

CFD WITH OPENSOURCE SOFTWARE

A COURSE AT CHALMERS UNIVERSITY OF TECHNOLOGY
TAUGHT BY HÅKAN NILSSON

Project work:

Solving electric field using Maxwell's equations and compressibleInterFoam solver

OpenFOAM-2.2.x

Author:
Josefine SVENUNGSSON

Peer reviewed by:
MINGHAO WU
HÅKAN NILSSON

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 to learn some details similar to the ones presented in the report and in the accompanying files. The tutorial is developed for OpenFOAM-2.2.x but can be implemented also for OpenFOAM-2.4.x. 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.

February 4, 2016

Introduction

The aim of this project is to model the interaction of a laser beam with a metal, while modelling the beam light as an electromagnetic wave. As the beam material interaction will ultimately result in metal melting and the formation of a hole filled with gas (called a keyhole) in the liquid metal, a multiphase solver is needed. As the problem is temperature dependent and the material properties (e.g. density, thermal conductivity and specific heat) will later on depend on temperature the solver is chosen compressible. The project is thus based on the multifluid solver `compressibleInterFoam`. This solver is presented in chapter 1. The analytical background is based on the governing equations for electromagnetic theory. A set of partial differential equations, namely Maxwell's equations, are used to find an expression for the electric field. The electric field generates a source term at the gas/metal interface since the light wave is partially reflected and partially absorbed when it hits the opaque media (the metal). This analytical background is presented in the first part of chapter 2. The remaining part of chapter 2 is devoted to the tutorial developed in this project. This tutorial describes the implementation of a solver for two-phase compressible fluid flow with heat transfer including a heat source represented by an electric field. The existing `compressibleInterFoam` solver is used as a basis and modified to include the laser heat input at the metal gas interface.

- The partial differential equation (derived from Maxwell's equations) governing the electric field is implemented.
- The electric field is used to calculate the energy source term resulting from energy deposition from a laser beam. This source term is rescaled to reproduce the power input of the laser beam.
- The laser beam energy source term enters the energy conservation equation formulated here with the temperature.
- A new boundary condition is implemented.
- A new test case is set up to validate the implementation. It illustrates a simplification of a gas filled cavity surrounded by molten metal.

All work presented here is done in `OpenFOAM-2.2.x`, and should also be able to compile with `OpenFOAM-2.4.x`.

The developed solver is capable of handling a heat source of incoming light beam of different wave length, power and efficiency. The mesh size should however not exceed the laser wave length to obtain a good resolution of the electric field. The developed tutorial can be used for different materials as long as the assumptions stated in this tutorial are valid.

The reader should learn how to modify an existing solver and handling boundary conditions for new equations. A new boundary condition is implemented to handle the initial electric field on the top surface as a Gaussian distribution.

Learning outcomes

The aim of this report is that the reader should gain a deeper understanding of the compressibleInterFoam solver and learn how to modify it, add an equation for electric field, implement a new source term coupling the energy equation to the electric field, and a new boundary condition for electric field.

Specific learning outcomes are:

- how to use the compressibleInterFoam solver
- how to modify the compressibleInterFoam solver
- how to add an equation for electric field calculations
- how to add a source term for energy equation with rescaling to control the beam power
- how to implement a new BC for the electric field
- how to set up a new test case

Chapter 1

Description of the OpenFOAM solver compressibleInterFoam

A description of the compressibleInterFoam solver and the accompanying test case is given here.

The solver is located in `$FOAM_SOLVERS/multiphase/compressibleInterFoam`.

The compressibleInterFoam solver is a solver for two compressible non-isothermal and immiscible fluids using the Volume of Fluid, VOF, method for interface capturing. The thermodynamic and transport properties as viscosity, density and specific heat are of mixture type. They are derived from properties of the mixture components, the last one being assumed constant. The solver includes equations for the mixture energy, momentum, pressure (based on the continuity equation) and for the volume fraction alpha of one fluid in the mixture.

Having a look into the structure of the compressibleInterFoam solver directory we find the main source file, compressibleInterFoam.C, and header files createFields.H, readControls.H, UEqn.H, TEqn.H, pEqn.H, alphaEqns.H and alphaEqnsSubCycle.H. A detailed description of these files is given in section 1.1.

1.1 Governing equations

The governing equations solved in the compressibleInterFoam solver are the continuity, momentum and energy equations

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U) = 0 \quad (1.1)$$

$$\frac{\partial(\rho U)}{\partial t} + \nabla \cdot (\rho U U) = -\nabla p + \nabla \cdot (\mu \nabla U) + S_U \quad (1.2)$$

$$\frac{\partial(\rho C_p T)}{\partial t} + \nabla \cdot (\rho U C_p T) = \nabla \cdot (k \nabla T) + S_T \quad (1.3)$$

$$\frac{\partial \gamma}{\partial t} + \nabla \cdot (\gamma U) + \nabla \cdot ((1 - \gamma) \gamma U_r) = 0 \quad (1.4)$$

where ρ is the density, U the velocity vector, t is the time, p is the pressure, μ is the dynamic viscosity, C_p is the specific heat, T temperature and k thermal conductivity. S_U and S_P are the momentum and energy source terms respectively. All these fields are associated with the mixture, contrary to equation (4). Equation (4) is the partial differential equation provided by the Volume of Fluid model for governing the volume fraction γ of one fluid in the mixture. In VOF γ is defined as 1 in cells with only one phase, 0 in cells with only the other phase and between 0 and 1 in interface cells containing both of the two phases. The γ is the volume fraction solved, (in OpenFOAM γ is represented by α).

Equations (1.2) and (1.3) are written in OpenFOAM as below.

Momentum equation (UEqn.H)

```
fvVectorMatrix UEqn
(
  fvm::ddt(rho, U)
  +fvm::div(rhoPhi, U)
  +turbulence->divDevRhoReff(U)
);
```

Energy equation (TEqn.H)

```
fvScalarMatrix TEqn
(
  fvm::ddt(rho, T)
  +fvm::div(rhoPhi, T)
  -fvm::laplacian(twoPhaseProperties.alphaEff(turbulence->mut()), T)
  +(
    fvc::div(fvc::absolut(phi, U), p)
    +fvc::ddt(rho, K) + fvc::div(rhoPhi, K)
  )
  *(
    alpha1/twoPhaseProperties.thermo1().Cv()
    +alpha2/twoPhaseProperties.thermo2().Cv()
  )
);
```

As mentioned previously the compressibleInterFoam solver uses the Volume of Fluid interface capturing method to represent the transition region between the liquid and gas. Volume of Fluid is more diffusive but conserves mass better compared to Level Set method. The Volume of Fluid model implemented in OpenFOAM is a version improved for reducing diffusivity, namely the Sharper Surface Force VOF developed by [5]. Volume fraction α is calculated in the `alphaEqn.H` file and implemented in OpenFOAM in the twoPhaseMixture model which calculates a weighted average based on the volume fraction.

The volume fraction α is calculated (in `alphaEqn.H`) as

```
forAll(dgdt, celli)
{
  if (dgdt[celli] > 0.0 && alpha1[celli] > 0.0)
  {
    Sp[celli] -= dgdt[celli]*alpha1[celli];
    Su[celli] += dgdt[celli]*alpha1[celli];
  }
  else if (dgdt[celli] < 0.0 && alpha1[celli] < 1.0)
  {
    Sp[celli] += dgdt[celli]*(1.0 - alpha1[celli]);
  }
}
```

Heat transfer across the interface of the two fluids is accounted for by the equation of state which couple pressure, temperature and density. `compressibleInterFoam` solver uses the perfect fluid equation of state.

A mixture model is used for the interface cells (cells with $0 < \alpha < 1$) to calculate weighted values of for example density and specific heat by the volume fraction.

We will start investigating the solver by having a look into the source file, `compressibleInterFoam.C`. Open the `compressibleInterFoam.C` file and have a look.

```
\*-----*/
```



```

}
runTime.write();

Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << "s"
  << " ClockTime = " << runTime.elapsedClockTime() << "s"
  << nl <<endl;
}
Info<< "End\n" << endl;
return 0;
}

```

*****//

The code starts with several include files which define the frame for the simulation including solver, mesh generation and definition/declaration of scalars, vectors and tensors. The `while (runTime.run())` command initiates the time loop and calculates the new time step based on the Courant Number which is also calculated in the loop.

