beamProperties
(
        IOobject
        (
            "beamProperties",
            runTime.constant(),
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
);

//Weld (or heat source) speed
dimensionedVector Ulaser
(
        "Ulaser",
        dimensionSet(0, 1, -1, 0, 0, 0, 0),
     beamProperties.lookup("Ulaser")
);

//Initial position of laser beam
dimensionedVector Xbeam0
(
        "Xbeam0",
        dimensionSet(0, 1, 0, 0, 0, 0, 0),
```

```
     beamProperties.lookup("Xbeam0")
);

//laser starting time
dimensionedScalar timeStartLaser
(
        "timeStartLaser",
        dimensionSet(0, 0, 1, 0, 0, 0, 0),
     beamProperties.lookup("timeStartLaser")
);

//radius of laser beam
dimensionedScalar rBeam
(
        "rBeam",
        dimensionSet(0, 1, 0, 0, 0, 0, 0),
     beamProperties.lookup("rBeam")
);

//efficiency of laser beam
dimensionedScalar eta_L
(
        "eta_L",
        dimensionSet(0, 0, 0, 0, 0, 0, 0),
     beamProperties.lookup("eta_L")
);

//laser power
dimensionedScalar Q_L
(
        "Q_L",
        dimensionSet(1, 2, -3, 0, 0, 0, 0),
     beamProperties.lookup("Q_L")
);

// local position vector
const volVectorField& cellCentre = mesh.C();

volScalarField r_local = Foam::sqrt
    (
        (Foam::sqr(cellCentre.component(vector::X)-(Xbeam0.component(vector::X)
       +(Ulaser.component(vector::X)*runTime.time())))
       +Foam::sqr(cellCentre.component(vector::Z)-(Xbeam0.component(vector::Z)
       +(Ulaser.component(vector::Z)*runTime.time())))
    );


//Info<< "Reading field Qhs \n" << endl;
volScalarField Qhs
(
        IOobject
        (
```

```
            "Qhs",
            runTime.timeName(),
            mesh,
            IOobject::READ_IF_PRESENT,
            IOobject::AUTO_WRITE
        ),
        mesh,
        dimensionedScalar
        (
             "Qhs",
             dimensionSet(1, 0, -3, 0, 0, 0, 0),
             scalar(0.0)
        )
);

// power density distribution factor
scalar dl = 3.5;

Qhs = eta_L*Q_L/(pi*Foam::sqr(rBeam))*dl*Foam::exp(-dl*(Foam::sqr(r_local)
      /Foam::sqr(rBeam)));
```

3.3 Moving heat source

So far we have added the energy equation, defined and declared the required variables and initialized the heat source. In order to move the heat source, the local position vector `r_local` needs to be updated at each `deltaT`. Create a file named `updateLaserFields.H` and add

```
    //  For a moving heat source with speed Ulaser.
    volScalarField r_local = Foam::sqrt
    (
        (Foam::sqr(cellCentre.component(vector::X)-(Xbeam0.component(vector::X)
       +(Ulaser.component(vector::X)*runTime.time())))
       +Foam::sqr(cellCentre.component(vector::Z)-(Xbeam0.component(vector::Z)
       +(Ulaser.component(vector::Z)*runTime.time())))
    );

    Qhs = eta_L*Q_L/(pi*Foam::sqr(rBeam))*dl*Foam::exp(-dl*(Foam::sqr(r_local)
          /Foam::sqr(rBeam)));
```

In the final step, include new files to the source file `interThermalFoam.C`. Open `interThermalFoam.C` and after `#include "correctPhi.H"`, add

```
#include "createLaserFields.H"
```

Then at the end of while (pimple.loop()), add

```
// update and move laser heat source
#include "updateLaserField.H"
```

```
// energy conservation equation (expressed for T)
#include "TEqn.H"
```

Save and close the file. Now clean and compile.

