CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Project work:

A interphaseChangeFoam tutorial

Developed for OpenFOAM-2.0.x

Author:
Martin Andersen

Peer reviewed by:
Not yet

October 19, 2011

1 Introduction

This tutorial describes how to include a heat source in interPhaseChangeFoam to evapourate water. In order to do this the interPhaseChangeFoam solver is expanded to include a temperature field and the phase change algorithm is modified to take the temperature into account.

Before starting to change the interPhaseChangeFoam solver the reader will in Section 3 be guided through the process of copying the existing solver and the test case *cavitatingBullet* which comes with the OpenFOAM installation.

2 Description of interPhaseChangeFoam

In *interPhaseChangeFoam.C* the solver is described as follows:

> *Solver for 2 incompressible, isothermal immiscible fluids with phase-change (e.g. cavitation). Uses a VOF (volume of fluid) phase-fraction based interface capturing approach.*
> *The momentum and other fluid properties are of the "mixture" and a single momentum equation is solved.*
> *The set of phase-change models provided are designed to simulate cavitation but other mechanisms of phase-change are supported within this solver framework.*
> *Turbulence modelling is generic, i.e. laminar, RAS or LES may be selected.*

Three different cavitation models are provided with the *interPhaseChangeFoam* solver. They are Merkle, Knuz and SchnerrSauer. Here the Merkle mass transfer model will be described short. For a description of the implementation of Knuz and SchnerrSauer see e.g. http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2008/NaixianLu/REPORT_interPhaseChangeFoam.pdf. The vaporisation (transformation of liquid to vapor), $\dot{m}^-$ is modeled as being proportional to the liquid volume fraction, α, and the amount by which the pressure is under the saturation pressure. And the condensation (transformation of vapor to liquid), $\dot{m}^-$ is modeled similar.

$$\dot{m}^- = \frac{C_\mathrm{v}\rho_\mathrm{v}}{\frac{1}{2}\rho_\mathrm{l}U_\infty^2 t_\infty}\alpha\min(0, p - p_\mathrm{Sat}) \tag{1}$$

$$\dot{m}^+ = \frac{C_\mathrm{c}}{\frac{1}{2}U_\infty^2 t_\infty}(1-\alpha)\max(0, p - p_\mathrm{Sat}) \tag{2}$$

where:
C_c, C_v, U_∞ and t_∞ are empirical constants based on the mean flow,
ρ_l and ρ_v are the density of the liquid and vapor,
p is the pressure,
p_{Sat} is the vaporisation pressure.

This is a quit intuitive approach for the mass transfer. In the implementation in *interPhaseChangeFoam* these equations are split into a part used in the pressure-loop and a part for the α-loop. In the α-loop everything but α is held constant, in the pressure loop everything but p is constant. The two loops can be seen in the source code:

interPhaseChangeFoam.C

```
#include "fvCFD.H"
#include "MULES.H"
#include "subCycle.H"
#include "interfaceProperties.H"
#include "phaseChangeTwoPhaseMixture.H"
#include "turbulenceModel.H"
#include "pimpleControl.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

    pimpleControl pimple(mesh);

    #include "../interFoam/correctPhi.H"
    #include "CourantNo.H"
    #include "setInitialDeltaT.H"

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

    while (runTime.run())
    {
        #include "readTimeControls.H"
        #include "CourantNo.H"
        #include "setDeltaT.H"

        runTime++;

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

        #include "alphaEqnSubCycle.H"

        turbulence->correct();

        // --- Pressure-velocity PIMPLE corrector loop
        for (pimple.start(); pimple.loop(); pimple++)
        {
            #include "UEqn.H"

            // --- PISO loop
            for (int corr=0; corr<pimple.nCorr(); corr++)
            {
                #include "pEqn.H"
            }
        }

        twoPhaseProperties->correct();

        runTime.write();

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

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

    return 0;
}
```

This splitting of the mass transfer model makes it possible to iterate one parameter at a time making each iteration simpler and thereby quicker. And doing this is small intervals will not harm the converging rate of the algorithm.