The `#include alphaEqnsSubCycle.H` initiate calculation of alpha and include also the `alphaEqns.H` file which calculates the phase fraction for each sub-cycle.

`#include UEqn.H` initiates the solving of momentum equation and `#include TEqn.H` the energy equation.

The pressure correction loop calculates the pressure and corrects the turbulence value. The time step finishes with writing out information about time and residuals. A new time step is initiated and so it continues until a converged solution is found, or the maximum number of iterations is reached.

1.2 compressibleInterFoam tutorial depthCharge2D

OpenFOAM provides tutorials for all the solvers in `$FOAM_TUTORIALS`. Tutorials directory can also be reached by using the environment variable `tut`. Have a look in the tutorials directory to find a tutorial for `compressibleInterFoam` solver to learn more about how the solver works.

Initilize the OpenFOAM environment using command

```
OF22x
```

Have a look in the directory

```
ls $FOAM_TUTORIALS/multiphase/compressibleInterFoam/laminar
```

there are two test cases;

`depthCharge2D` and `depthCharge3D`.

We will look at the `depthCharge2D` test case.

Copy the test case to an appropriate directory (`$WM_PROJECT_USER_DIR` for example).

Change directory to where you want to copy the test case to:

```
cd $FOAM_RUN
```

Copy the test case and change directory

```
cp -r $FOAM_TUTORIALS/multiphase/compressibleInterFoam/laminar/depthCharge2D .
```

```
cd depthCharge2D
```

The test case folder consist of `0-`, `constant-` and `systems-` directories. There is also a `Allrun`-script which can be used for running the simulation. We will not use the script now, instead go step-by-step through the simulation.

Start by having a look into the `constant`-directory.

```
ls constant
```

We see that this directory consist of a subdirectory `polymesh` and files for specific properties, such as `thermophysicalProperties`, `thermophysicalPropertiesair`, `thermophysicalPropertieswater` and `turbulenceProperties`.

Open the `polymesh` directory to find the dictionary for mesh generation, `blockMeshDict`.

```
ls constant/polymesh
```

`blockMeshDict` specifies the geometry and the mesh. Now generate the mesh using the `blockMesh` utility by typing

```
blockMesh
```

Use post-processor `paraView` to visualize the geometry and mesh, type

```
paraFoam
```

The `systems`-directory contains dictionaries for the simulation settings. Have a look in to the `systems`-directory

```
ls system
```

We find `controlDict`, `fvSchemes`, `fvSolution`, `setFieldsDict` dictionaries.

`controlDict` consist of settings for running the simulation, `fvSchemes` dictionary consist of definitions for the discretization for each term, `fvSolution` consist of all settings concerning solving of matrices and pressure-velocity coupling and `setFieldsDict` contains information for assigning specific field properties to certain parts of the domain.

In this test case the domain is rectangular and partially filled with water and air. The type `sphereToCell` is used to assign a spherical bubble with air in the water of the domain. Before initializing `setFields` the original files containing initial conditions in `0`-directory has to be copied. Copy the original files

```
cp 0/alphawater.org 0/alphawater
cp 0/T.org 0/T
cp 0/p.org 0/p
cp 0/p_rgh.org 0/p_rgh
```

When running `setFields` the utility reads the field values from the files in the time directory and overwrite them with new values according to settings in the `setFieldsDict` dictionary. Due to this it is wise to always have a copy of the original files in `0`-directory named `.org`.

Now to run `setFields` utility in `case`-directory write

```
setFields
```

The domain contains water and air. A spherical bubble of air is immersed in the water, see figure 1. The air bubble has an initially higher pressure and temperature than the surrounding water.

Before running the test case have a look at the run settings in `system/controlDict`.

```
vi system/controlDict
```

`controlDict` dictionary controls for example the simulation run time, time step as well as frequency for writing output. Let's keep this dictionary unchanged. Close the `controlDict` file.

Run the case in background and write a log-file by typing

```
compressibleInterFoam >&log&
```

View the results in `paraView`

```
paraFoam
```

`paraView` is the tool for pre- and post-processing and it is used to display results with different plot-functions or by animation over time steps of specific properties as temperature, pressure, velocity and distribution of for example water and air.

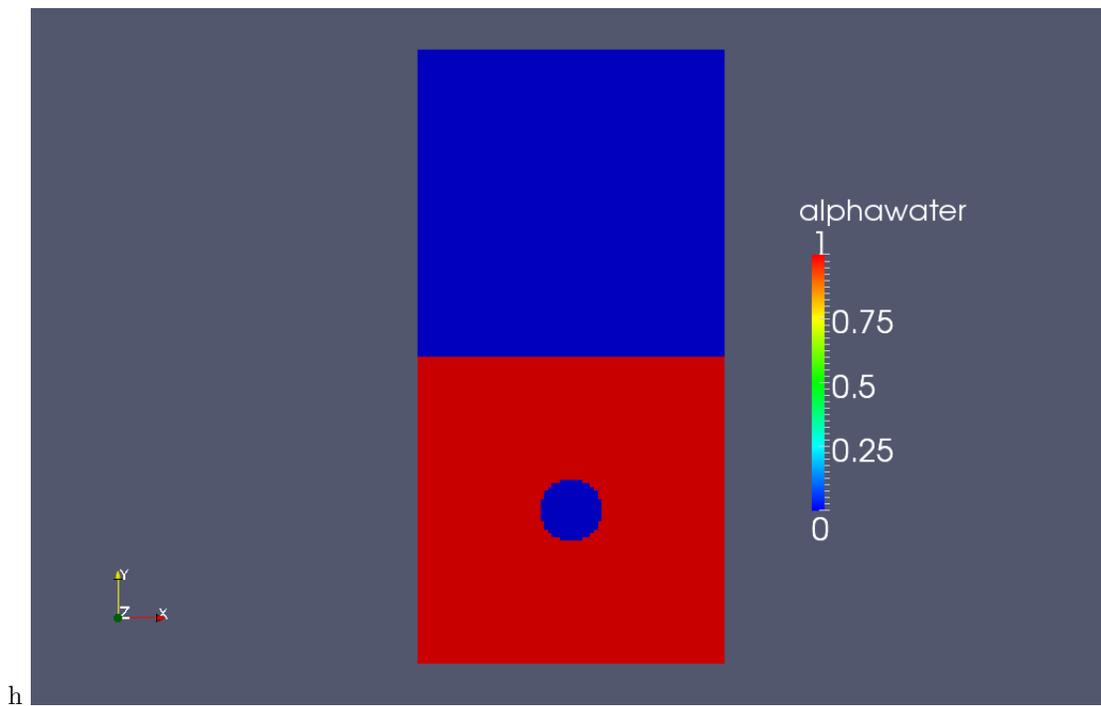


Figure 1.1: Intial field

Chapter 2

Solver for electric field

The modified compressibleInterFoam solver is called compressibleInterFoamEikonalSolver and is presented here. It can be used for simulating an incoming laser beam represented by an electric field.

Now when we are familiar with the compressibleInterFoam solver it is time to start to modify the solver in order to implement the new solver. In the following sections the analytical solution of wave equations and motivation for using wave equations to represent a laser beam is given.

2.2 Theory behind the modification of the solver

The modification to be made in the solver include adding an equation for solving the electric field. The electric field is coupled explicitly to the energy equation through a source term. The theory behind the equation for electric field is given in the following sections.

A new boundary condition, BC, handling the initial electric field is implemented, see section 4.8, and a test case is set up to test the new solver. The test case is further described in chapter 5 Test case.

2.3 Representation of laser beam heat source

Laser beam heat source can interact with a non-transparent material in two ways called diffusion interaction and keyhole interaction depending on the laser beam power (thus its ability to penetrate into the material). This project handles keyhole interaction. The keyhole is a hole filled with metal vapor and formed in the melted metal due to the high vapor pressure built up as a result of intense metal vaporization. The most common method for modelling an incoming laser beam in the keyhole interaction regime is the ray tracing method. The laser beam is then discretized in a finite number of rays and each ray carries energy of the incident laser. As the ray travels into the keyhole it will be reflected several times and at each reflection a part of the incoming energy is absorbed by the irradiated surface. The amount of energy absorbed locally by the keyhole surface depends on the laser beam wavelength, polarization and the angle of incidence of the incoming electric field with respect to the keyhole surface.

