

CFD with OpenSource software, project

Implementing “chtMultiRegionFoam” Solver for Electric Welding in OpenFoam 1.6.x

Author

Alireza Javidi

Reviewers

Erick Svenning

Jelena Andric

October 2010

Contents:

- 1 Introduction 2**
- 2 chtMultiRegionFoam solver 3**
 - 2-1 OpenFoam – 1.5.x 3**
 - 2-2 OpenFoam – 1.6.x and 1.7.x 5**
- 3 Implementing chtMultiRegionFoam 7**
 - 3-1 mesh generation 7**
 - 3-2 case structure 8**
 - 3-3 Boundary conditions 8**
 - 3-4 Thermal conductivity 11**
 - 3-5 Post processing 12**
 - 3-6 results 12**
- 4 Adding Electric Potential equation 13**
 - 4-1 Maxwell equation 13**
 - 4-2 Electric conductivity 14**
 - 4-3 Electrical potential equation 17**
 - 4-4 Boundary conditions 17**
 - 4-5 Results 20**
- 5 Conclusions and future works 22**
- 6 References 22**

1 Introduction

Understanding the modeling of electric arcs has always been an interesting scientific subject since the first detection of the arcing phenomena. Electric arc modeling is one of the main tasks in the simulation of an overall welding process. The purpose of the arc welding modeling effort is to predict the energy distribution supplied to the workpiece by the source of the welding.

The Major physical processes which should be taken into account in the modeling of the arc welding are the heat transfer and the electric conduction.

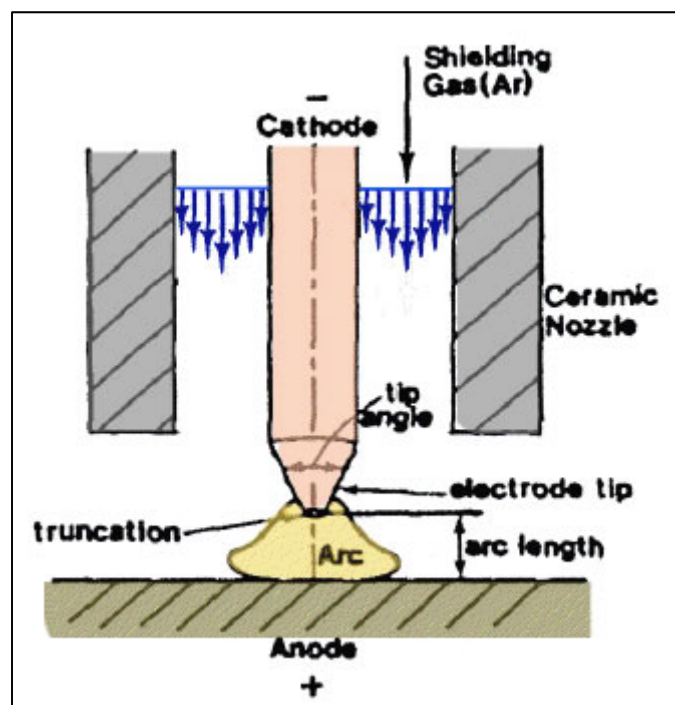


Figure (1-1) schematic sketch of welding

Due to the electric potential difference applied between anode (work piece) and cathode, an electric current flow through the shielding gas. The electrons emitted by the cathode gain kinetic energy from the applied electric field. This energy is large enough to heat the shielding gas via Joule heating and form thermal plasma with high temperature. The heat generated in the plasma and the electric potential within the plasma influence temperature and electric fields inside the solid. The electric welding process is characterized by a large temperature range, from ambient temperature to about 25000 degree centigrade. Since all of the material properties are temperature dependent it can lead to huge variations on the transport coefficients. Therefore, the energy equation in the domains and the heat interaction between solid part (work piece) and fluid part (plasma) can have an important role in the electric welding modeling.

The aim of this report is to understand and describe a solver – so called *chtMultiRegionFoam* – in OpenFoam 1.6.x which can be employed to make an energy coupling between two different regions and also employ this solver to make a coupling not only in energy but also in electric potential field for modeling electric welding.

2 chtMultiRegionFoam solver

The solver is based on combination of *heatConductionFoam* and *buoyantFoam* for conjugate heat transfer between a solid region and a fluid region. Different possible ways to employ the coupling boundary condition are discussed in the solver tutorial by Moradnia[1]. However, a new type of coupling boundary is introduced in OpenFoam 1.6.x. This report will briefly review the alternatives to impose the coupling boundary condition in version 1.5.x and then discuss about version 1.6.x. It should be noticed that version 1.7.x has the same features as version 1.6.x.

2-1 OpenFoam – 1.5.x

It is not in the scope of this report to describe about the solver in OpenFoam 1.5.x. However, it should be mentioned that there are some differences between 1.5.x and 1.6.x versions. A very brief description of the solver in version 1.5.x is done in this section to demonstrate the different way of imposing coupling boundary. For further

information about implementing the solver in version 1.5.x can be found in a tutorial written by Moradnia[1].