```
wclean
wmake
```

Thus we added energy equation and explicitly coupled it with the heat source. The new variables are declared, the new fields are constructed and initialized. After successful compilation, the solver can be tested for the first test case with the moving heat source. The detail of the test case is described in chapter 6.

Chapter 4

Governing equations in MRF

4.1 Reynolds transport theorem

When analyzing fluid flow, the finite region where the fundamental laws of conservation of mass, momentum and energy are formulated is called control volume. The control volume concept allows the study of net effect when fluid flows in and out of the finite region [11] . The study of fluid flow with CFD mostly uses the concept of control-volume analysis. In control volume analysis, the domain of interest is referred to system where the laws of fluid mechanics apply. The system is separated from its surroundings by boundaries where the conditions of the fluids are usually known or closely approximated. The most general form of Reynolds transport theorem for a fixed control volume is written as

$$\frac{d}{dt}(B_{syst}) = \frac{d}{dt}\left(\int_{CV} \beta\rho dv\right) + \left(\int_{CS} \beta\rho(V \cdot n)dA\right), \tag{4.1}$$

where B is any property of the fluid in the system such as energy or momentum and β is the amount of B per unit mass in any small portion of the fluid, t is time, CV refers to control volume, CS refers to control volume surface, ρ is the density of the fluid, dv is the volume of the small portion of the fluid and V is the velocity of the fluid passing through the small control volume surface, n is the outward pointing vector perpendicular to the control volume surface and dA is the area of a small portion of the control volume surface.

4.2 Reynolds transport theorem for moving control volume

Most fluid mechanics problems are solved using fixed control volume. However, there are cases when moving control volume is needed. Moving control volume simplifies the computationally expensive fluid flow problem in many cases. Some cases where moving control volume is more efficient than the fixed control volume is in the analysis of flow through jet engines, exhaust flow through a rocket nozzle, or a moving ship on the surface of water.

In case of a moving control volume, an observer fixed to the control volume experiences relative velocity V_{rel} of fluid crossing through the control volume surface given by

$$V_{rel} = V - V_{CV}, \tag{4.2}$$

where V is the absolute velocity of the fluid and V_{CV} is the velocity of the moving control volume. Thus the Reynolds transport theorem for a uniformly moving control volume is written as

$$\frac{d}{dt}(B_{syst}) = \frac{d}{dt}\left(\int_{CV} \beta\rho dv\right) + \left(\int_{CS} \beta\rho(V_{rel} \cdot n)dA\right). \tag{4.3}$$

4.3 Motion in MRF

The solution of governing equations in a fixed coordinate system requires a significantly finer grid for precise representation of the spatial distribution of a moving heat source. Additionally, in welding, AM and similar applications, several molten droplets of metal impinge on the melt pool per second. Therefore, a very small time step size is necessary for stable solution and to accurately predict the dynamics of impinging droplets. In order to reduce computation time and cost, such a process can be modeled on a moving reference frame with coordinate system attached to a moving heat source. Thus, for a constant heat source speed and a sufficiently elongated liquid pool, the problem transforms into a steady state after a short time. In such case, the heat source and droplet injector are fixed in space whereas the liquid pool uniformly moves with a certain velocity. The velocity of the liquid pool is the same as the heat source speed but in opposite direction.

In this chapter, the governing equations in the solver `interThermalFoam` are transformed to a moving reference frame. As explained in section 4.2, the velocity of the computational domain in MRF when the coordinate system is attached to a moving heat source is given as

$$U_{rel} = U - U_{weld} \, , \tag{4.4}$$

where U_{rel} is the velocity at any point in the computational domain in MRF. U is the absolute convective velocity of the fluid and U_{weld} is the uniform velocity of the moving coordinate system. Using Eq. 4.4, the governing equations in the `interThermalFoam` solver introduced in chapter 3 are modified to a moving reference frame. The derivation is explained in the following section.

4.4 Derivation

4.4.1 Continuity equation

The continuity equation in the `interThermalFoam` solver is given as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U) = 0 \, . \tag{4.5}$$

Replacing the convective velocity U by the relative velocity U_{rel}, we get

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U_{rel}) = 0 \, . \tag{4.6}$$

Substuting U_{rel} from Eq. 4.4 yields

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho (U - U_{weld})) = 0 \, , \tag{4.7}$$

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U) - \nabla \cdot (\rho U_{weld}) = 0 \, . \tag{4.8}$$

The third term on the left hand side becomes zero since the density, ρ and the welding velocity, U_{weld} are both constant. Thus we end up with the unchanged continuity equation as in Eq. 4.5, namely

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U) = 0 \, . \tag{4.9}$$

4.4.2 Momentum equation

The momentum equation in the `interThermalFoam` solver is given as

$$\frac{\partial}{\partial t}\rho U + \nabla \cdot (\rho U U) = -\nabla p + \nabla \cdot (\mu_{eff}(\nabla U + (\nabla U)^T)) + \rho g + F_S \ . \tag{4.10}$$

Replacing the convective velocity U by the relative velocity U_{rel} we get

$$\frac{\partial}{\partial t}\rho U_{rel} + \nabla \cdot (\rho U_{rel} U_{rel}) = -\nabla p + \nabla \cdot (\mu_e f f(\nabla U_{rel} + (\nabla U_{rel})^T)) + \rho g + F_S \ . \tag{4.11}$$

Substituting U_{rel} from Eq. 4.4 yields

$$\begin{aligned} \frac{\partial}{\partial t}\rho(U - U_{weld}) + \nabla \cdot (\rho(U - U_{weld})(U - U_{weld})) = \\ -\nabla p + \nabla \cdot (\mu_e f f(\nabla(U - U_{weld}) + (\nabla(U - U_{weld}))^T)) + \rho g + F_S \ . \end{aligned} \tag{4.12}$$

The time derivative term on the left hand side of the Eq. 4.12 is simplified as

$$\frac{\partial}{\partial t}\rho(U - U_{weld}) = \frac{\partial}{\partial t}\rho U - \frac{\partial}{\partial t}\rho U_{weld} = \frac{\partial}{\partial t}\rho U \ . \tag{4.13}$$

Here, $\frac{\partial}{\partial t}\rho U_{weld}$ is 0 since ρ and U_{weld} are both constant. The convective term on the left hand side of the Eq. 4.12 is simplified as

$$\begin{aligned} \nabla \cdot (\rho(U - U_{weld})(U - U_{weld})) = \\ \nabla \cdot (\rho U U) - \nabla \cdot (\rho U_{weld} U) - \nabla \cdot (\rho U U_{weld}) + \nabla \cdot (\rho U_{weld} U_{weld}) \ . \end{aligned} \tag{4.14}$$

The first term on the right hand side of Eq. 4.14 remains as it is. The second term on the right hand side of Eq. 4.14 can be expanded into

$$\nabla \cdot (\rho U_{weld} U) = \rho \cdot (U_{weld} \cdot \nabla U) + U(U_{weld} \cdot \nabla \rho) \ . \tag{4.15}$$

since ρ is constant, the second term becomes 0 and we end up with

$$\nabla \cdot (\rho U_{weld} U) = \rho \cdot (U_{weld} \cdot \nabla U) \ . \tag{4.16}$$

Similarly, the third and the fourth term on the right hand side of Eq. 4.14 also reduces to 0 since ρ and U_{weld} are both constant. Therefore the final solution of 4.14 can be written as

$$\nabla \cdot (\rho(U - U_{weld})(U - U_{weld})) = \nabla \cdot (\rho U U) - \rho \cdot (U_{weld} \cdot \nabla U) \ . \tag{4.17}$$

The diffusive term on the right hand side of Eq. 4.12 is expanded as

$$\begin{aligned} \nabla \cdot (\mu_e f f(\nabla(U - U_{weld}) + (\nabla(U - U_{weld}))^T)) = \\ \nabla \cdot (\mu_e f f(\nabla(U) - \nabla(U_{weld})) + (\nabla(U)^T - (\nabla(U_{weld}))^T)) \ . \end{aligned} \tag{4.18}$$

Since U_{weld} is constant, $\nabla(U_{weld})$ becomes 0. Therefore, the diffusive term remains the same as the original one in Eq. 4.10 namely

$$\nabla \cdot (\mu_e f f(\nabla(U - U_{weld}) + (\nabla(U - U_{weld}))^T)) = \nabla \cdot (\mu_e f f(\nabla(U) + (\nabla(U)^T) \ . \tag{4.19}$$

From Eq. 4.11, Eq. 4.13, Eq. 4.17, and Eq. 4.19, the final form of momentum equation in moving reference frame can be written as

$$\frac{\partial}{\partial t}\rho U + \nabla \cdot (\rho U U) - \rho \cdot (U_{weld} \cdot \nabla U) = -\nabla p + \nabla \cdot (\mu_e f f(\nabla U + (\nabla U)^T)) + \rho g + F_S \ . \tag{4.20}$$

4.4.3 Alpha equation

The equation of conservation of phase fraction given by alpha equation in the `interThermalFoam` solver is written as

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\alpha U) + \nabla \cdot ((1-\alpha)\alpha U_r) = 0 \,. \tag{4.21}$$

Replacing the convective velocity U by the relative velocity U_{rel} we get

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\alpha U_{rel}) + \nabla \cdot ((1-\alpha)\alpha U_r) = 0 \,. \tag{4.22}$$

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\alpha (U - U_{weld})) + \nabla \cdot ((1-\alpha)\alpha U_r) = 0 \,. \tag{4.23}$$

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\alpha U) - \nabla \cdot (\alpha U_{weld}) + \nabla \cdot ((1-\alpha)\alpha U_r) = 0 \,. \tag{4.24}$$

The third term on the left hand side is expanded into

$$\nabla \cdot (\alpha U_{weld}) = (\nabla \alpha) \cdot U_{weld} + \alpha \cdot (\nabla \cdot U_{weld}) \,. \tag{4.25}$$

Since, U_{weld} is constant, $(\nabla \cdot U_{weld})$ is 0. So we end up with

$$\nabla \cdot (\alpha U_{weld}) = (\nabla \alpha) \cdot U_{weld} \,. \tag{4.26}$$

From Eq. 4.26 and 4.24, the final form of the alpha equation in reference frame is written as

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\alpha U) - (\nabla \alpha) \cdot U_{weld} + \nabla \cdot ((1-\alpha)\alpha U_r) = 0 \,. \tag{4.27}$$

4.4.4 Energy equation

The energy conservation equation in `interThermalFoam` solver introduced in chapter 3 is given in Eq. 4.28

$$\rho Cp \frac{\partial T}{\partial t} + \nabla \cdot (\rho Cp U T) - \nabla \cdot (K \nabla T) = 0 \,. \tag{4.28}$$

Replacing the convective velocity U by the relative velocity U_{rel}, we get

$$\rho Cp \frac{\partial T}{\partial t} + \nabla \cdot (\rho Cp U_{rel} T) - \nabla \cdot (K \nabla T) = 0 \,. \tag{4.29}$$

$$\rho Cp \frac{\partial T}{\partial t} + \nabla \cdot (\rho Cp (U - U_{weld}) T) - \nabla \cdot (K \nabla T) = 0 \,. \tag{4.30}$$

$$\rho Cp \frac{\partial T}{\partial t} + \nabla \cdot (\rho Cp U T) - \nabla \cdot (\rho Cp U_{weld} T) - \nabla \cdot (K \nabla T) = 0 \,. \tag{4.31}$$

The first and the second term on the left hand side of the equation Eq. 4.31 remains as it is. The third term on the left had side of the equation can be expanded into

$$\nabla \cdot (\rho Cp U_{weld} T) = \rho Cp \cdot (U_{weld} \cdot \nabla T)) + T(U_{Weld} \cdot \nabla (\rho Cp)) \,. \tag{4.32}$$

From Eq. 4.31 and 4.32, the final form of the energy conservation equation in moving reference frame is written as

$$\rho Cp \frac{\partial T}{\partial t} + \nabla \cdot (\rho Cp U T) - \nabla \cdot (K \nabla T) = \rho Cp \cdot (U_{weld} \cdot \nabla T)) + T(U_{Weld} \cdot \nabla (\rho Cp)) \,. \tag{4.33}$$

4.5 Implementation in OpenFOAM

In order to implement the governing equations in MRF in OpenFOAM, firstly copy `interThermalFoam` sovler developed in Chapter 3 to a new solver named interThermalReferenceFoam solver in your user application solver directory.