A different approach is to use wave equations to calculate the energy deposition inside the domain. The energy is then represented by an equation for electric field originating from Maxwell's equations. This is the approach used in this work.

Section 4.4 further explains the theory behind using an electric field to represent a laser beam with a Gaussian distribution.

2.4 Gaussian beam

Gaussian beams can be used as a solution to wave equations of electromagnetic field in free space or in a homogeneous medium. They are thus useful for setting the boundary conditions for the electric field describing the laser beam. A Gaussian beam is a TEM (Transverse ElectroMagnetic) mode with a mathematical expression for the electric field from Helmholtz equation. A transverse mode is the pattern of an electromagnetic field perpendicular to the axis of propagation of the laser light. TEM₀₀ is a circular pattern.

The parameters needed for characterising a Gaussian beam are wave length dependent, for example beam width, $w(z)$, the beam divergence and the Rayleigh range, z_R depends on wave length.

Figure 2 show a schematic image of a Gaussian beam with description of important parameters. w_0 is minimum waist, $w(z)$ is the width and the beam propagates along the z -direction.

The Rayleigh range is the distance from the beam waist to where the spot size has increased by a factor of $\sqrt{2}$. For any given wave length, λ , the shape of the Gaussian beam is determined from the minimum waist. When the wave length and polarisation of the beam are known, the minimum waist of the beam, w_0 , determines the amplitude of the electric and magnetic field, [7], [9]. For a given wave length the beam divergence is inversely proportional to the spot size and the beam diverges rapidly when focussed to a small spot size [9].

In order to use a Gaussian beam model a relation between the minimum waist and the wave length of the beam has to be fulfilled, $w_0 > 2\lambda/\pi$.

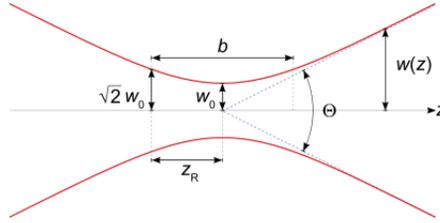


Figure 2.1: Schematic image of Gaussian beam [9]

2.5 Governing equations - Maxwell's equations

In this work energy deposition by a laser beam onto a substrate is calculated from Maxwell's equations. An electric field can be described using electromagnetic theory of optics, or Maxwell's equations, with the set of equations

$$\nabla \cdot \bar{D} = \rho \quad (2.1)$$

$$\nabla \cdot \bar{B} = 0 \quad (2.2)$$

$$\nabla \times \bar{E} = -\frac{\partial \bar{B}}{\partial t} \quad (2.3)$$

$$\nabla \times \bar{H} = \bar{J} + \frac{\partial \bar{D}}{\partial t} \quad (2.4)$$

$$\bar{B} = \mu \bar{H} \quad (2.5)$$

$$\bar{D} = \epsilon \bar{E} \quad (2.6)$$

where D is the electric displacement, ρ is the volume charge density, B the magnetic field, E the electric field, H magnetic field intensity and J current density, [6]. μ is the permeability and ϵ is the permittivity.

Combining equation (4.2) and (4.7) for constant ϵ gives:

$$\nabla \bar{E} = 0 \quad (2.7)$$

Equation (4.5), (4.6) and (4.7) give:

$$\nabla \times \bar{B} = \mu\sigma\bar{E} + \mu\epsilon\frac{\partial\bar{E}}{\partial t} \quad (2.8)$$

Assuming harmonic wave ¹ propagation the electric field is given by :

$$\bar{E}(\bar{r}, t) = \bar{E}(\bar{r})e^{i\omega t} \quad (2.9)$$

By inserting equation (4.10) in to (4.9) time can be simplified:

$$\nabla^2\bar{E}(\bar{r}) + \omega^2\mu(\epsilon - \frac{i\sigma}{\omega})\bar{E}(\bar{r}) = 0 \quad (2.10)$$

The equations can be further simplified by assuming that the media (gas) is

1. a uniform homogeneous media
2. an isotropic media
3. a non-conducting media
4. a media free of charge
5. a non-dispersive media

Assumption 4, a media free of charge, is valid if the Debye length, λ_D , is short enough compared to the wavelength of the light. The assumption can be validated by calculating the Debye length

$$\lambda_D = \sqrt{\frac{\epsilon_0 T_e k_B}{n_e q_e}} \quad (2.11)$$

where ϵ_0 is permittivity of vacuum, T_e is electron temperature, k_B is Boltzman constant, n_e is electron density and q_e is the electric charge. This is the simplified equation where the ion term is dropped.

From the harmonic time dependent equation for electromagnetic field the total electric energy density can be defined as:

$$W = 2w_e = \frac{\epsilon}{2}\bar{E}^2 \quad (2.12)$$

The quadratic term $\mathbf{E} \cdot \mathbf{E}$ originates from Maxwell's equations and the poynting vector [6] and describes the intensity of the flow of energy. The poynting vector product is

$$S(r, t) = \frac{1}{T} \int [E(r, t) \times H(r, t)] dt \quad (2.13)$$

Further details of the derivations can be found in ...XXX.

2.6 Equations to be solved for the electric field

The equation to be solved in order to include the electric field is Eq.(2.10). The equation is further simplified by assuming a non-conductive media so that the σ -term disappears, giving:

$$\nabla^2\bar{E} + k_0^2 \cdot \epsilon_r \cdot \mu_r \cdot \bar{E} = 0 \quad (2.14)$$

where k_0 wave number of free space, ϵ_r the relative permittivity and μ_r the relative permeability, [3].

¹A harmonic wave is a simple periodic wave which is a plane wave at the focal point.

Equation (4.14) is the equation that should be solved together with the source term for the total energy input

$$W = 2 \cdot \bar{E}^2 \cdot \epsilon \quad (2.15)$$

To represent the incoming beam the electric field at the laser input boundary of the computational domain has to be calculated, see section 4.8. The distribution of the field is Gaussian in the direction normal to the beam propagation axis.

2.7 How to modify the compressibleInterFoam solver

Now the compressibleInterFoam solver should be modified to a solver including calculation of the electric field. We will do it step by step.

Start by creating a directory for the user defined solvers and copying the existing compressibleInterFoam solver to that directory and change the name of the solver and .C file

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

We need now to copy the file alphaCourantNo.H from the interFoam solver to the user's solvers directory

```
cp $FOAM_APP/solvers/multiphase/interFoam/alphaCourantNo.H .
```

Now when the files are copied and name is changed update the Make/files file and compile. (Open the file with an editor of your choice.)

```
vi Make/files
```

Replace the code with the following two lines

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

The Make/options file can be kept unchanged.

Now clean and compile

```
wclean
wmake
```

For a successful compilation this should generate an output message saying something like

```
~/chalmers/users/.../platforms/linux64GccDPOpt/bin/compressibleInterFoamE' is up to date
```

Define and declare new fields

Now it is time to define and declare the new fields needed for the calculation. We need a vector field E , two scalar fields S_{laser} , for the energy input, and Q_{laser} for the heat source, and a dimensioned scalar $sumSlaserVol$ for the total energy input over the domain.

Create a new file called createEmgFields.H add

```
IOdictionary emgProperties
(
    IOobject
    (
        "emgProperties",
        runtime.constant(),
        mesh,
        IOobject::MUST_READ,
        IOobject::NO_WRITE
    )
)
```

```

);
// laser power
dimensionedScalar laserPower
(
    emgProperties.lookup("laserPower")
);
// laser Efficiency
dimensionedScalar laserEfficiency
(
    emgProperties.lookup("laserEfficiency")
);
// effective laser power
dimensionedScalar EffLaserPower = laserPower*laserEfficiency;
dimensionedScalar muVac
(
    emgProp("muVac")
);
dimensionedScalar muR
(
    emgProp("muR")
);
dimensionedScalar eps
(
    emgProperties.lookup("eps")
);
dimensionedScalar epsR
(
    emgProperties.lookup("epsR")
);
dimensionedScalar lambda
(
    emgProp("lambda")
);
// laser beam electric field
Info<< "Reading field E\n" << endl;
volVectorField E
(
    IOobject
    (
        "E",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
// speed of light in free space
dimensionedScalar c0
(
    "c0",

```

```

    dimensionSet(0, 1, -1, 0, 0, 0, 0),
    299792458
);
// calculate k0 - wave number of free space to be used in EEqn.H
// in this case relative permeability & permittivity = 1
dimensionedScalar k0
(
    "k0",
    dimensionSet(0, -1, 0, 0, 0, 0, 0),
    2.0*constant::mathematical::pi/lambda.value()
);
// local laser energy
Info << "create Slaser \n" << endl;
volScalarField Slaser
(
    IOobject
    (
        "Slaser",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
    ),
    mesh,
    dimensionSet(1, -1, -2, 0, 0, 0, 0) // as E&E*eps
);
Info << " initialize sumSlaserVol " << endl;
dimensionedScalar sumSlaserVol
(
    "sumSlaserVol",
    dimensionSet(1, 2, -2, 0, 0, 0, 0), // as E&E*eps*Vol
    0
);
// rate of heat input from laser beam (source term for TEqn)
Info << "create Qlaser \n" << endl;
volScalarField Qlaser
(
    IOobject
    (
        "Qlaser",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    mesh,
    dimensionSet ( 1, -1, -3, 0, 0, 0, 0) // as EffLaserPower/mesh.V()
);
//Info << "initialization of Qlaser done \n" << endl;
All new fields are created in a separate file in order to make the code easier to read.