There are three different ways to employ coupling boundary in the solver[1]

- *solidWallTemperatureCoupled*:

Fixed value boundary condition for temperature.

```
myInterfacePatchName
{
    type                solidWallTemperatureCoupled;
    neighbourRegionName fluid;
    neighbourPatchName  fluidSolidInterface;
    neighbourFieldName  T;
    K                   K;
    value               uniform 300;
}
```

- *solidWallHeatFluxTemperature*:

This allows introducing constant heat flux to a patch and it has nothing to do with the coupling.

```
myInterfacePatchName
{
    type                solidWallHeatFluxTemperature;
    K                   K;          < Name of K field >
    q                   uniform 1000; < Heat flux [W/m2] >
    value               300.0;      < Initialtemperature [K] >
}
```

- *solidWallHeatFluxTemperatureCoupled*:

It is fixed heat flux boundary condition for temperature.

```

myWallPatchName
{
    type                solidWallHeatFluxTemperatureCoupled;
    neighbourRegionName fluid;
    neighbourPatchName  fluidSolidInterface;
    neighbourFieldName  T;
    K                   K;
    value               uniform 300;
}

```

To make a coupling in this version of OpenFoam the *solidWallTemperatureCoupled* boundary should be used in one neighbor and *solidWallHeatFluxTemperatureCoupled* in another neighbor of the patch.

2-2 OpenFoam – 1.6 and 1.7

This version of OpenFoam provides new interface boundary condition for coupling. There are two alternatives for interface boundary conditions:

- *solidWallHeatFluxTemperature:*

This alternative is the same as previous version and it allows introducing constant heat flux to a patch. It is important to notice that it has nothing to do with coupling. The function of this boundary condition is equal to fixedGradient boundary condition.

```

myInterfacePatchName
{
    type                solidWallHeatFluxTemperature;
    K                   K;          < Name of K field >
    q                   uniform 1000; < Heat flux [W/m2] >
    value               300.0;      < Initialtemperature [K] >
}

```

- *solidWallMixedTemperatureCoupled*:

This is a mixed boundary condition for temperature, to be used by conjugate heat transfer solver. Both sides use a mix of zero gradient and neighbor value. In fact, this coupling condition decides the direction the heat flux. It can be set automatically to be fixed value or fixed gradient depending on the direction of heat flux.

```
myInterfarcePatchName
{
  type                solidWallMixedTemperatureCoupling;
  neighbourFieldName  T;
  K                   K;
  value               uniform 300;
}
```

3 Implementing chtMultiRegionFoam

To start the implementation of the case, it looks more convenient to build a clear standard OpenFoam case model. The implementation is as follows^[3]:

3-1 mesh generation:

Since the main focus of this report is on the coupling conditions and the interaction between different regions, simple test geometry is used.

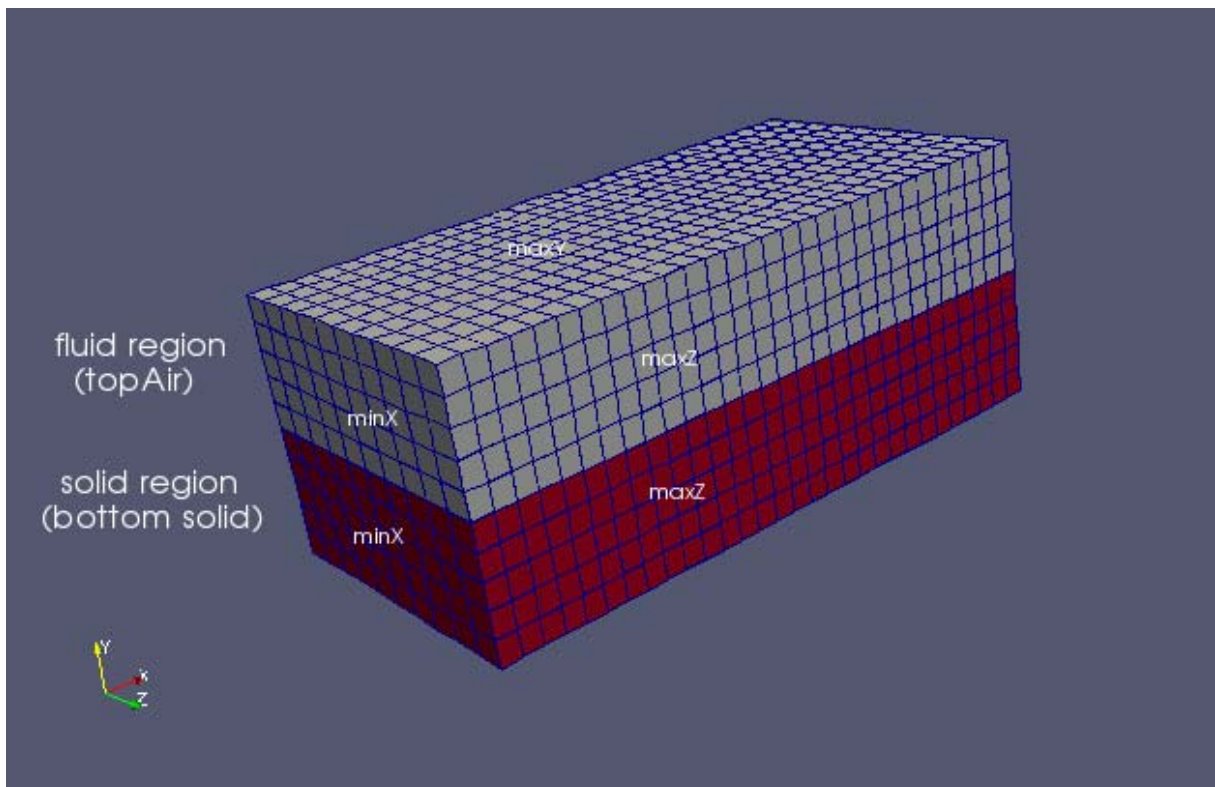


Figure (3-1) geometry of the test case with two different regions

To have a functional mesh case for this solver, the following procedure should be done on the mesh after generating the case by blockMesh:

- `setset -batch makecellSets.setset` (to define region sets)
- `setsToZones -noFlipMap` (to add zones to the mesh with similar sets name)
- `splitMeshRegions -cellZones -overwrite` (to split mesh into multiple regions)

Note: The mesh at the coupled patch should be 1:1. It means the number of *nFaces* in solid and fluid region should be the same.

3-2 case structure:

It is preferable to make a new case file and construct the case as follows:

- “0”-folder: it contains the region folders (in our case *topAir* and *bottomSolid*). In each region folder there are respective data fields for initialization (U, T, p for *topAir* and cp, K, rho, T for *bottomSolid*).
- “constant”-folder: it contains the *regionProperties* files which introduce the region names and also the domain folders (*topAir*, *bottomSolid*). Each region folder has the properties files and also the *PolyMesh*.
- “system”-folder: with *controlDict* file and in each region folder with corresponding *fvSchemes* and *fvSolution* file.

3-3 Boundary conditions:

Between different domains boundary conditions for temperature should be imposed by coupling boundary. See codes (3-2, 3-3).

As can be seen in highlighted part of the codes, the interface boundary name is changed to “*topAir_to_bottomSolid*” and “*bottomSolid_to_topAir*” with respect to the region.

It should be noticed that fix temperature values are imposed to the boundary *maxY*(=100 °C) and *minY*(=2000 °C) to provide temperature gradient from bottom and top faces of the case. It makes it easier to check the validity of the results by plotting temperature distribution in both fluid and solid regions.

There is no need to define coupling condition in the interface boundary for the other parameters like U, rho, p. Since there is no coupling for these parameters the boundary can be defined the same as other boundary conditions.

```

FoamFile
{
    version      2.0;
    format       ascii;
    class        volScalarField;
    location     "0";
    object       T;
}
// * * * * *

dimensions      [ 0 0 0 1 0 0 0 ];
internalField   uniform 300;
boundaryField
{
    maxY
    {
        type      fixedValue;
        value     uniform 100;
    }
    minX
    {
        type      zeroGradient;
        value     uniform 300;
    }
    maxX
    {
        type      zeroGradient;
        value     uniform 300;
    }
    minZ
    {
        type      zeroGradient;
        value     uniform 300;
    }
    maxZ
    {
        type      zeroGradient;
        value     uniform 300;
    }
    topAir_to_bottomSolid
    {
        type      solidWallMixedTemperatureCoupled;
        value     uniform 300;
        neighbourFieldName T;
        K         K;
    }
}

```

code (3-2) Temperature boundary field for *topAir* region

```

\*-----*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        volScalarField;
    location     "0";
    object       T;
}
// *****

dimensions      [ 0 0 0 1 0 0 0 ];

internalField   uniform 300;

boundaryField
{
    minX
    {
        type          zeroGradient;
        value         uniform 300;
    }
    maxX
    {
        type          zeroGradient;
        value         uniform 300;
    }
    minZ
    {
        type          zeroGradient;
        value         uniform 300;
    }
    maxZ
    {
        type          zeroGradient;
        value         uniform 300;
    }
    minY
    {
        type          fixedValue;
        value         uniform 2000;
    }
    bottomSolid_to_topAir
    {
        type          solidWallMixedTemperatureCoupled;
        value         uniform 300;
        neighbourFieldName T;
        K              K;
    }
}

// *****

```

code (3-3) Temperature boundary field for *bottomSolid* region

3-4 Thermal conductivity:

Same thermal conductivity in both solid and fluid region leads to linear electric potential distribution in the case. An expected linear integrated distribution of the EIPot in both domains can easily reveal any possible drawback or problem in the coupling boundary function. Thermal conductivity (K) and Specific heat (cp) in the solid part is defined as an initial condition and it is set to K=80 and cp=450. In the fluid part, some material properties are defined in “*thermophysicalProperties*”. The *thermophysicalProperties* dictionary is read by any solver that uses the thermophysical model library. A thermophysical model is constructed in OpenFOAM as a pressure-temperature $p - T$ system from which other properties are computed[4].

The quantities of the fluid in *thermophysicalProperties* are defined as follows:

Mixture *gasName* *n* *W* *cp* *Hf* *mu* *Pr*

“n” is number of moles, “W” is molecular weight, “cp” is heat capacity, “Hf” is heat of fusion, “mu” is dynamic viscosity and “Pr” is Prandtl number.

Thermal conductivity is equal to “ $cp * mu * rPr$ ” where rPr is reciprocal Prandtl number. Thus, to have a same cp and K in both fluid and solid region in the “*thermophysicalProperties*” file these properties changed and make an imaginary fluid with the same thermal conductivity and specific heat.