```
cd $WM_PROJECT_USER_DIR/applications/solvers/
cp -r interThermalFoam interThermalReferenceFoam
cd interThermalReferenceFoam
mv interThermalFoam.C interThermalReferenceFoam.C
```

Now, update *make/files* options either by opening and manually writing or by running command as

```
sed -i s/"interThermalFoam"/"interThermalReferenceFoam"/g Make/files
```

Once again make sure the code in *make/files* looks like

```
interThermalReferenceFoam.C
EXE = $(FOAM_USER_APPBIN)/interThermalReferenceFoam
```

The *Make/options* remains unchanged. Now clean and compile.

```
wclean
wmake
```

4.5.1 Modification of UEqn.H

The derivation of momentum equation in Eq. 4.20 in MRF has an additional divergence term induced when a coordinate system is attached to a moving heat source with velocity U_{weld}. The additional term $-\rho \cdot (U_{weld} \cdot \nabla U)$ is written in OpenFOAM language as - rho*(fvc::grad(U) & Uweld). Open `UEqn.H` and add it on the left hand side of the `fvVectorMatrix UEqn`.

```
    fvVectorMatrix UEqn
    (
        fvm::ddt(rho, U)
        + fvm::div(rhoPhi, U)
        - rho*(fvc::grad(U) & Uweld) //additional term in MRF
        + MRF.DDt(rho, U)
        + turbulence->divDevRhoReff(rho, U)
     ==
        fvOptions(rho, U)
    );

    UEqn.relax();
```

Save and close the file.

4.5.2 Modification of alphaEqn.H

The derivation of the alpha equation in Eq. 4.27 in MRF has an additional divergence term induced when a coordinate system is attached to a moving heat source with velocity U_{weld}. The additional term $(\nabla \alpha) \cdot U_{weld}$ in OpenFOAM language is written as (Uweld & fvc::grad(alpha1)). Open `alphaEqn.H` and add it on the left hand side of the `fvScalarMatrix alpha1Eqn`.

```
        fvScalarMatrix alpha1Eqn
        (
            (
                LTS
              ? fv::localEulerDdtScheme<scalar>(mesh).fvmDdt(alpha1)
              : fv::EulerDdtScheme<scalar>(mesh).fvmDdt(alpha1)
            )
          + fv::gaussConvectionScheme<scalar>
            (
                mesh,
                phiCN,
                upwind<scalar>(mesh, phiCN)
            ).fvmDiv(phiCN, alpha1)

            - (Uweld & fvc::grad(alpha1)) //additional term in MRF

        );
```

Save and close the file.

4.5.3 Modification of TEqn.H

The derivation of energy equation in MRF in Eq. 4.32 has two additional divergence term induced when a coordinate system is attached to a moving heat source with velocity U_{weld}. To implement the governing equations in MRF, open `TEqn.H` and add two lines of code on the right hand side of `fvScalarMatrix TEqn` as shown

```
fvScalarMatrix TEqn
 (
      fvm::ddt(rhoCp, T)
   // convective term
    + fvm::div(rhoPhiCpf, T)
   // heat conduction
    - fvm::laplacian(kappaf, T)

     ==
     rhoCp * (Uweld & fvc::grad(T))         //additional term in MRF
   + T     * (Uweld & fvc::grad(rhoCp))     //additional term in MRF

 );

 TEqn.relax();

    solve ( TEqn == Qhs*mag(fvc::grad(alpha1))*factor );
```

Save and close the file.

4.5.4 Construct fields

The next step is to define a vector field U_{weld}. Open `createLaserFields.H` and just below the declaration of `dimensionedVector Ulaser`, add

```
Info<< "Reading field Uweld\n" << endl;
volVectorField Uweld
```

```
(
    IOobject
    (
        "Uweld",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
        ),
        mesh,
        dimensionedVector("Uweld",dimensionSet(0,1,-1,0,0,0,0), Ulaser.value())
);
```

For a solver in MRF, the heat source and drop injector stays stationary at its initial location and does not need to be moved and updated at each deltaT. Open the main solver file `interThermalReferenceFoam.C` and comment or delete `#include "updateLaserFields.H"`

```
// move the laser source
//#include "updateLaserFields.H"
```

Save and close interThermalReferenceFoam.C. Open `createLaserFields.H` file and change the equation for `r_local` as

```
    volScalarField r_local = Foam::sqrt
    (
        Foam::sqr(cellCentre.component(vector::X)-(Xbeam0.component(vector::X)))
       +Foam::sqr(cellCentre.component(vector::Z)-(Xbeam0.component(vector::Z)))
    );
```

Save and close createLaserFields.H. Now clean and compile.