3 Running a interPhaseChangeFoam case

This section covers the steps needed to run *interPhaseChangeFoam* on the test case *cavitatingBullet* in a user owned directory.

3.1 cavitatingBullet test case

First setup the OpenFOAM envirelment in the terminal.[1]

```
source $HOME/OpenFOAM/OpenFOAM-2.0.x/etc/myBashrc
```

Copy the *cavitatingBullet* to a user owned directory.

```
cp -r  $FOAM_TUTORIALS/multiphase/interPhaseChangeFoam/cavitatingBullet $FOAM_RUN
cd $FOAM_RUN/cavitatingBullet
```

The file structure of the *cavitatingBullet* case is similar to other OpenFOAM tutorials where the case directory has a `0/`, `constant/` and `system/` directory.

In the `0/` directory you find the initial field distributions. They are in this case

alpha1 is the phase field. The value 1 coresponds to liquid (in this case water) and the value 0 corespond to gas (in this case water vapour). In this case the initial value over the entire field is 1.

p_rgh in the pressure, p, minus $\rho \cdot g \cdot h$, where ρ is the denisty of the mater, g is gravity (usualy -9.8) and h is the height. In this case the initial value is 100000 Pa over the entire field.

U is the velosity field. In this case the initial velosity in the entire field is `(0 0 20)`, i.e. a flow of 20 m/s in the z-direction.

In the `constant/` you find different constants and stationary fields used in the simulation. In the `system/` directory you find different things to setup the simulation, e.g. the *controlDict* with all kinds of control variabels.

Before running the *cavitatingBullet* case open the file *system/controlDict* and change the *endTime* to 0.001.

```
./Allrun
```

This will take time depending on your computer. In the directory there are now a few *log.** files and a lot of time directories. You can have a loot at the simulation results with paraView.

[1] You might have an alias *OF20x* to do this

```
paraFoam
```

To see the vapor bobbles formed due to the flow around the bullet, follow the instructions in Figure 1.

When you are there also take a look at the velosity field and the pressure by selecting the corresponding entries in the drop-down menu in Figure 1c

3.2 Copy the solver

Before we start modifying the *interPhaseChangeFoam* we copy it to a new location and rename it to *myInterPhaseChangeFoam*.

```
mkdir -p $WM_PROJECT_USER_DIR/applications/solvers/multiphase
cd $WM_PROJECT_USER_DIR/applications/solvers/multiphase
cp -r $FOAM_SOLVERS/multiphase/interPhaseChangeFoam .
mv interPhaseChangeFoam myInterPhaseChangeFoam
cd myInterPhaseChangeFoam
mv interPhaseChangeFoam.C myInterPhaseChangeFoam.C
```

The solver *interPhaseChangeFoam* includes a file from *interFoam* by a relativ path. To keep things simple, here this file will be copied into the *myInterPhaseChangeFoam* folder:

```
cp $FOAM_SOLVERS/multiphase/interFoam/correctPhi.H .
```