$Cp=450$, $mu=0.018$ and $Pr=0.10125$

```

\*-----*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    object       thermophysicalProperties;
}
// ***** //

thermoType
hPsiThermo<pureMixture<constTransport<specieThermo<hConstThermo<perfectGas>>>>>>;

mixture        air_fake 1 28.9 450 0 1.8e-02 0.10125;

// ***** //

```

code (3-4) thermo physical Properties of fluid part

3-5 Running and Post processing:

Running the case is possible by typing “chtMultiRegionFoamElpotFSsigma”. To visualize the result the touch command should be used for all regions to make them readable by *paraview*.

```
paraFoam - touch -topAir
```

```
paraFoam - touch -bottomSolid
```

Afterwards, it is possible to visualize the result in *paraview* by opening separately *bottomSolid* and *topAir* regions

3-6 results:

As it is illustrated in fig (3-5) and graph (3-1), the temperature is increasing linearly from minimum value in *bottomSolid* region to the maximum value in the *topAir* region.

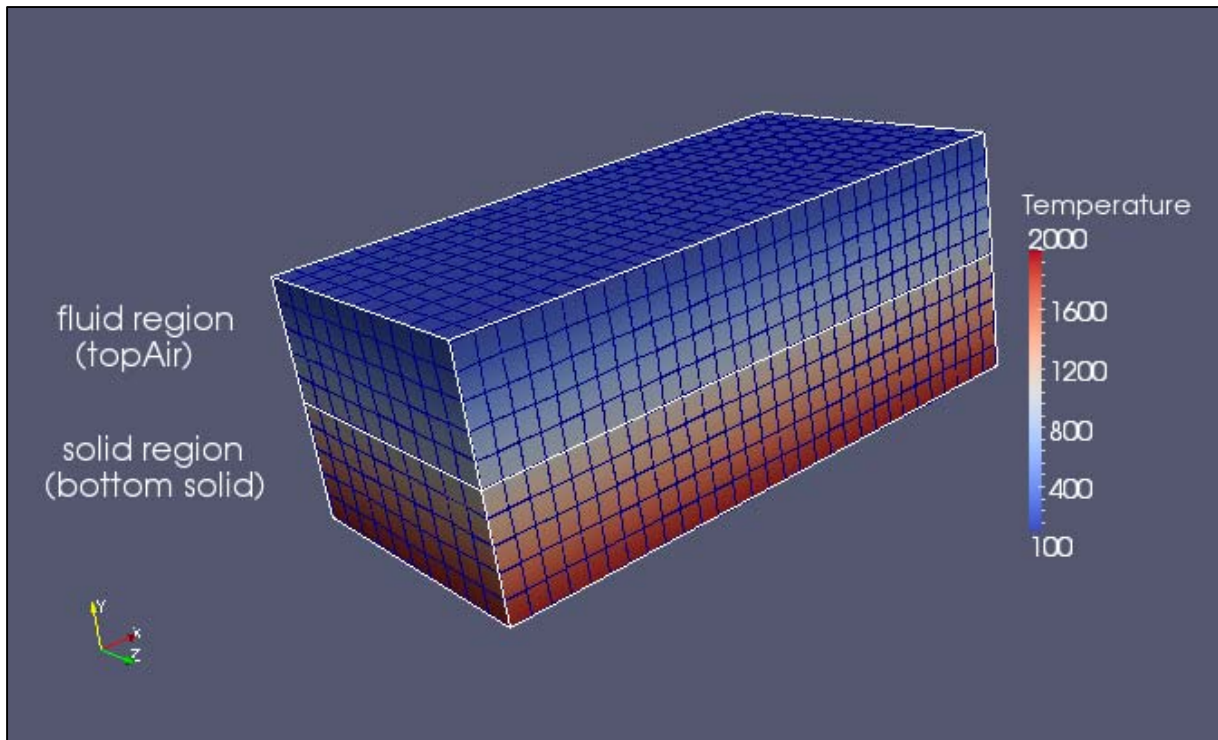
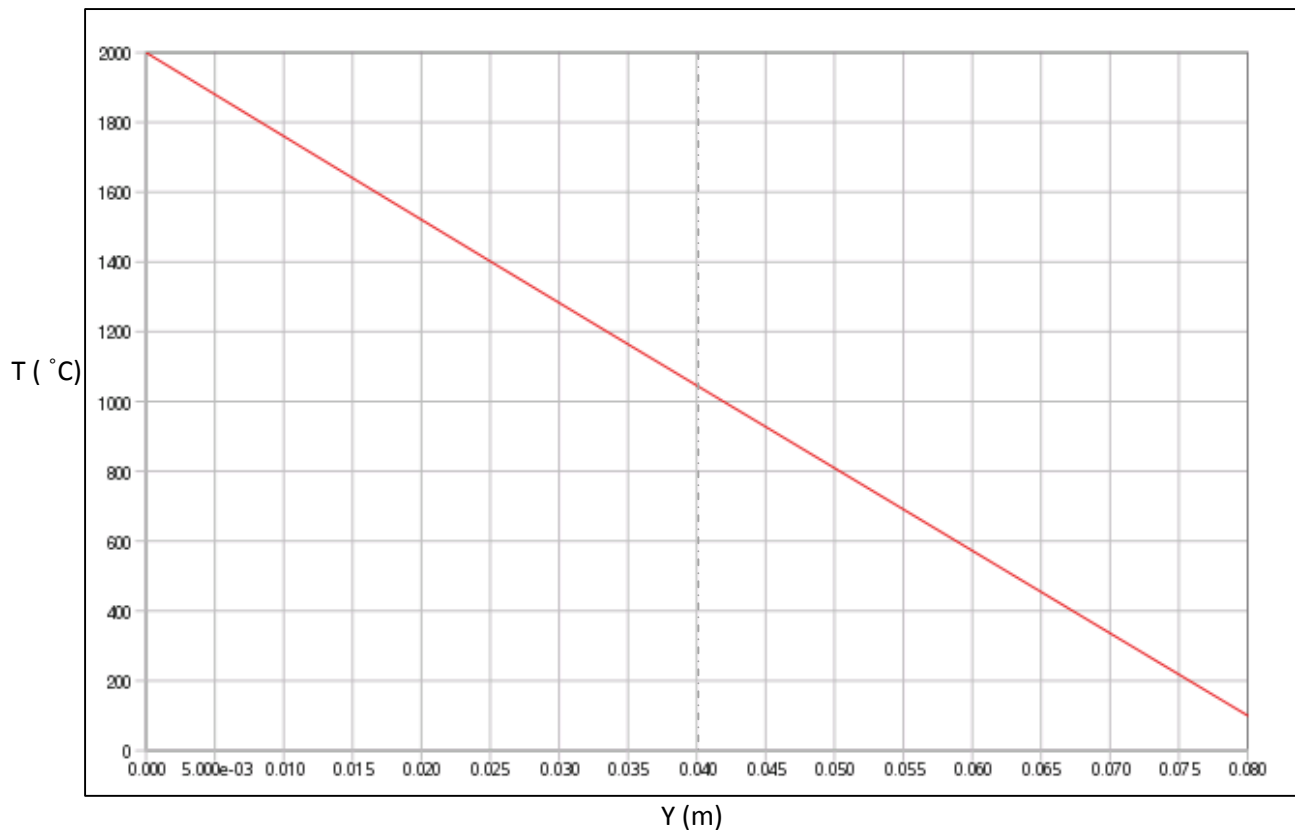