```
wclean
wmake
```

In Chapter 4, the new solver named interThermalReferenceFoam is created and the governing equations are modified in the moving reference frame. The equation of the local position vector `r_local` is updated to keep the heat source stationary. After the compilation, the solver can be tested for the second test case where heat source and drop injector remain fixed in a moving coordinate system. The details of the test case are described in chapter 6.

Chapter 5

Formulating a new boundary condition

As the heat source moves along the liquid pool, the droplets periodically fall into the liquid pool at certain frequency. To simplify this phenomenon, liquid droplets are periodically injected through the inlet boundary. However, there is no standard OpenFOAM BC to handle such a recurring and moving inlet boundary condition. Therefore, a new BC is formulated in this chapter where the user can input the initial location of the droplet, frequency of droplet formation, radius of droplet, velocity of falling droplet and velocity of droplet moving parallel to the liquid pool.

5.1 Getting started

The usual way to start formulating a new boundary condition is to first find the existing one that would be most closest to what you would like to do. Here, we will first copy `oscillatingFixedValue` boundary condition to the user directory and rename `localTranslatingFixedValue`.

```
mkdir -p $WM_PROJECT_USER_DIR/SRC/myBCs/myFiniteVolume/fields/fvPatchFields/derived
cd $WM_PROJECT_USER_DIR/SRC/myBCs/myFiniteVolume/fields/fvPatchFields/derived
cp -r $FOAM_SRC/finiteVolume/fields/fvPatchFields/derived/oscillatingFixedValue
localTranslatingFixedValue
```

Rename all the files inside `localTranslatingFixedValue` folder. `localTranslatingFixedValue` is the name of the new boundary condition.

```
cd localTranslatingFixedValue
mv oscillatingFixedValueFvPatchField.H localTranslatingFixedValueFvPatchField.H
mv oscillatingFixedValueFvPatchField.C localTranslatingFixedValueFvPatchField.C
mv oscillatingFixedValueFvPatchFields.H localTranslatingFixedValueFvPatchFields.H
mv oscillatingFixedValueFvPatchField.C localTranslatingFixedValueFvPatchFields.C
mv oscillatingFixedValueFvPatchFieldFwd.H localTranslatingFixedValueFvPatchFieldFwd.H
```

Replace all the "oscillating" string in all the files with "localTranslating".

```
sed -i s/oscillating/localTranslating/g localTranslatingFixedValueFvPatchField*
```

Start with `localTranslatingFixedValueFvPatchField.H` and modify the private data as

```
        // Private data

        //- Values of the field at the dropInlet
        Field<Type> dropInletValue_;

        //- Values of the field outside the dropInlet
        Field<Type> dropOutsideValue_;

        //- Initial location of the Centre of the droplet
        List<vector> dropCentre0_;

        //- Centre of the droplet after time t
        List<vector> dropCentre_;

        //- Radii of the droplet
        List<scalar> dropRadius_;

        //Face centers
        Field<vector> faceCentres_;

        //- Current time index
        label curTimeIndex_;

        //- Velocity of the droplet along the horizantal base metal
        List<vector> dropTranslatingVelocity_;

        //- Frequency of the falling droplet
        scalar dropFrequency_;

/*********************End of private data***********************/
```

Comment or delete the the declared private member function.

```
        // Private Member Functions
        //- Return current scale
        //scalar currentScale() const;

/******************* End of private member function *************/
```

Replace the member functions with

```
        // Member functions
        // Access

        //- Return the dropInlet value
        const Field<Type>& dropInletValue() const
        {
```

```
    return dropInletValue_;
}

//Return reference to the dropInlet value to allow adjustment
Field<Type>& dropInletValue()
{
        return dropInletValue_;
}

//- Return the dropOutside value
const Field<Type>& dropOutsideValue() const
{
        return dropOutsideValue_;
}

//- Return reference to the dropOutside value to allow adjustment
Field<Type>& dropOutsideValue()
{
        return dropOutsideValue_;
}


//-
List<vector> dropCentre0() const
{
        return dropCentre0_;
}
List<vector>& dropCentre0()
{
        return dropCentre0_;
}

//-
List<vector> dropCentre() const
{
        return dropCentre_;
}
List<vector>& dropCentre()
{
        return dropCentre_;
}

//-
List<scalar> dropRadius() const
{
        return dropRadius_;
}
List<scalar>& dropRadius()
{
        return dropRadius_;
}
```

```
        //-
        Field<vector> faceCentres() const
        {
                return faceCentres_;
        }
        Field<vector>& faceCentres()
        {
                return faceCentres_;
        }


        //- Translating velocity of the droplet
        Field<vector> dropTranslatingVelocity() const
        {
                return dropTranslatingVelocity_;
        }
        Field<vector>& dropTranslatingVelocity()
        {
                return dropTranslatingVelocity_;
        }

        //- Return dropfrequency
        scalar& dropFrequency()
        {
                return dropFrequency_;
        }

/****************** End of member function *************/
```

Save and close `localTranslatingFixedValueFvPatchField.H` file.

Open `localTranslatingFixedValueFvPatchField.C` file and do the following modifications.

Comment or delete the private member function with the definition of `currentScale()` function.

Modify the constructors to

```
template<class Type>
localTranslatingFixedValueFvPatchField<Type>::localTranslatingFixedValueFvPatchField
(
    const fvPatch& p,
    const DimensionedField<Type, volMesh>& iF
)
:
    fixedValueFvPatchField<Type>(p, iF),
    dropInletValue_(p.size()),
    dropOutsideValue_(p.size()),
    dropCentre0_(p.size()),
    dropCentre_(p.size()),
```

```
    dropRadius_(),
    faceCentres_(p.Cf()),
    dropTranslatingVelocity_(),
    dropFrequency_()

{}

template<class Type>
localTranslatingFixedValueFvPatchField<Type>::localTranslatingFixedValueFvPatchField
(
    const localTranslatingFixedValueFvPatchField<Type>& ptf,
    const fvPatch& p,
    const DimensionedField<Type, volMesh>& iF,
    const fvPatchFieldMapper& mapper
)
:
    fixedValueFvPatchField<Type>(ptf, p, iF, mapper),
    dropInletValue_(ptf.dropInletValue_, mapper),
    dropOutsideValue_(ptf.dropOutsideValue_, mapper)
{}


template<class Type>
localTranslatingFixedValueFvPatchField<Type>::localTranslatingFixedValueFvPatchField
(
    const fvPatch& p,
    const DimensionedField<Type, volMesh>& iF,
    const dictionary& dict
)
:
    fixedValueFvPatchField<Type>(p, iF),
    dropInletValue_("dropInletValue", dict, p.size()),
    dropOutsideValue_("dropOutsideValue", dict, p.size()),
    dropCentre0_(dict.lookup("dropCentre0")),
    dropCentre_(dict.lookup("dropCentre")),
    dropRadius_(dict.lookup("dropRadius")),
    faceCentres_(p.Cf()),
    dropTranslatingVelocity_(dict.lookup("dropTranslatingVelocity")),
    dropFrequency_(readScalar(dict.lookup("dropFrequency")))
{
    if (dict.found("value"))
    {
        fixedValueFvPatchField<Type>::operator==
        (
            Field<Type>("value", dict, p.size())
        );
    }
    else
    {
        fvPatchField<Type>::operator==(dropOutsideValue_);
        Field<Type>& patchfield = *this;
        forAll(p.Cf(),i)  //go through all the cells in the current patch
```