Find the line:

```
#include "../interFoam/correctPhi.H"
```

in *myInterPhaseChangeFoam.C* and change it to

```
#include "correctPhi.H"
```

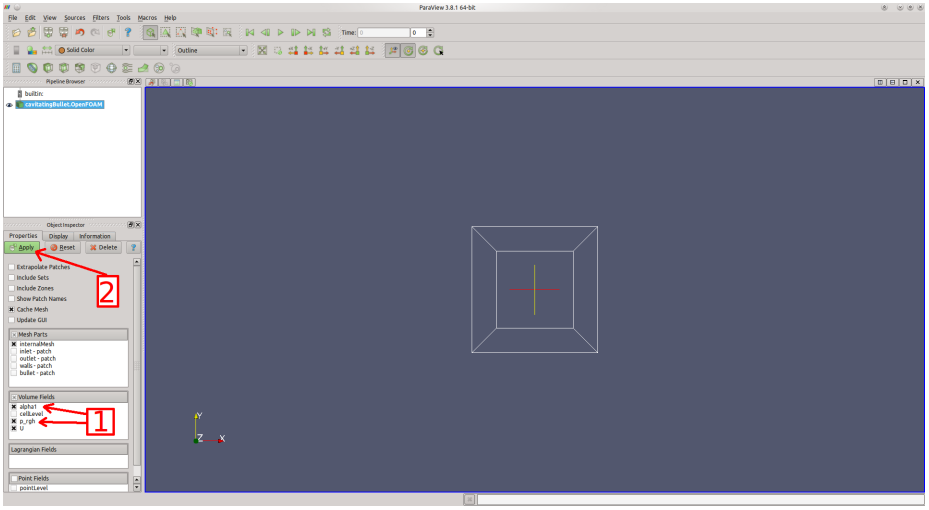
That was all the copying. Now we need to clean up and ajust the files to the new name.

Remove files coming from the compilation of *interPhaseChangeFoam*.

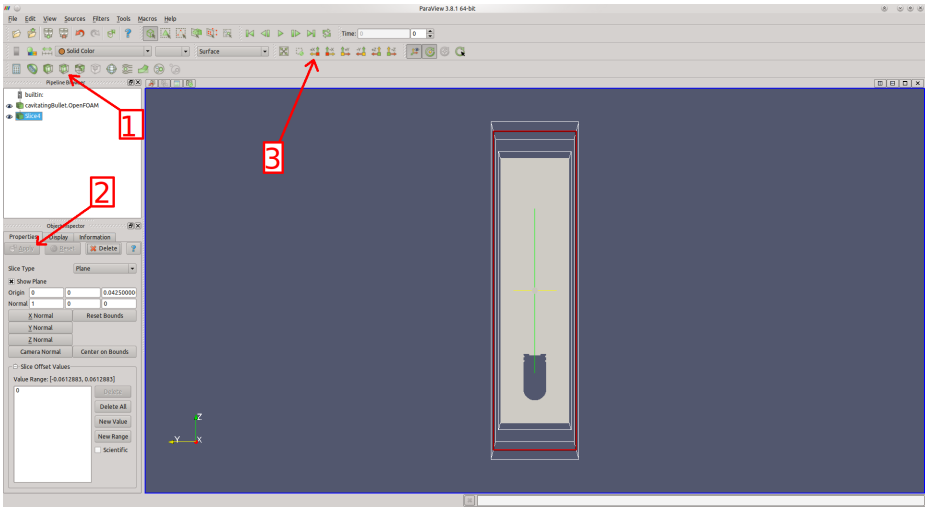
```
rm -r *.dep Make/linux*
wclean
```

Edit the build files to fit the *myInterPhaseChangeFoam* solver. The file *Make/files* shall look like this:

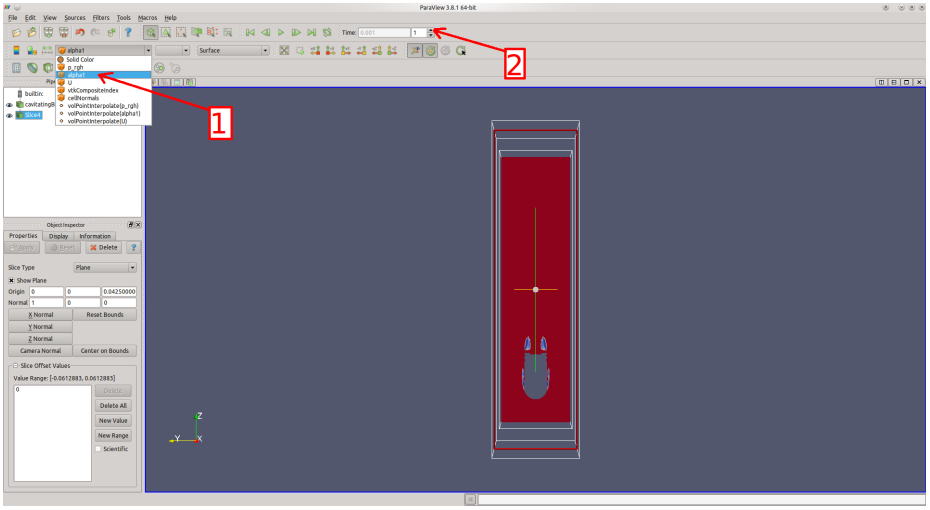
myInterPhaseChangeFoam.C
phaseChangeTwoPhaseMixtures/phaseChangeTwoPhaseMixture/phaseChangeTwoPhaseMixture.C
phaseChangeTwoPhaseMixtures/phaseChangeTwoPhaseMixture/newPhaseChangeTwoPhaseMixture.C
phaseChangeTwoPhaseMixtures/Kunz/Kunz.C
phaseChangeTwoPhaseMixtures/Merkle/Merkle.C
phaseChangeTwoPhaseMixtures/SchnerrSauer/SchnerrSauer.C



(a) Select the Volume Fields *alpha1* and *p_rgh*. Click *Apply*.



(b) Click *Slice*. Click *Apply*. Click *Set view direction to +X*



(c) Select the Volume Field *alpha1* from the drop down menu. Step one time step up

Figure 1: Screenshot

EXE = $(FOAM_USER_APPBIN)/myInterPhaseChangeFoam

Where the red text is the changes that be made from the original file. We are now ready to compile the solver

```
wmake
```

If everything worked correctly, the new solver binary should appear in the *FOAM_USER_APPBIN* directory. Check this with:

```
ls $FOAM_USER_APPBIN
```

We can now use the solver *myInterPhaseChangeFoam* on the *cavitatingBullet* case to verify that nothing have changed. This will be done in the next section.

3.3 Use the copied solver

```
cp -r $FOAM_TUTORIALS/multiphase/interPhaseChangeFoam/cavitatingBullet $FOAM_RUN/cavitatingBulletNew
cd $FOAM_RUN/cavitatingBulletNew
```

Edit the file *system/controlDict* so that *application* is changed from *interPhaseChangeFoam* to *myInterphaseChangeFoam*. Also change the *endTime* to 0.001.

```
./Allrun
```

Take a look at the result with paraView, as described in Section 3, to verify that it looks like the result from *interPhaseChangeFoam*.

4 Add a Temperature Field to the Solver

The fires step in adding temperature dependent evaporation to the solver is to include temperature in the solver. In this toturial we will only add a comon temperature transport constants for the two phases liquid and vapor, it is left as an assignment for the reader to implement separate constants for the two. Here we follow the guide from http://www.openfoamwiki.net/index.php/How_to_add_temperature_to_icoFoam.

4.1 Adding the temperature field

First return to the solver directory.

```
cd $WM_PROJECT_USER_DIR/applications/solvers/multiphase/myInterPhaseChangeFoam
```

Edit the file *createField.H* by adding the following to the top of the file:

```
    Info<< "Reading transportProperties\n" << endl;
    IOdictionary transportPropertiesDict
    (
        IOobject
        (
            "transportProperties",
            runTime.constant(),
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    );
    dimensionedScalar DT
    (
        transportPropertiesDict.lookup("DT")
    );

    Info<< "Reading field T\n" << endl;
    volScalarField T
    (
        IOobject
        (
            "T",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
```

FIXME: the first 12 lines may not be needed. The transportProperties file are read already newPhaseChangeTwoPhaseMixture. But at the moment I cannot find an elegant way of extracting DT through newPhaseChangeTwoPhaseMixture.

The first 16 lines are a new transport property related to the thermal diffusion which will be denoted as DT. The last 13 lines are the creation of the temperature field, T.