Figure (3-5) temperature in both solid and fluid region

Graph (3-1) shows the temperature distribution in the domain from $maxY(=100\text{ }^{\circ}\text{C})$ to $minY(=2000\text{ }^{\circ}\text{C})$. The line at $y=0.04$ shows the interface boundary between two different regions. As can be seen there is no fluctuation in the interface boundary at interface boundary. It means that the interface boundary condition can create a full energy coupling between two different regions in the case file.



Graph(3-1) temperature in both solid and fluid region

4 Adding Electric Potential equation

4-1 Maxwell equation:

The Maxwell's equation in terms of potential ϕ is reduced to the following form:

$$\nabla \cdot (\sigma_m \nabla \phi) = 0$$

Where, σ_m is the electric conductivity. The electric potential in welding modeling is used to get the electric current density (j) and forming the joule heating source in the enthalpy conservation equation. However, adding the joule heating source term is not in the scope of this report.

$$j = -\sigma_m \nabla \phi$$

4-2 Electric conductivity:

Before including the electric potential laplacian equation to the solver, the electric conductivity, so called *sigmaMag*, should be introduced to the solver. The electrical conductivity is a transport coefficient of the material in both fluid and solid field. It is temperature dependent and has a huge variation in the welding phenomenon due to the large temperature difference. However, in this report due to simplification of the model this parameters considered as a constant value. Obviously, after achieving this step the electric conductivity should be included in thermal library as a parameter of the model.

NOTE: A Closer look to the solid solver reveals that in the solid region the following quantities: thermal conductivity (K), specific heat (cp), and density (rho), are set as fields for unknown variables. They are thus initialized within the computational domain, and set on the boundaries, but these quantities are not the unknowns of the energy equation to be solved, and should not need boundary conditions. It seems that this may have been done for simplifying the implementation. It would seem more consistent to implement these quantities via the library "*thermophysicalModels*", as done for the corresponding quantities in the fluid region. In the present project, for simplicity, the electrical conductivity will be implemented in the same manner and will be introduced as a scalar field in the domain, setting initial and boundary conditions (although meaningless). But this quick implementation does not seem to be a good option for extending the model to temperature dependent thermo-physical parameters. Thus, what is coming in this report for setting up *sigmaMag* is done as for a field with initialization and boundary conditions, and not as a parameter.

As it is mentioned, the electric conductivity needs to be added to the solver both in fluid and solid region. For this purpose, it should be included in *setRegionalSolidFields.H* and *creatSolidFields.H* for solid case as it is illustrated in codes (4-1, 4-2).


```

fvMesh& mesh = solidRegions[i];

volScalarField& rho = rhos[i];
volScalarField& sigmaMag = sigmaMags[i];
volScalarField& ElPot = ElPots[i];
volScalarField& cp = cps[i];
volScalarField& K = Ks[i];
volScalarField& T = Ts[i];

```

Code (4-1) setting *ElPot* and *sigmaMag* in *setRegionalSolidFields.H*

```

Info<< "    Adding to ElPots\n" << endl;
ElPots.set
(
    i,
    new volScalarField
    (
        IOobject
        (
            "ElPot",
            runTime.timeName(),
            solidRegions[i],
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        solidRegions[i]
    )
);

Info<< "    Adding to sigmaMags\n" << endl;
sigmaMags.set
(
    i,
    new volScalarField
    (
        IOobject
        (
            "sigmaMag",
            runTime.timeName(),
            solidRegions[i],
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        solidRegions[i]
    )
);