```
        {
            bool isDrop=false;//isInlet could be a better name.
            forAll(dropRadius_,j)//go through all the holes
            {
                if(mag(p.Cf()[i]-dropCentre_[j])<=dropRadius_[j]){isDrop=true;}
                {
                    if(isDrop){patchfield.data()[i]= dropInletValue_[i];}
                }
            }
        }
    }
}

template<class Type>
localTranslatingFixedValueFvPatchField<Type>::localTranslatingFixedValueFvPatchField
(
    const localTranslatingFixedValueFvPatchField<Type>& ptf
)
:
    fixedValueFvPatchField<Type>(ptf),
    dropInletValue_(ptf.dropInletValue_),
    dropOutsideValue_(ptf.dropOutsideValue_),
    dropCentre0_(ptf.dropCentre0_),
    dropCentre_(ptf.dropCentre_),
    dropRadius_(ptf.dropRadius_),
    dropTranslatingVelocity_(ptf.dropTranslatingVelocity_),
    dropFrequency_(ptf.dropFrequency_)
{}


template<class Type>
localTranslatingFixedValueFvPatchField<Type>::localTranslatingFixedValueFvPatchField
(
    const localTranslatingFixedValueFvPatchField<Type>& ptf,
    const DimensionedField<Type, volMesh>& iF
)
:
    fixedValueFvPatchField<Type>(ptf, iF),
    dropInletValue_(ptf.dropInletValue_),
    dropOutsideValue_(ptf.dropOutsideValue_),
    dropCentre0_(ptf.dropCentre0_),
    dropCentre_(ptf.dropCentre_),
    dropRadius_(ptf.dropRadius_),
    dropTranslatingVelocity_(ptf.dropTranslatingVelocity_),
    dropFrequency_(ptf.dropFrequency_)
{}
// * * * * * * * * * * * * * * *End of Constructors  * * * * * * * * * * * * * //
```

Modify the Member Functions to

```
template<class Type>
void localTranslatingFixedValueFvPatchField<Type>::autoMap
(
    const fvPatchFieldMapper& m
)
{
    fixedValueFvPatchField<Type>::autoMap(m);
    dropInletValue_.autoMap(m);
    dropOutsideValue_.autoMap(m);
    faceCentres_.autoMap(m);
}


template<class Type>
void localTranslatingFixedValueFvPatchField<Type>::rmap
(
    const fvPatchField<Type>& ptf,
    const labelList& addr
)
{
    fixedValueFvPatchField<Type>::rmap(ptf, addr);

    const localTranslatingFixedValueFvPatchField<Type>& tiptf =
    refCast<const localTranslatingFixedValueFvPatchField<Type> >(ptf);
    dropInletValue_.rmap(tiptf.dropInletValue_, addr);
    dropOutsideValue_.rmap(tiptf.dropOutsideValue_, addr);
    faceCentres_.rmap(tiptf.faceCentres_, addr);
}

template<class Type>
void localTranslatingFixedValueFvPatchField<Type>::updateCoeffs()
{
    if (this->updated())
    {
        return;
    }

    scalar t_ = this->db().time().value();        //indirect access to the runTime.
    float tinj_ = 1/(2*dropFrequency_);           //Droplet injection time.
    const int ninj = floor(t_/tinj_);             //Number of droplet injection

    if (curTimeIndex_ != this->db().time().timeIndex())
    {
        fvPatchField<Type>::operator==(dropInletValue_);
        Field<Type>& patchfield = *this;
        forAll(this->faceCentres(),i)
        {
            bool isDrop=false;
            forAll(dropRadius_,j)
            {
                if(mag(this->faceCentres()[i]-dropCentre_[j])<=dropRadius_[j]
               && (t_ > 0.5 && ninj % 2 == 0)){isDrop=true;}
```

```
            }
                if(!isDrop){patchfield.data()[i]= dropOutsideValue_[i];}
        }
        curTimeIndex_ = this->db().time().timeIndex();
        dropCentre_   = dropCentre0_ + dropTranslatingVelocity_ * t_;
        Info << "curTimeIndex_" << curTimeIndex_ << endl;
    }
    fixedValueFvPatchField<Type>::updateCoeffs();
}


template<class Type>
void localTranslatingFixedValueFvPatchField<Type>::write(Ostream& os) const
{
    fixedValueFvPatchField<Type>::write(os);
    dropInletValue_.writeEntry("dropInletValue", os);
    dropOutsideValue_.writeEntry("dropOutsideValue", os);
    os.writeKeyword("dropCentre0")
    << dropCentre0_ << token::END_STATEMENT << nl;
    os.writeKeyword("dropCentre")
    << dropCentre_ << token::END_STATEMENT << nl;
    os.writeKeyword("dropRadius")
    << dropRadius_ << token::END_STATEMENT << nl;
    os.writeKeyword("dropTranslatingVelocity")
    << dropTranslatingVelocity_ << token::END_STATEMENT << nl;
    os.writeKeyword("dropFrequency")
    << dropFrequency_ << token::END_STATEMENT << nl;
}
// * * * * * * * * * * * * *End of Member Functions * * * * * * * * * * * //
```

Create a Make directory by copying from the `$FOAM\_SRC/finiteVolume/Make`

```
cd $WM_PROJECT_USER_DIR/SRC/myBCs
cp -r $FOAM_SRC/finiteVolume/Make
```

Remove all the lines from the original *Make/files* and replace it with

```
fvPatchFields = myFiniteVolume/fields/fvPatchFields
derivedFvPatchFields = $(fvPatchFields)/derived
$(derivedFvPatchFields)/localTranslatingFixedValue/localTranslatingFixedValueFvPatchFields.C
LIB = $(FOAM_USER_LIBBIN)/libmyBCs
```

Save and close.
Remove all the lines in *Make/options* file and add

```
EXE_INC = \
    -I$(LIB_SRC)/triSurface/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude
LIB_LIBS = \
    -lOpenFOAM \
    -ltriSurface \
    -lmeshTools \
    -lfiniteVolume
```

Save and close.

Clean and compile the library

```
wclean libso
wmake libso
```

The two solvers `interThermalFoam` and `interThermalReferenceFoam`, and the boundary condition are ready to be applied for simulation. Chapter 6 describes setting up two test cases and modeling relative motion from two different approaches.

Chapter 6

Setting up a test case

6.1 Test Case 1

The next step is to set up two new test cases by copying an existing test case of *damBreak* tutorial. Initially, the vertices and the blocks are adjusted to modify the computational domain to 3D. The two phases in *transportProperties* dictionary is modified from air and water phase to argon and steel phase respectively. The corresponding value of transport properties is updated accordingly. The thermophyscial properties of steel and argon is defined in the new dictionaries named *thermophysicalPropertiesPhase1* and *thermophysicalPropertiesPhase2* respectively. The properties of heat source is defined in a new dictionary named *beamProperties* in the *constant* directory. The initial conditions and boundary conditions in *0* directory is changed as needed.