```

Equation governing the electric field

Now close and save the `createEmgFields.H` file and create a `EEqn.H` file. Add the following lines to `EEqn.H`

```
{
    solve
    (
        fvm::laplacian(E) + Foam::sqr(k0)*(1.0-alpha1)*E
    );
}
```

Save and close.

The source term

Now create a file for the source term calculation called `emgSourceTerm.H`

```
// Electro-magnetic source terms needed for the fluid equations
// local laser energy
Slaser= 0.5*(E&E)*eps*Foam::neg(-alpha1);
// Need to sum Slaser * mesh.V over all the cells to scale QsourceLaser
// problem 1:
// for some reason the dimension of Slaser is "lost" when summin in the
// following loop

scalar adimSumSlaserVol = 0.0;
forAll(mesh.C(), celli)
{
    adimSumSlaserVol += Slaser[celli]*mesh.V()[celli];
}
sumSlaserVol.value() = adimSumSlaserVol;
// rate of heat input from the laser
Qlaser = (Slaser/sumSlaserVol)*EffLaserPower;
```

Save and close the file.

Couple the electric field to the energy equation

So far we have defined (and declared) all new fields, added an equation for `E` and the source term. Now we need to couple the electric field `E` to the energy equation. This is done by explicitly coupling the source term `Qlaser` to the `TEqn.H`.

Open the `TEqn.H` file and add the source term under `TEqn.solve()`;

```
solve
(
    TEqn == Qlaser
    *(
        alpha1/twoPhaseProperties.thermo1().Cv()
        + alpha2/twoPhaseProperties.thermo2().Cv()
    )
);
```

The final step is to add the new files to the source file `compressibleInterFoamEikonal.C`. Open the file and after `#include createFields.H` add

```
// declare the emg-related fields and parameters
```

```
#include "createEmgFields.H"
Then between UEqn.H and TEqn.H add
    #include "EEqn.H" // Electric field of laser beam
    Info << " EEqn solved " << endl;
    #include "emgSourceTerm.H" // Heat source produced by laser
    Info << " emgSourceTerm solved " << endl;
Save and close. Compile the code
wmake
```

2.8 How to implement a new boundary condition for the electric field

After having modified the solver a boundary condition for the electric field E is needed. In order to represent the incoming laser beam the initial electric field has to be calculated from the equation of electric field with a Gaussian distribution

$$E_r = \tau * E_0 e^{-\left(\frac{r_b}{w(z)}\right)^2} \times \cos(kw * z_f + kw * \left(\frac{r_b}{2R_{z_f}}\right) - \psi) \quad (2.16)$$

where E_0 is the amplitude of the electric field, r_b the distance from the beam centre to the cell centre, $w(z)$ the width of the beam, kw the wave number, z_f the distance from the beam centre on the patch to the focal point, R_{z_f} the evolving radius of curvature of the wave front and ψ the Gouy phase.

In OpenFOAM boundary conditions are modelled and represented in a hierarchical structure of classes. The top abstract class is the `fvPatchField` class. Now a new boundary condition handling this equation has to be implemented. This will be done by modifying the `parabolicVelocity` BC which is provided through `OpenFOAM-extend`.

Start by creating a new directory and download the `parabolicVelocity` boundary condition to the user-directory for sources, `$WM_PROJECT_USER_DIR/src`.

```
cd $WM_PROJECT_USER_DIR/src/
mkdir -p $WM_PROJECT_USER_DIR/src/myFiniteVolume/fields/fvPatchFields/derived/
svn checkout svn://svn.code.sf.net/p/openfoam-extend/svn/trunk/Core/OpenFOAM-1.5-dev/src/\
finiteVolume/fields/fvPatchFields/derived/parabolicVelocity
cd $WM_PROJECT_USER_DIR/src/myFiniteVolume/fields/fvPatchFields/derived
mv parabolicVelocity gaussianElectric
cd gaussianElectric
```

Now change the name of the `.C` and `.H` files.

```
mv parabolicVelocityFvPatchVectorField.C gaussianElectricFvPatchVectorField.C
mv parabolicVelocityFvPatchVectorField.H gaussianElectricFvPatchVectorField.H
```

Use the `sed` command to change `parabolicVelocityFvPatchVectorField` to `gaussianElectricFvPatchVectorField` everywhere in the `.C` and `.H` files

```
sed -i s/"parabolicVelocityFvPatchVectorField"/"gaussianElectricFvPatchField"/\
gaussianElectricFvPatchField.*
```

Create a `Make` directory by copying from the `$FOAM_SRC/finiteVolume/Make` and change the path and name of executable to be compiled in the `Make/files` file

```
cp -r $FOAM_SRC/finiteVolume/Make $FOAM_USER_SRC/myFiniteVolume/fields
vi ../../../../Make/files
```

Remove the existing lines and paste the following lines

```
fvPatchFields = fields/fvPatchFields
derivedFvPatchFields = $(fvPatchFields)/derived
```

```
$(derivedFvPatchFields)/gaussianElectric/gaussianElectricFvPatchVectorField.C
LIB = $(FOAM_USER_LIBBIN)/libmyFiniteVolume
```

Save and close the file.

Create a Make/options file

```
vi Make/options
```

Add the following

```
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 the file.

Clean and compile the library

```
wclean libso
```

```
wmake libso
```

Now start to modify the boundary condition in order to implement the equation (4.17).

Open `gaussianelectricFvPatchVectorField.C` and under constructors remove the existing constructors and add new constructors under

```
gaussianElectricFvPatchVectorField::gaussianElectricFvPatchVectorField
(
  const fvPatch& p,
  const DimensionedField<vector, volMesh>& iF
)
:
```

Add

```
  fixedValueFvPatchVectorField(p, iF),
  n_(0.0, 1.0, 0.0),      //unit vector normal to patch
  tau_(1.0, 0.0, 0.0),   //unit vector along the 1D patch
  lambda_(0),            // laser beam wavelength
  omega0_(0),           // minimum beam radius w0
  EO_(0),               // amplitude of electric field at the focal point
  focalP_(0.0, 0.0, 0.0) // location of focal point
```

Then add the following member functions

```
// * * * * * Member Functions * * * * *
```

```
void gaussianElectricFvPatchVectorField::updateCoeffs()
{
  if (updated())
  {
    return;
  }
  // wave number k = 2*pi/lambda
  const scalar kw = 2.0*constant::mathematical::pi/lambda_;
  //Info << " kw = " << kw << endl;
  // Raleigh range
  const scalar zRa = constant::mathematical::pi*Foam::sqr(omega0_)/lambda_;
```

```

//Info << " zRa = " << zRa << endl;
    // cell centre of patch cell
const vectorField& Ccf = patch().Cf();
    // distance from beam center to cell centre
const scalarField rb = Foam::mag((Ccf - focalP_)&tau_);
//Info << " rb = " << rb << endl;

// distance from beam centre on patch to focal point
// IMPORTANT: Need to adapt if the beam is not normal to the patch
const scalar zf = Foam::mag((focalP_-Ccf[0])&n_);
Info << " zf = " << zf << endl;