```

Code (4-2) setting *sigmaMag* and *ElPot* in *creatSolidFields.H*

The same method is used to add *sigmaMag* to the fluid case of the solver:

```
Info<< "    Adding to sigmaMagFluid\n" << endl;
sigmaMagFluid.set
(
  i,
  new volScalarField
  (
    IOobject
    (
      "sigmaMag",
      runTime.timeName(),
      fluidRegions[i],
      IOobject::MUST_READ,
      IOobject::AUTO_WRITE
    ),
    fluidRegions[i]
  )
);

Info<< "    Adding to ElPotFluid\n" << endl;
ElPotFluid.set
(
  i,
  new volScalarField
  (
    IOobject
    (
      "ElPot",
      runTime.timeName(),
      fluidRegions[i],
      IOobject::MUST_READ,
      IOobject::AUTO_WRITE
    ),
    fluidRegions[i]
  )
);
```

Code (4-3) adding *sigmaMag* and *ElPot* to *creatFluidFields.H* and

4-3 Electrical potential equation:

Electric potential fields, so called *EIPot*, should be introduced to the solver as well as *sigmaMag* codes (4-2, 4-3) show that *EIPot* is added to both *creatFields* and *setRegionFields* of solid and fluid region.

The next step is to add the electric potential laplacian equation to the solver. It is done by adding *VEqn.H* file in both solid and fluid solver. This file includes laplacian equation of electric potential(see code (4-4)).

```
// Solve equation for electric potential ElPot
//Info << " Solve the electric potential equation " << endl;
Info<< "debut VEqn.H - sigmaMag max/min : " << max(sigmaMag).value() << "
" << min(sigmaMag).value() << endl;
{
  solve
  (
    fvm::laplacian(sigmaMag, ElPot)
  );
}
```

Code (4-4) Electric potential laplacian equation

4-4 Boundary conditions:

To define the coupling boundary condition for electric potential, the same pattern as for the energy coupling is implemented. Codes (4-5, 4-6) illustrate the coupling boundary condition for *EIPot* in fluid and solid region.

```

\*-----*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        volScalarField;
    location     "0";
    object       ElPot;
}
// ***** //
dimensions     [1 2 -3 0 0 -1 0];

internalField  uniform 0;

boundaryField
{
    maxY
    {
        type          fixedValue;
        value         uniform -2;
    }
    minX
    {
        type          zeroGradient;
    }
    maxX
    {
        type          zeroGradient;
    }
    minZ
    {
        type          zeroGradient;
    }
    maxZ
    {
        type          zeroGradient;
    }
    topAir_to_bottomSolid
    {
        type          solidWallMixedTemperatureCoupled;
        value         uniform 0;
        neighbourFieldName ElPot;
        K             sigmaMag;
    }
}

// ***** //

```

Code (4-5) *ElPot* boundary in *topAir* region

```

\*-----*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        volScalarField;
    location     "0";
    object       ElPot;
}
// *****
* * //
dimensions      [1 2 -3 0 0 -1 0];

internalField   uniform 0;

boundaryField
{
    minY
    {
        type          fixedValue;
        value          uniform 3;
    }
    minX
    {
        type          zeroGradient;
    }
    maxX
    {
        type          zeroGradient;
    }
    minZ
    {
        type          zeroGradient;
    }
    maxZ
    {
        type          zeroGradient;
    }
    bottomSolid_to_topAir
    {
        type          solidWallMixedTemperatureCoupled;
        value          uniform 0;
        neighbourFieldName ElPot;
        K              sigmaMag;
    }
}

// ***** //

```

Code (4-6) *ElPot* boundary in *bottomSolid* region

4-5 Results:

By choosing the same electric conductivity (σ_{Mag} quantity) we should have a linear Electric potential field in both regions from 3V in *bottomSolid* to -2V in *topAir*. The result is shown in figure (4-7) and in graph(4-1)