```
run
cp -r $FOAM_TUTORIALS/multiphase/interFoam/laminar/damBreak movingLaser
cd movingLaser
```

The first test case is named *movingLaser*. This test case is solved with *interThermalFoam* solver. The test case is in the fixed coordinate system where the heat source and drop injector moves in the computational domain. Since the problem is symmetric along the horizontal x-axis, only one half of the computational domain is solved.

Since the main purpose of this tutorial is to model relative motion of heat source and a liquid pool from two different approaches, the major modifications in the test case is to define laser beam properties in a new dictionary named *beamProperties* in *constant* directory and to implement the new BC for velocity field *U* and phase fraction field *alpha.steel.org* in *0* directory are given here. The rest of the case files are available for download and not explained in detailed here.

6.1.1 The *beamProperties* dictionary

The properties of a laser beam heat source which is explicitly coupled with the energy conservation equation is defined in a new dictionary called *beamProperties*. Create a new file named *beamProperties* in *constant* directory and add

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      beamProperties;
}
```

```
// initial location of the laser beam source center
   Xbeam0    Xbeam0 [0 1 0 0 0 0 0] (0.01   0.0   0.0);
// time from which the laser is switched on
   timeStartLaser    timeStartLaser [0 0 1 0 0 0 0] 0.0;
// radius of the laser beam
   rBeam    rBeam [0 1 0 0 0 0 0] 1.4e-3;
// absorptivity of the sample surface to the laser
   eta_L      eta_L [0 0 0 0 0 0 0] 0.13;
// laser beam power
   Q_L        Q_L [1 2 -3 0 0 0 0] 3850;
// Laser beam welding speed
   Ulaser      Ulaser [0 1 -1 0 0 0 0] (0.01 0 0); // moving laser beam with 0.01 m/s
```

6.1.2 The *0/U* file

The new BC developed in Chapter 5 is implemented in *dropInlet* patch for velocity field. With this BC, the users can define the droplet impinging velocity, radius, frequency and initial location. The velocity of the droplet moving parallel to liquid pool given by *dropTranslatingVelocity* is the same as the velocity of the heat source.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      U;
}

dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{
    dropletInlet
    {
        type                            localTranslatingFixedValue;
        value                           uniform (0 0 0);
        dropInletValue                  uniform (0.0 -1.0 0);
        dropOutsideValue                uniform (0 0 0);
        dropCentre0                     ((0.01 0.01 0.0));
        dropCentre                      ((0.01 0.01 0.0));
        dropRadius                      (0.0006);
        dropTranslatingVelocity         ((0.01 0 0));
        dropFrequency                   167;
    }
    bottomSample
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    leftAndRightSample
    {
        type            fixedValue;
        value           uniform (0 0 0);
```

```
    }
    backSample
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    topAtmosphere
    {
        type            pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }
    leftAndRightAtmosphere
    {
        type            pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }
    backAtmosphere
    {
        type            pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }
    symmetryBc
    {
        type            symmetryPlane;
    }
}
```

6.1.3 The *0/alpha.steel.org* file

Similarly, the new BC is also implemented in *dropInlet* patch for alpha field to model periodically falling droplets.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      alpha.steel;
}
dimensions      [0 0 0 0 0 0 0];
internalField   uniform 0;
boundaryField
{
    dropletInlet
    {
        type                        localTranslatingFixedValue;
        value                       uniform 0;
        dropInletValue              uniform 1.0;
        dropOutsideValue            uniform 0;
        dropCentre0                 ((0.01 0.01 0.0));
        dropCentre                  ((0.01 0.01 0.0));
        dropRadius                  (0.0006);
        dropTranslatingVelocity     ((0.01 0 0));
        dropFrequency               167;
```

```
    }
    bottomSample
    {
        type            fixedValue;
        value           uniform 1;
    }
    leftAndRightSample
    {
        type            zeroGradient;

    }
    backSample
    {
        type            zeroGradient;
    }
    topAtmosphere
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }
    leftAndRightAtmosphere
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }
    backAtmosphere
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }
    symmetryBc
    {
        type            symmetryPlane;
    }
}
```

Now everything is set to run the first test case. Start by copying the original `alpha.steel.org` files beefore running the `setFields` utility

```
cp 0/alpha.steel.org 0/alpha.steel
```

Generate mesh and run setFields.

```
blockMesh
setFields
```

Figure 6.1 shows the computational domain, mesh and the initial fields. Run the first solver, i.e. interThermalFoam, and write output to a log file.

```
interThermalFoam >& log.tIterFoam &
```

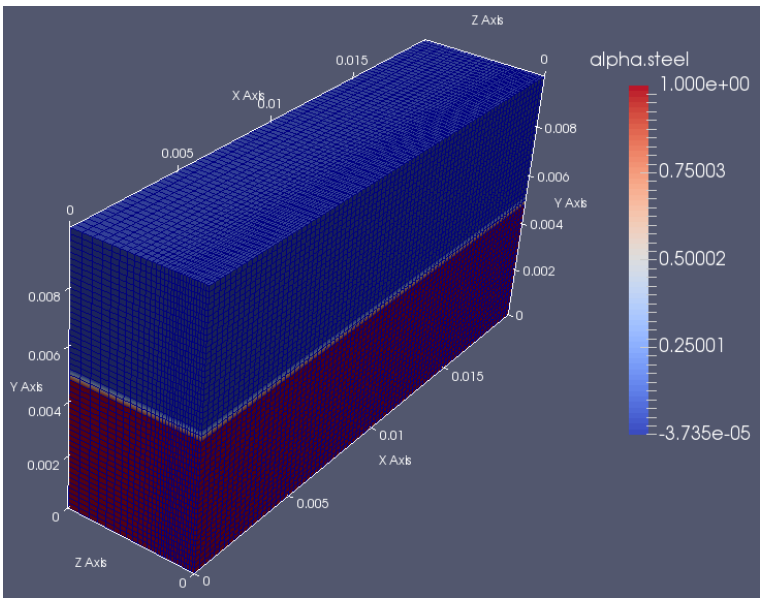

Figure 6.1: Computational domain with initial fields

6.2 Test Case 2

The second test case is called *movingFrame*. The test case is run using the second solver i.e. `interThermalReferenceFoam`. Start by copying test case 1.

```
run
cp -r movingLaser movingFrame
cd movingFrame
```

In this test case, the laser beam stays stationary at its initial location which is implemented in `interThermalReferenceFoam` solver. However unlike in test case 1, the location of the formation of droplets also remains at the same location. This is handled by the boundary condition of *alpha.steel* and *U* by setting *dropTranslatingVelocity* to (0, 0, 0). Open *alpha.steel.org* file and under the boundary condition for *dropInlet* patch inside boundaryField, change *dropTranslatingVelocty* to

```
dropTranslatingVelocity ((0 0 0));
```

Save and close the file.