// Particular case (to avoid division by zero in the focal plane)
// in the focal plane :
//     omegaz=omega0    beam waist
//     Rzf = infinity  evolving radius of curvature of wave front
//     psi = 1         Gouy phase
// thus
vectorField Etau = tau_*EO_*Foam::exp(-(Foam::sqr(rb/omega0_)));

// If the patch is not in the focal plane
if (zf > 1e-12)
{
    // radius of curvature of wave front at the distance zf from focal point
    const scalar Rzf = zf*(1.0+(Foam::sqr(zRa/zf)));
    //Info << " Rzf = " << Rzf << endl;
    // Gouy phase
    const scalar psi = Foam::atan(zf/zRa);
    //Info << " psi = " << psi << endl;
    // beam width at distance zf from focal plane
    const scalar omegaZ = omega0_*Foam::sqrt(1.0+Foam::sqr(zf/zRa));
    //Info << " omega0 = " << omega0_ << " omegaZ = " << omegaZ << endl;
    //Etau = tau_*EO_*Foam::exp(-(Foam::sqr(rb/omegaZ)));
    Etau = tau_*EO_*Foam::exp(-(Foam::sqr(rb/omegaZ)))
        *Foam::cos( omega0_ - kw*zf - kw*Foam::sqr(rb/(2.0*Rzf)) + psi );
    //     *Foam::cos( kw*zf + kw*Foam::sqr(rb/(2.0*Rzf)) - psi );
}

vectorField::operator=(Etau);
fixedValueFvPatchVectorField::updateCoeffs();
}

```

Under //Write in the end of the file add

```

os.writeKeyword("n")
    << n_ << token::END_STATEMENT << nl;
os.writeKeyword("tau")
    << tau_ << token::END_STATEMENT << nl;
os.writeKeyword("lambda")
    << lambda_ << token::END_STATEMENT << nl;
os.writeKeyword("omega0")
    << omega0_ << token::END_STATEMENT << nl;
os.writeKeyword("EO")
    << EO_ << token::END_STATEMENT << nl;
os.writeKeyword("focalP")

```

```
<< focalP_ << token::END_STATEMENT << nl;
```

And finally change the last command line to

```
makePatchTypeField(fvPatchVectorField, gaussianElectricFvPatchVectorField);
```

Save and close the file.

Now open the `gaussianElectricFvPatchField.H` file and change the private data to the following:

```
// Private data
//unit vector normal to patch
vector n_;
//unit vector along the 1D patch
vector tau_;
//- laser beam wavelength
scalar lambda_;
//- min waist of beam ( = waist on focal plane)
scalar omega0_;
//- amplitude of electric field
scalar E0_;
// -Laser spot center
vector focalP_;
```

The private data is the input parameters that are controlled in the case 0-directory when running a simulation with the boundary condition.

Further down in the file change the member functions

```
//- Return unit vector normal to patch
vector& n()
{
    return n_;
}
//- Return unit vector along 1D patch
vector& tau()
{
    return tau_;
}
//- Return the laser beam wavelength
scalar& lambda()
{
    return lambda_;
}
//- Return min waist
scalar& omega0()
{
    return omega0_;
}
//- Return amplitude of electric field
scalar& E0()
{
    return E0_;
}
//- Return the location of the focal point
vector& focalP()
{
```

```
        return focalP_;  
    }
```

Save and close the file.

Clean and compile the library using `wclean` and `wmake` command.

```
wclean libso  
wmake libso
```

Now the solver and the boundary condition are implemented and can be used for simulation. Chapter 3 describes how to set up a new test case to test the new solver and boundary condition.

Chapter 3

Test Case

Usually the OpenFOAM test cases are set up in the run directory and consists of a constant, a system and a time directory. The constant directory is used for geometry description including the mesh. The geometry is defined in the subdirectory polymesh.

All information needed to define the numerical solution schemes and the simulation run settings is set in system-directory. A dictionary for setFields can be specified in the systems directory to apply specific features for parts of the calculation domain. This utility is used for example when the VOF-method is applied.

The time directories are created during run and are named by numbers representing the calculation time steps. A 0-directory is always needed in order to initialize the field values and the boundary conditions.

The test case used here is a 2D domain with liquid and gas, steel and air respectively. Initial fields for gas/liquid are set using `setFields` utility with the option `rotatedBoxToCell` in `system/setFieldsDict`.

3.1 Geometry and mesh

To set up a new test case copy the existing test case for the original compressibleInterFoam solver `depthCharge2D` and change the geometry and mesh.

```
run
cp -r $FOAM_TUTORIALS/multiphase/compressibleInterFoam/laminar/depthCharge2D .
cd depthCharge2D/constant/polymesh
```

The test case consist of a 2D geometry of size $0,08 \times 0,06 \times 0,001$ m. The domain is divided into six blocks and the mesh is generated by the `blockMesh` utility. The mesh used is relatively coarse but can easily be changed to a finer mesh by the user in the `blocks` definition.

Open the `blockMeshDict` file and add vertices and blocks

```
convertToMeters 0.001;

vertices
(
    (0.2 0 0) // vertice 0
    (0.5 0 0) // 1
    (0.3 0.3 0) // 2
    (0 0.3 0) // 3
    (0.2 0 0.01) // 4
    (0.5 0 0.01) // 5
    (0.3 0.3 0.01) // 6
    (0 0.3 0.01) // 7
    (0.2 0.6 0) // 8
    (0.5 0.6 0) // 9
```

```

(0.5 0.6 0.01) // 10
(0.2 0.6 0.01) // 11
(0.6 0 0)      // 12
(0.4 0.3 0)   // 13
(0.6 0 0.01)  // 14
(0.4 0.3 0.01) // 15
(0.6 0.6 0)   // 16
(0.6 0.6 0.01) // 17
(0.9 0 0)     // 18
(0.7 0.3 0)   // 19
(0.9 0 0.01)  // 20
(0.7 0.3 0.01) // 21
(0.9 0.6 0)   // 22
(0.9 0.6 0.01) // 23
);
blocks
(
  hex (0 1 2 3 4 5 6 7) (30 30 1) simpleGrading (1 1 1)
  hex (3 2 9 8 7 6 10 11) (30 30 1) simpleGrading (1 1 1)
  hex (1 12 13 2 5 14 15 6) (10 30 1) simpleGrading (1 1 1)
  hex (2 13 16 9 6 15 17 10) (10 30 1) simpleGrading (1 1 1)
  hex (12 18 19 13 14 20 21 15) (30 30 1) simpleGrading (1 1 1)
  hex (13 19 22 16 15 21 23 17) (30 30 1) simpleGrading (1 1 1)
);

```

Then under boundary specify the faces for each of the boundary patches

```

top
{
  type patch;          //inlet;
  faces
  (
    (8 9 10 11)        // top surface block 1
    (9 16 17 10)       // top surface block 3
    (16 22 23 17)      // top surface block 5
  );
}
bottom
{
  type patch;          //outlet;
  faces
  (
    (5 1 0 4)          // bottom surface block 0
    (14 12 1 5)        // bottom surface block 2
    (20 18 12 14)      // bottom surface block 4
  );
}
left
{
  type patch;
  faces
  (
    (0 3 7 4)
    (3 8 11 7)
  );
}

```

```

    );
}
right
{
    type patch;
    faces
    (
        (21 23 22 19)
        (20 21 19 18)
    );
}
frontAndBack
{
    type empty;
    faces
    (
        (0 1 2 3)          // front
        (3 2 9 8)
        (1 12 13 2)
        (2 13 16 9)
        (12 18 19 13)
        (13 19 22 16)
        (4 7 6 5) // back
        (7 11 10 6)
        (5 6 15 14)
        (6 10 17 15)
        (14 15 21 20)
        (15 17 23 21)
    );
}

```

Save and close the file.

The geometry and mesh can now be generated using OpenFOAM utility `blockMesh`.

Change directory to case top directory and run `blockMesh`.

```
cd ../../
```

```
blockMesh
```

The geometry and mesh is seen in figure 1.

Open the `setFieldsDict` which is located in the systems directory.

```
vi system/setFieldsDict
```

Then under `regions` add the following lines to assign gas properties to the 2 blocks representing the cavity. The option `rotatedBoxToCell` has to be used since the geometry consists of blocks rotated 30 degrees around the z-axis.