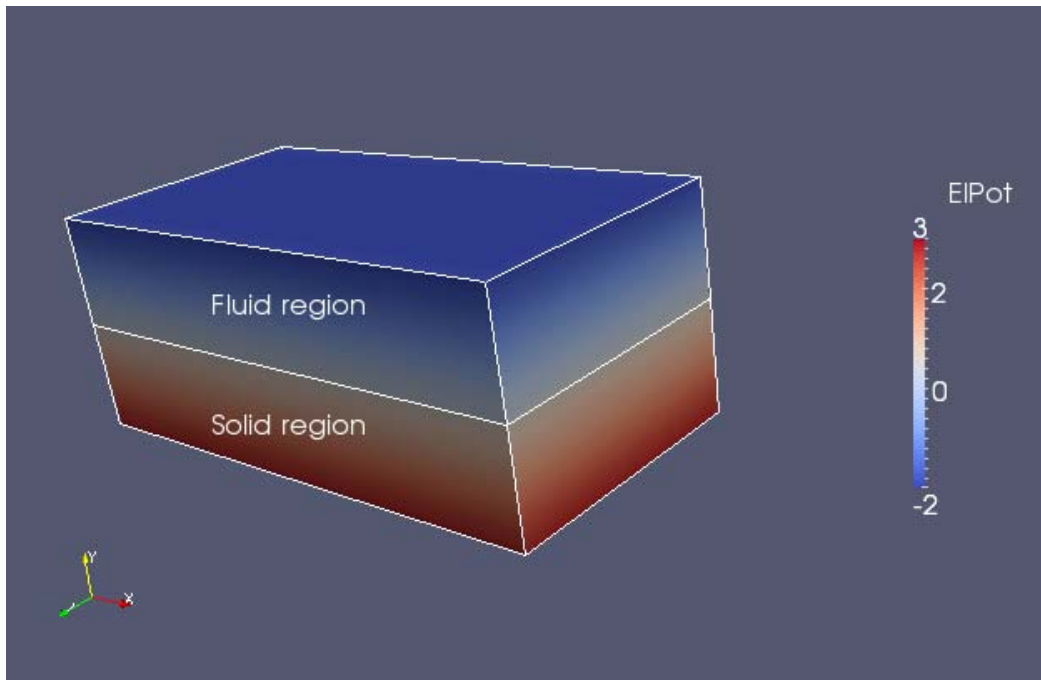
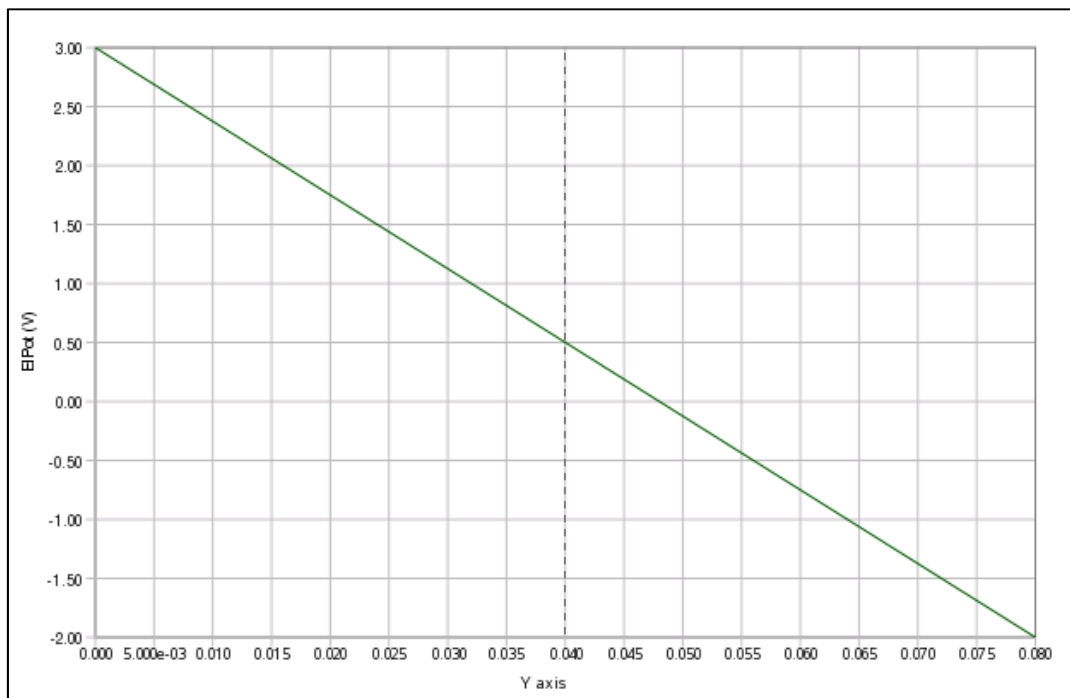


Figure (4-7) electric potential field with same electric conductivity in fluid and solid regions



Graph (4-1) electric potential field with the same electric conductivity in fluid and solid regions

To check the capability of the solver to handle different conductivities in each region, the electric conductivity of the fluid part reduced from 80 S/m to 20 S/m. Due to the difference between electric conductivity in solid and fluid domain a change in the electric potential field in the regions was expected. In graph (4-2) it is illustrated that there is an electric potential difference between two regions.