Do the same as above in *0/U* file as well.

```
dropTranslatingVelocity ((0 0 0));
```

Save and close the file. Everything else remains the same for test case 2 and it is ready to be run. Start by copying the original alpha.steel.org files before turrning the *setFields* utility

```
cp 0/alpha.steel.org 0/alpha.steel
```

Generate mesh and run setFields

```
blockMesh
setFeids
```

Run the second solver, i.e. interThermalReferenceFoam, and write output to a log file.

```
interThermalReferenceFoam >& log.interThermalReferenceFoam &
```

Chapter 7

Results

7.1 Temperature distribution

Figure 7.1 shows a comparison of temperature distribution on the surface of liquid steel at 0.5s between *movingLaser* and *movingFrame* test case. The temperature distribution on the steel surface agrees very well between the two cases. Similarly Figure 7.2 shows a comparison of temperature distribution along symmetrical plane between two test cases. The temperature distributions agree very well with minor variation in argon gas phase. The white horizontal line in Figure 7.2 represents the interface between steel and argon gas phase.

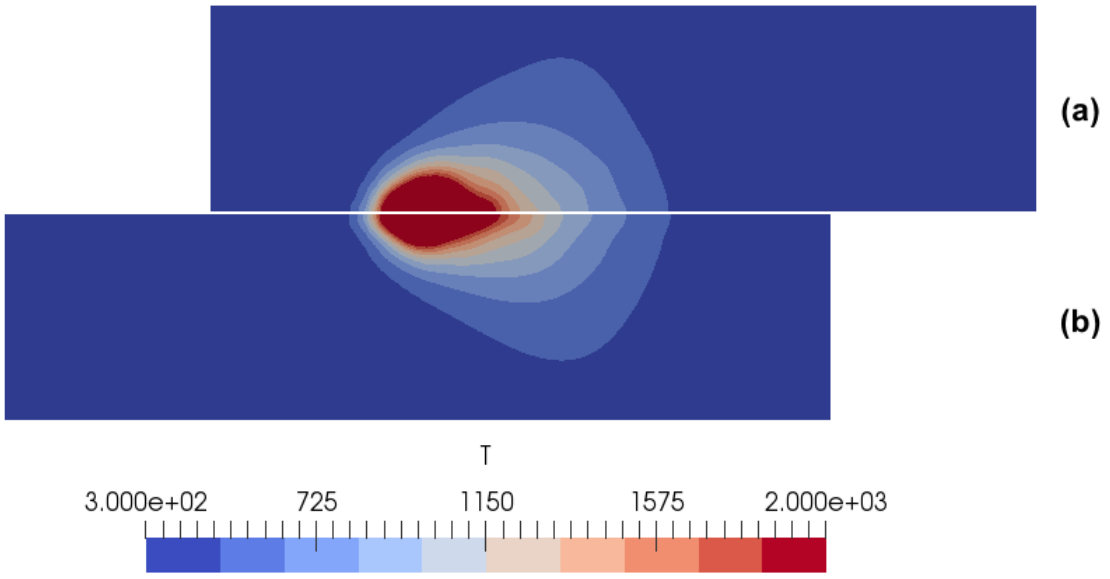

Figure 7.1: Temperature distribution on top metal surface at 0.5s (a) Moving Laser (b) Moving Frame.

7.2 Surface deformation

Figure 7.3 shows comparison of surface deformation on the liquid steel surface produced by the impact of the droplet. The surface deformation between the two test cases agrees very well.

Furthermore, the simulation time for a test case in moving reference frame is 11.25% lower than for a test case with moving heat source using AMD Opteron(tm) Processor 6386 SE. For cases with