```

(
    rotatedBoxToCell
    {
        origin (0.5e-3 0 0);
        i (0.1e-3 0.0 0);
        j (-0.2e-3 0.3e-3 0);
        k (0.0 0.0 0.01e-3);

        fieldValues
        (
            volScalarFieldValue alphasteel 0

```

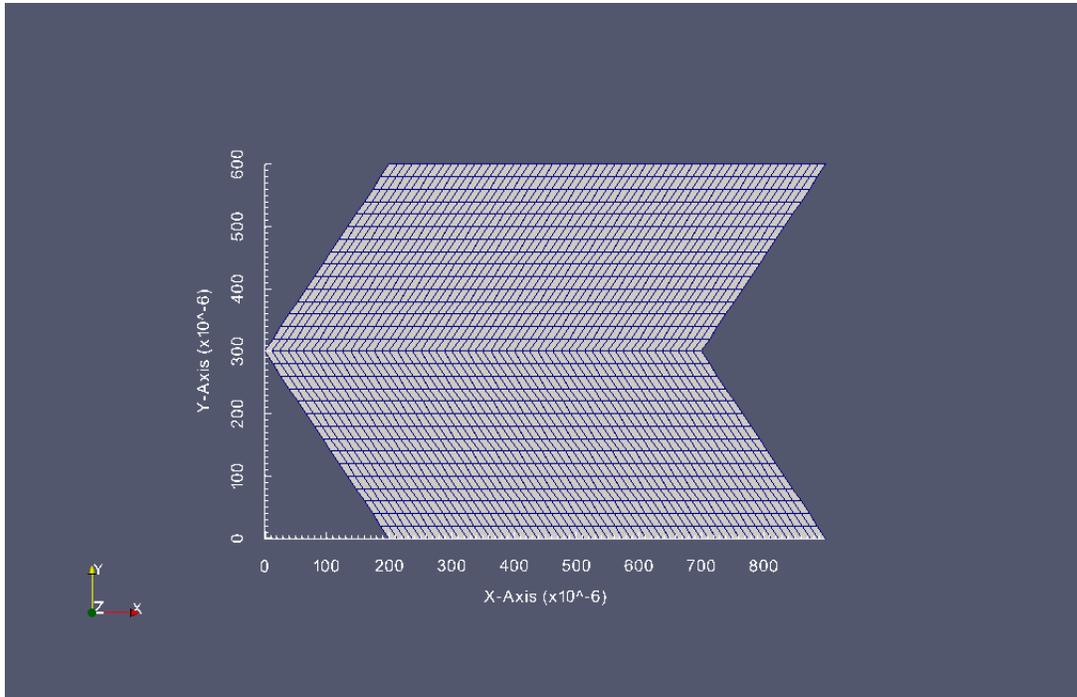


Figure 3.1: Geometry and mesh of test case

```

    );
}
rotatedBoxToCell
{
    origin (0.3e-3 0.3e-3 0);
    i (0.1e-3 0.0 0);
    j (0.2e-3 0.3e-3 0);
    k (0.00 0.00 0.01e-3);
    fieldValue
    (
        volScalarFieldValue alphasteel 0
    );
}
);

```

A dictionary for volume fraction alpha needs to be created. Change the `alphawater.org` to `alphasteel.org` and add the following lines under `dimensions` `[0 0 0 0 0 0 0]`;

```

internalField    uniform 1;
boundaryField
{
    top
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value            uniform 0;
    }
    bottom
    {

```

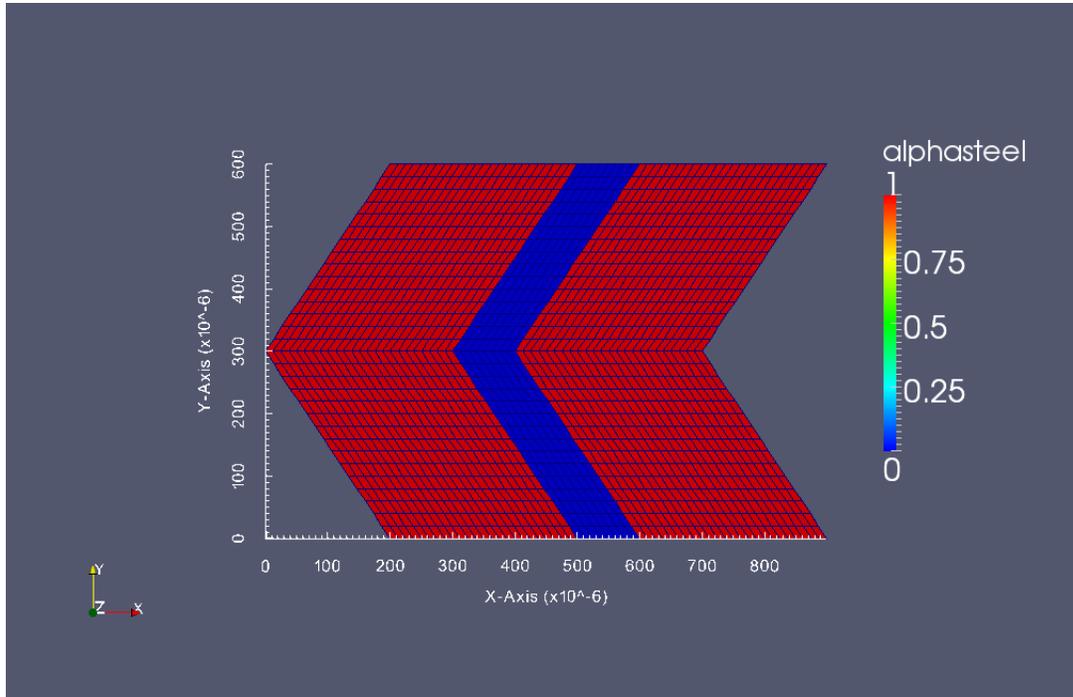


Figure 3.2: Initial fields

```

        type      outletInlet;
        outletValue uniform 0;
        value     uniform 0;
    }
    left
    {
        type      zeroGradient;
    }
    right
    {
        type      zeroGradient;
    }
    frontAndBack
    {
        type      empty;
    }
}

```

The domain is initially filled with liquid and gas, through `setFields` utility. Run the utility by typing

```
setFields
```

Figure 4 shows the geometry after `setFields`. The reason for the angled geometry is to include masking effects which appear when the incident laser beam is reflected on the metal preventing the beam to reach deeper into the substrate.

3.2 Assumptions

In order to simplify the calculations some assumptions are made in the test case:

- alpha is a constant $\alpha = 0$ or 1
- ρ is a constant
- gravitation $g = 0$
- surface tension $\sigma = 0$

In order to use the wave equation together with the Navier-Stokes equation a "large" wavelength has to be used. The wavelength of the light is compared to the Debye length in order to verify the assumption of a media free of charge. (With free of charge it means no local charge in a cell.) The wave length is specified by the user through the boundary condition.

3.3 Boundary and initial conditions

Boundary conditions and initial conditions are specified in the case 0-directory. Table 1 summarizes the boundary conditions to be used in this test case. The width of the domain is considered to be large enough, compared to the gas filled cavity, in order to use a fixedValue condition on left and right side.

Patch	E	T	U	p
top	gaussianElectric	zeroGradient	inletOutlet	zeroGradient
bottom	zeroGradient	zeroGradient	zeroGradient	zeroGradient
left	fixedValue	fixedValue	fixedValue	zeroGradient
right	fixedValue	fixedValue	fixedValue	zeroGradient
frontAndBack	empty	empty	empty	empty

Table 3.1: Boundary conditions

How to set initial and boundary conditions for alphasteel is described in previous section, and will not be described here again. To set the initial and boundary conditions for E, T, U, p and p_rgh start with creating a file for E in 0-directory

vi 0/E

Add the following lines

```
FoamFile
{
    version      2.0;
    format       ascii;
    class        volVectorField;
    object       E;
}
// * * * * *
dimensions      [1 1 -3 0 0 -1 0];
internalField   uniform (0.0 0.0 0.0);
boundaryField
{
    top
    {
        type            gaussianElectric;
        n                (0 1 0); // normal to patch (assumed to be also the laser beam direction)
        tau              (1 0 0); // direction along the 1D boundary line
        lambda           1.0e-3; // laser beam wave length
        omega0           5e-4;  // laser beam waist in focal plane
    }
}
```