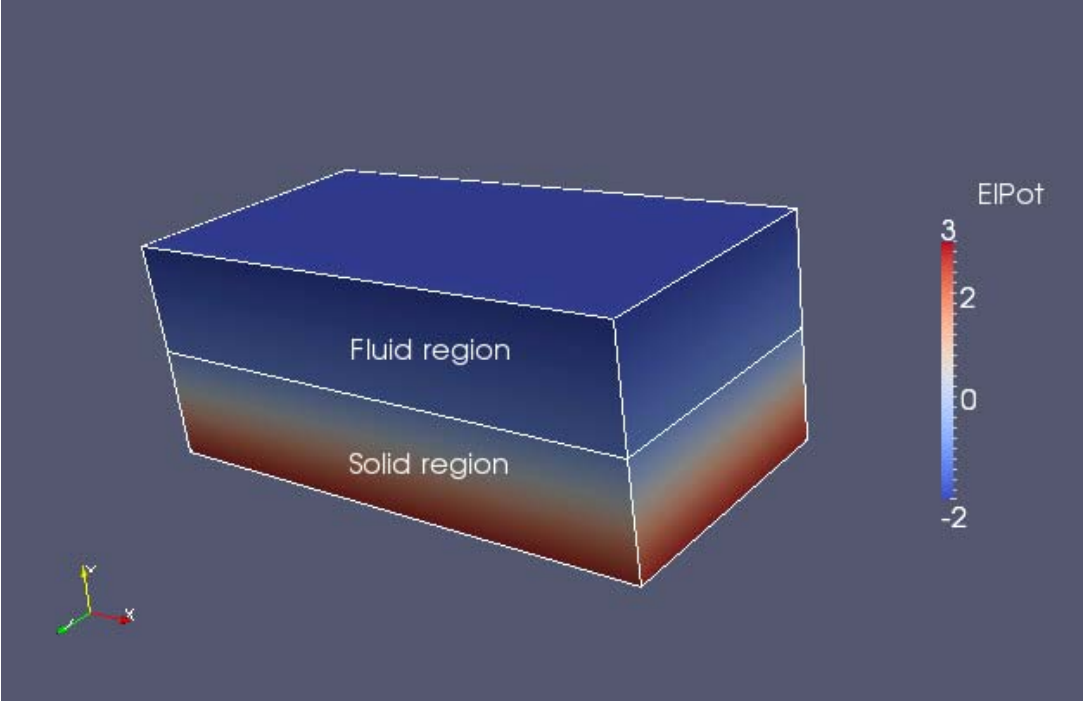
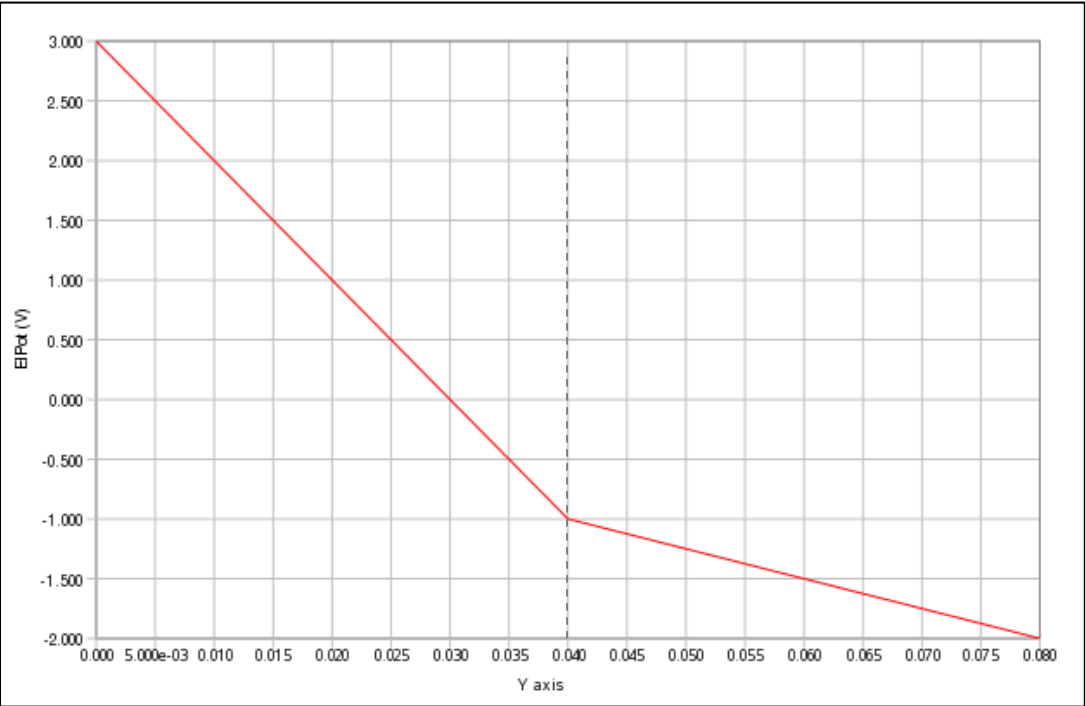


Figure (4-8) electric potential field with different electric conductivity in fluid and solid regions



Graph (4-2) electric potential field with different electric conductivity in fluid and solid regions

5 Conclusions and future works

In “*chtMultiRegionFoam*” solver, the coupling boundary condition can be implemented between different regions while solving laplacian electric potential in the domain.

However, as it is mentioned in section (4-2), unlike the fluid solver, in the solid solver the material quantities like thermal conductivity (K), specific heat (cp) and density (rho) are considered as a field for unknown variables. However, these parameters are not the unknown of the energy quantities to be solved. It seems it has been done for some simplifying since in most cases the solid quantities are not temperature dependent. For extending the model to temperature dependent solid parameters the future work can be to implement those quantities via the library “*thermophysicalModels*”, as done for the corresponding quantities in the fluid region.

6 References

[1] Moradnia, Pirooz 2008: “A description of how to do Conjugate Heat Transfer in OpenFOAM”

http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2008/chtFoam.pdf

[2] Järavstråt, Niklas, 2009, “Adding electric conduction and Joule heating to *chtMultiregionFoam*”

http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2009/NiklasJarvstrat/Project0126.pdf

[3] cfd-online OpenFoam forum

<http://www.cfd-online.com/Forums/openfoam-solving/>

[4] Open source CFD Toolbox

<http://www.openfoam.com/docs/user/thermophysical.php>