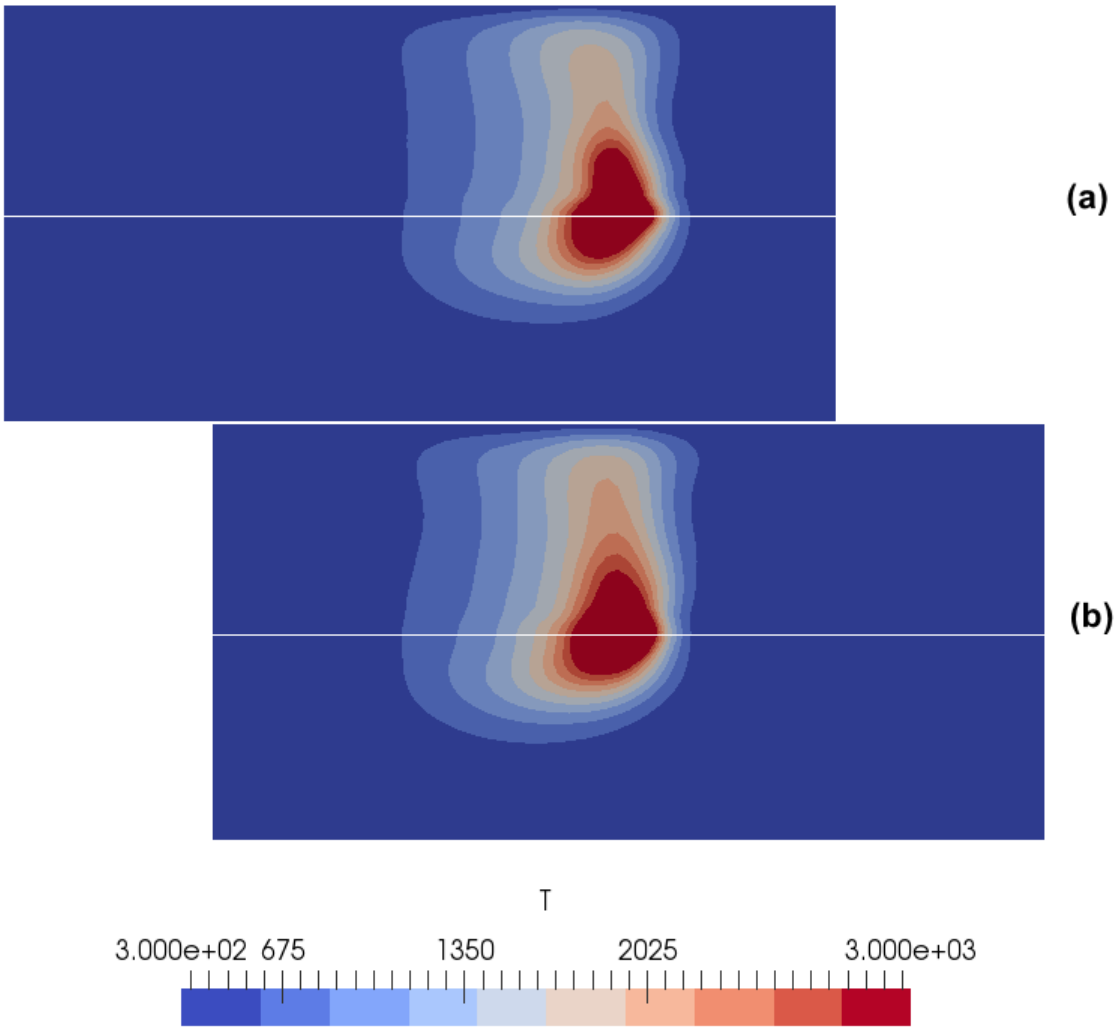

Figure 7.2: Temperature distribution along symmetrical plane at 0.5s (a) Moving Laser (b) Moving Frame.

longer simulation time, the solver modified for moving reference frame provide significant time saving with fairly identical results.

From the analysis of the results obtained from the test cases with moving heat source and moving reference frame, we can draw the conclusion that the implementation of governing equations in moving reference frame are fairly accurate to model relative motion between two bodies.

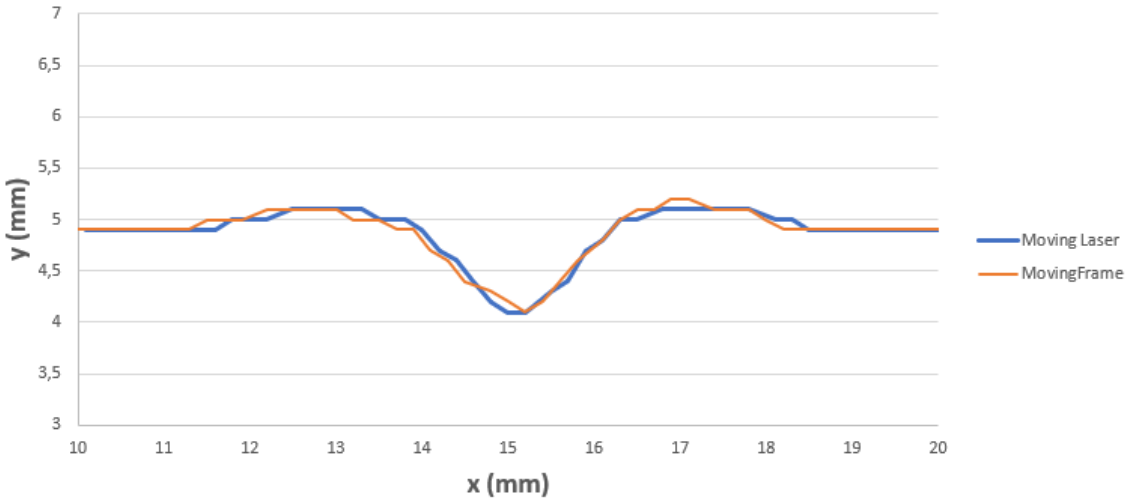

Figure 7.3: Deformation on liquid steel due to droplet impact at 0.52s.

Chapter 8

Future work

Both of the the solver discussed in the report can be improved by considering other phenomenon occurring in similar engineering applications. For example including Marangoni convection and phase change described in [12] would allow the use of the modified solver for welding simulation. Additionally, for similar multiphysics simulations with relative motion between two bodies, governing equations in moving reference frame can be similarly derived and used in simulations to reduce computational time.

Study questions

1. What is the most widely used distribution model for the heat source term in welding application?
2. What is factor? Why is it used in multiphase flow simulation with high density ratio?
3. How to modify governing equations to a moving reference frame and implement in OpenFOAM?
4. Why is it important to construct and declare the fields of the variables in source term in a separate file?
5. Why is it important to model motion in moving reference frame?
6. How to formulate new BC form an existing BC?
7. What member data do you need to declare to formulate BC for periodically falling droplets.

Bibliography

[1] Junling Hu, H Guo, and Hai-Lung Tsai. Weld pool dynamics and the formation of ripples in 3d gas metal arc welding. *International Journal of Heat and Mass Transfer*, 51(9-10):2537–2552, 2008.

[2] Xiangman Zhou, Haiou Zhang, Guilan Wang, and Xingwang Bai. Three-dimensional numerical simulation of arc and metal transport in arc welding based additive manufacturing. *International Journal of Heat and Mass Transfer*, 103:521–537, 2016.

[3] Akash Aggarwal and Arvind Kumar. Particle scale modelling of selective laser melting-based additive manufacturing process using open-source cfd code openfoam. *Transactions of the Indian Institute of Metals*, 71(11):2813–2817, 2018.

[4] Vaibhav K Arghode, Arvind Kumar, Suresh Sundarraj, and Pradip Dutta. Computational modeling of gmaw process for joining dissimilar aluminum alloys. *Numerical Heat Transfer, Part A: Applications*, 53(4):432–455, 2008.

[5] Renzhi Hu, Xin Chen, Guang Yang, Shuili Gong, and Shengyong Pang. Metal transfer in wire feeding-based electron beam 3d printing: Modes, dynamics, and transition criterion. *International Journal of Heat and Mass Transfer*, 126:877–887, 2018.

[6] S Márquez Damian. Description and utilization of interfoam multiphase solver. *URL: http://infofich. unl. edu. ar/upload/3be0e16065026527477b4b948c4caa7523c8ea52. pdf*, 2012.

[7] Mickael Courtois, Muriel Carin, Philippe Le Masson, Sadok Gaied, and Mikhaël Balabane. A complete model of keyhole and melt pool dynamics to analyze instabilities and collapse during laser welding. 2014.

[8] Pei-quan Xu, Chen-ming Bao, Feng-gui Lu, Chun-wei Ma, Jian-ping He, Hai-chao Cui, and Shang-lei Yang. Numerical simulation of laser–tungsten inert arc deep penetration welding between wc–co cemented carbide and invar alloys. *The International Journal of Advanced Manufacturing Technology*, 53(9-12):1049–1062, 2011.

[9] John Goldak, Aditya Chakravarti, and Malcolm Bibby. A new finite element model for welding heat sources. *Metallurgical transactions B*, 15(2):299–305, 1984.

[10] Jeremiah U Brackbill, Douglas B Kothe, and Charles Zemach. A continuum method for modeling surface tension. *Journal of computational physics*, 100(2):335–354, 1992.

[11] Frank M White. Fluid mechanics. 2015.

[12] AD Brent, Vaughan R Voller, and KTJ Reid. Enthalpy-porosity technique for modeling convection-diffusion phase change: application to the melting of a pure metal. *Numerical Heat Transfer, Part A Applications*, 13(3):297–318, 1988.