```

    E0          1.0e+1; // electric field intensity on beam axis in focal plane
    focalP     (5.5e-4 6.0e-4 5.0e-6); // location of focal point
    value      uniform (0.0 0.0 0.0); // needed but not used
}
bottom
{
    type      fixedValue;
    value     uniform (0.0 0.0 0.0);
}
left
{
    type      fixedValue;
    value     uniform (0.0 0.0 0.0);
}
right
{
    type      fixedValue;
    value     uniform (0.0 0.0 0.0);
}
frontAndBack
{
    type      empty;
}
defaultFaces
{
    type      empty;
}
}
}
// ***** //
Save and close the file.

```

We also need to change the file for temperature T. Open the file and add the `boundaryFields`

```

top
{
    type      zeroGradient;
}
bottom
{
    type      zeroGradient;
}
left
{
    type      zeroGradient;
}
right
{
    type      zeroGradient;
}
frontAndBack
{
    type      empty;
}

```

```

}
defaultFaces
{
    type          empty;
}

```

For p and p_rgh change in one file and copy to the other file since they are the same. Open the p.org file and set the `internalField` and `boundaryField` to

```

internalField  uniform 1e5;
boundaryField
{
    top
    {
        type          zeroGradient;
    }
    bottom
    {
        type          zeroGradient;
    }
    left
    {
        type          zeroGradient;
    }
    right
    {
        type          zeroGradient;
    }
    frontAndBack
    {
        type          empty;
    }
    defaultFaces
    {
        type          empty;
    }
}

```

Then copy p.org to p_rgh.org

```
cp 0/p.org 0/p_rgh.org
```

Finally open the file for U and change the `boundaryField`

```
vi 0/U
```

```

boundaryField
{
    top
    {
        type          inletOutlet;
        inletValue    uniform (0 0 0);
        value         uniform (0 0 0);
    }
    bottom
    {
        type          zeroGradient;
    }
}

```

```

}
left
{
    type          fixedValue;
    value         uniform (0 0 0);
}
right
{
    type          fixedValue;
    value         uniform (0 0 0);
}
frontAndBack
{
    type          empty;
}

```

To specify input parameters for the boundary condition and for the heat source we need set up a file called `emgProperties` where the previously implemented boundary condition `gaussianElectric` will read its input. Create the file and add

```

// * * * * * //
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    location     "constant";
    object       emgProperties;
}
// * * * * * //

// permeability of vacuum
muVac          muVac [1 1 -2 0 0 -2 0] 1.256637e-6;
// relative permeability
muR            muR [0 0 0 0 0 0 0] 1.0;
// permittivity of vacuum
eps            eps [-1 -3 4 0 0 2 0] 8.854183e-12;
// relative permittivity
epsR           epsR [0 0 0 0 0 0 0] 1.0;
// wavelength
lambda         lambda [0 1 0 0 0 0 0] 1e-3;
// Laser Power
laserPower     laserPower [1 2 -3 0 0 0 0] 2e3;
// Laser efficiency
laserEfficiency laserEfficiency [0 0 0 0 0 0 0] 0.3;
// * * * * * //

```

The thermophysical properties file needs to be updated according to which liquid and gas being used.

In `constant` directory change the thermophysical properties to the properties for steel. Change the

name of the file `thermophysicalPropertieswater` to `thermophysicalPropertiessteel`.

```
mv constant/thermophysicalPropertieswater constant/thermophysicalPropertiessteel
```

Open the file

`thermophysicalPropertiessteel` and change

```
thermoType
{
    type            heRhoThermo;
    mixture         pureMixture;
    transport       const;
    thermo          hConst;
    equationOfState perfectFluid;
    specie          specie;
    energy          sensibleInternalEnergy;
}
mixture
{
    specie
    {
        nMoles      1;
        molWeight   50;
    }
    equationOfState
    {
        R           3000;
        rho0        7287;
    }
    thermodynamics
    {
        Cp          573;
        Hf          0;
    }
    transport
    {
        mu          5e-3;
        Pr          0.427;// As function of temperature (mu*Cp)/k and k is 3.75e-3*T
    }
}
// ***** //
```

In this test case surface forces are ignored so gravity g should be set to zero. Open the file `constant/g` and set

```
value    (0.0 0.0 0.0);
```

In `thermophysicalProperties` change the value for σ to

```
sigma    sigma [ 1 0 -2 0 0 0 ] 0;
```

`turbulenceProperties` and `thermophysicalPropertiesair` are kept unchanged.

3.4 Solver settings

When adding a new equation we need also to complete the `fvSchemes` and `fvSolution` files to tell OpenFOAM which discretization schemes and solvers to apply. The `fvSolution` file consists of settings related to solving of matrix equations. It has a solvers and PISO subdictionary and it

controls solvers tolerances and algorithms for equation solving. The solvers subdictionary specify which linear solver to be used for solving each discretized equation, including options for smoother and tolerances. The PISO subdictionary provides information about the iterative solution of velocity and pressure equations.

Open the `fvSolution` file in system directory and add a solver for E before U

```
E
{
    solver          smoothSolver;
    smoother        GaussSeidel;
    tolerance        1e-06;
    relTol           0;
    nSweeps          1;
}
EFinal
{
    solver          smoothSolver;
    smoother        GaussSeidel;
    tolerance        1e-06;
    relTol           0;
    nSweeps          1;
}
```

Then in the same file at the end of PIMPLE add

```
pRefPoint (5.5e-4 6.5e-4 5.0e-4); // represents center point of the domain on top surface
pRefValue 101300; //atmospheric pressure
```

Save and close the file.

Open the `fvScheme` in systems directory and at the end of `divSchemes` add

```
div(V)          Gauss upwind;
```

The last step before running the test case is to change the `controlDict` dictionary in system directory

```
application      compressibleInterFoamEikonal;
```

At the end (or beginning) of the `controlDict` file add the library for the new boundary condition

```
libs ("libmyFiniteVolume.so");
```

3.5 Run the simulation

Now everything is set and the simulation can be run. Start by copying the original files for the `setFields` utility

```
cp 0/E.org 0/E
cp 0/alphasteel.org 0/alphasteel
cp 0/T.org 0/T
cp 0/p.org 0/p
cp 0/p_rgh.org 0/p_rgh
cp 0/U.org 0/U
```

Generate the mesh and run `setFields`

```
blockMesh
```

```
setFields
```

Initilize the solver to run in the background and write output to a log file

```
compressibleInterFoamEikonal >&log&
```

3.7 Input parameters and values

The table 3.2 shows the properties of the laser beam and the electric field used in this test case. In system/fvSolution the solver control need to be updated with the solver for E.

Parameter (unit)	Symbol	Value
Permeability of vacuum [$\text{kg m s}^{-2} \text{A}^{-2}$]	μ_0	$1.256637 \cdot 10^{-6}$
Relative permeability	μ_r	1.0
Permittivity of vacuum [$\text{kg}^{-1} \text{m}^{-3} \text{s}^4 \text{A}^2$]	ϵ_0	$8.854183 \cdot 10^{-12}$
Relative permittivity	ϵ_r	1.0
Wave length [m]	λ	$50 \mu\text{m}$
Laser Power [W]		2.0kW
Laser Efficiency		0.3

Table 3.2: Input parameters and values

Note: OpenFOAM has *solver applications* and *linear solvers*. Solver applications are the programs used directly for running the simulation while linear solvers are the algorithms used to solve linear algebraic equation systems.

3.8 Material properties

In this test case steel and air are used for liquid and gas. Material properties used in the test case are shown in table 3.3.

They are set in the constant-directory in the file `thermophysicalPropertiesair` and

Properties (unit)	Symbol	Value
Density of steel [kg m^{-3}]	μ_0	$1.256637 \cdot 10^{-6}$
Specific heat of steel [$\text{J kg}^{-1} \text{K}$]	C_p	537.0
Dynamic viscosity steel [Pa s]	μ	$5 \cdot 10^{-3}$
Prandtl number steel	Pr	0.427
Specific heat of air [$\text{J kg}^{-1} \text{K}$]	C_p	1007.0
Dynamic viscosity air [Pa s]	μ	$1.84 \cdot 10^{-5}$
Prandtl number air	Pr	0.7

Table 3.3: Material properties

`thermophysicalPropertiessteel` respectively as described previously.

Chapter 4

Result

4.1 compressibleInterFoamEikonal results

The solver works but input parameters need to be optimized in order to give a converged solution with good results.

The profile of the temperature field looks as expected with a Gaussian distribution, even though the temperature values are too large.

The conclusion is that the mesh should be refined and input parameters optimized in order to give a more "real" prediction of temperature values. The distribution profile is good enough.

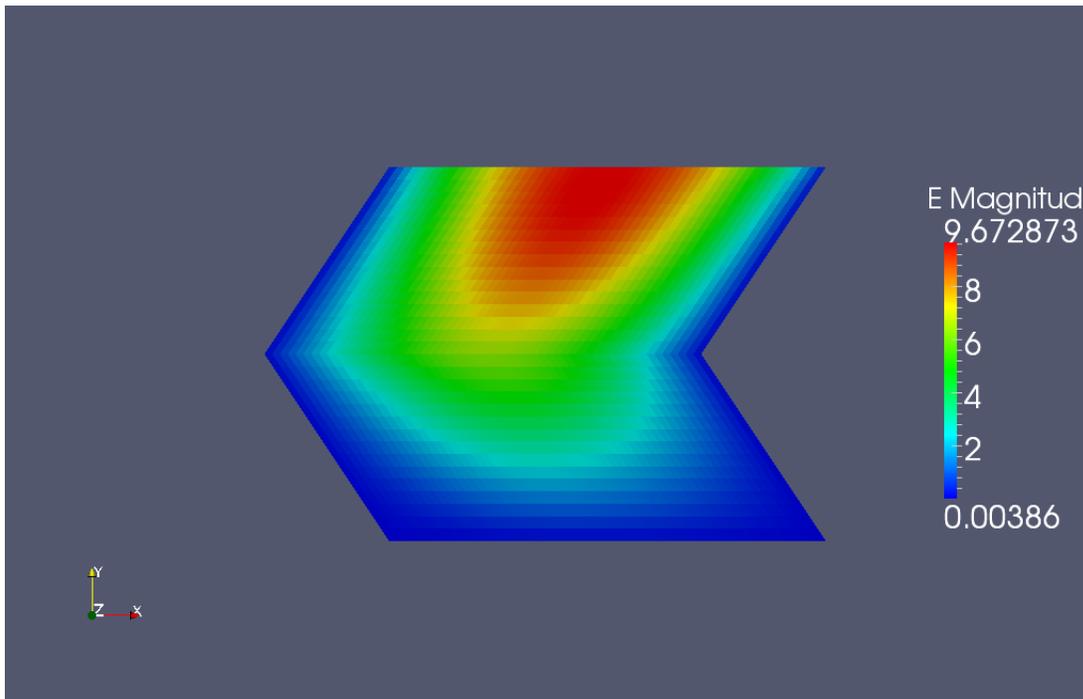


Figure 4.1: Electric field distribution

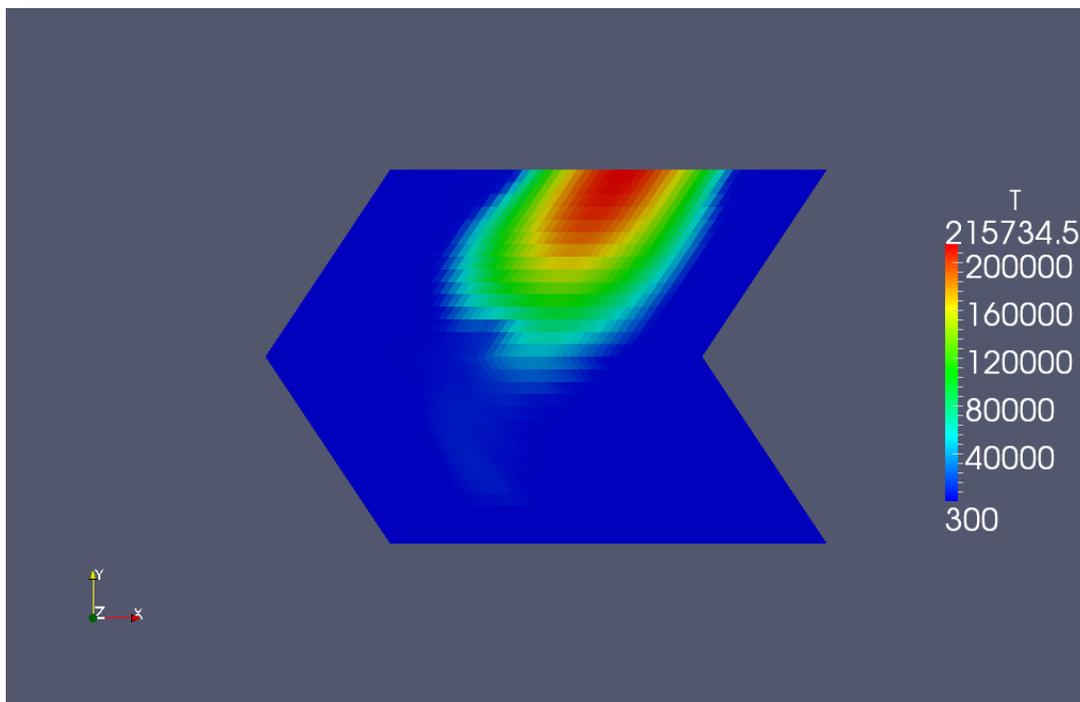


Figure 4.2: Temperature distribution

Chapter 5

Future work

The test case may be improved in computational time and convergence rate by optimizing the discretization schemes and the solver control in fvSchemes and fvSolution respectively.

How to treat a set of equations requiring very different grid size?

- Can chtMultiRegion solver be used?

chtMultiRegionFoam is a combination of heatConductionFoam and buoyantFoam to include heat transfer between a solid and liquid region.

- Can a multi-scale model be used?

In OpenFOAM there exists a multi-scale approach which can be used to simulate cavitation. In this multi-scale approach small spherical bubbles are modelled with DBM, discrete bubble modelling, on a Lagrangian framework. For larger structures VOF on Eulerian framework is used. All small bubbles are identified and listed and a method for determining whether the bubble is close to an Eulerian surface is used by evaluating the location of the Lagrangian bubble surface.

An Eulerian mesh is the "background" mesh with a fixed grid while a Lagrangian mesh deforms together with the material.

Study questions

1. What are the different ways to construct and declare a volume scalar or vector field?
2. SetFields utility includes many different selection "tools". How can you find the different available options/types?
3. What inputs are needed to use rotatedBoxToCell type selection in setFields? And how can you define the selection?
4. What are the basic steps when implementing a new BC?
5. What kind of information is provided through fvSolution file?
6. What option should be used together with `wmake` command in order to compile a dynamically linked library?
7. Why is it important to use `wclean` before compiling a new application?
8. Where is the source file for fvPatchField boundary conditions located?
9. What is a dynamic library/dynamic linking?
10. What different types of volume fields exist in OpenFOAM?
11. What type of data is stored in volume fields?
12. What is the difference between applications solvers and linear solvers?

Bibliography

- [1] A. K. Sen, S.H. Davis (1982), Steady thermocapillary flows in two-dimensional slots. *Journal of Fluid Mechanics* **121**, pp. 163-186
- [2] T. Maric, J. Höpken, K. Mooney (2014) The OpenFOAM Technology Primer
Publisher: sourceflux UG
- [3] M. Courtois, M. Carin, P. Le Masson, S. Gaied, M. Balabane (2013) A new approach to compute multi-reflections of laser beam in a keyhole for heat transfer and fluid flow modelling in laser welding *Journal of Physics D* **46**, pp.505305-505319
- [4] M. Courtois, M. Carin, P. Le Masson, S. Gaied, M. Balabane (2014) A complete model of keyhole and melt pool dynamics to analyze instabilities and collapse during laser welding *Journal of Laser Applications* **26**, pp.042001
- [5] A. Q. Raeini, M. J. Blunt, B. Bijeljic (2012) Modelling two-phase flow in porous media at the pore scale using the volume-of-fluid method *Journal of Computational Physics* **231**, pp. 5653-5668
- [6] A. Satya Narayanan, S. K. Saha (2015) Waves and oscillations in nature - An Introduction
CRC Press, Taylor and Francis Group
- [7] O. Svelto (2010) Principles of laser
Springer, 5th edition
- [8] www.openfoam.org
- [9] https://en.wikipedia.org/wiki/Gaussian_beam