4.2 Adding a new equation to solve

After adding the temperature field next step is to add an equation describing the temperature transport. The temperature transport depends on the velocity field. Therefore the following equation are to be included after the Pressure-velocity PIMPLE corrector loop, but before *runTime.write();* in *myInterPhaseChangeFoam.C*.

```
fvScalarMatrix TEqn
(
    fvm::ddt(T)
    + fvm::div(phi, T)
    - fvm::laplacian(DT, T)
);
TEqn.solve();
```

This equation make use of the face flux variable, phi, which is already used in the momentum equation solution.

Thats all it takes to add a temperature field with a common temperature constant for the water and vapor. After saving the files, compile the solver using wmake.

```
wmake
```

5 Add Temperature field to a Test Case

Make a new copy of the *cavitatingBullet* case, and change the solver to *myInterPhaseChangeFoam* in the *controlDict*:

```
cp -r  $FOAM_TUTORIALS/multiphase/interPhaseChangeFoam/cavitatingBullet $FOAM_RUN/temperatureTest
cd $FOAM_RUN/temperatureTest
sed -i s/interPhaseChangeFoam/myInterPhaseChangeFoam/g system/controlDict
```

This time, to reduce the computional time, the "bullet" will be removed. This is done simply by removing the line about snappyHexMesh in the file *Allrun*. The file should now look like this:

```
#!/bin/sh
cd ${0%/*} || exit 1    # run from this directory

# Source tutorial run functions
. $WM_PROJECT_DIR/bin/tools/RunFunctions

# Generate the base block mesh
runApplication blockMesh

# Run the solver
runApplication ‘getApplication‘
```

The files related to the "bullet" can also be removed (`rm -r constant/triSurface system/snappyHexMeshDict`) but they do no harm.

To include the temperature into the test case the constant DT and an initial temperature field needs to be added. First add DT by including the following line the the file *constant/transportProperties*

```
DT            DT [0 2 -1 0 0 0 0] 0.002;   // [m^2/s]
```

The units of DT is m^2/s, but the value is not importent at this point.

Now add an initial temperature field. The easiest way is to copy an existing field and adjust it.

```
cp 0/p_rgh 0/T
```

The initial temperature field will be 90°C and the inlet will be a heat source of 110°C. Edit the file *0/T* to look like this:

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

dimensions      [0 0 0 1 0]; // [K]

internalField   uniform 363.15; //90 degrees Celsius

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 383.15; //110 degrees Celsius
    }

    outlet
    {
        type            fixedValue;
        value           $internalField;
    }

    walls
    {
        type            symmetryPlane;
    }
}
```

Temperature field and the transport constant for the solver is in place, but the solver do not know which discretization schemes to use for the new solver. Open the file *system/fvSchemes* and include two lines so that the *divSchemes* and the *laplacianSchemes* sections includes the the following:

```
divSchemes
{
...
 div(phi,T)           Gauss upwind;
}

laplacianSchemes
{
...
 laplacian(DT,T)      Gauss linear corrected;
}
```

Next, open the fvSolution dictionary and add the T:

```
solvers
{
    T
    {
        solver            BICCG;
        preconditioner    DILU;
        tolerance         1e-7;
        relTol            0;
    };
...
}
```

Now run the test case and look at the temperature field in paraView.

```
./Allrun
paraFoam
```

Have a look at the temperature distribution in paraView by following the guide in Figure 2.

6 Add Temperature Dependence to Phase Change

The solver now includes a temperature field and a we are now ready to make a temperature dependent phase change algorithm. To keep it simple in this tutorial the cavating algorithm Merkel already implemented in the *interPhaseChangeFoam* solver will be modified to be temperature dependent. As described in Section 2 a constant saturation pressure is used to determing if the water evaporates (or vapor condensate). In this section this static saturation pressure will be replaced by a temperature dependent function.

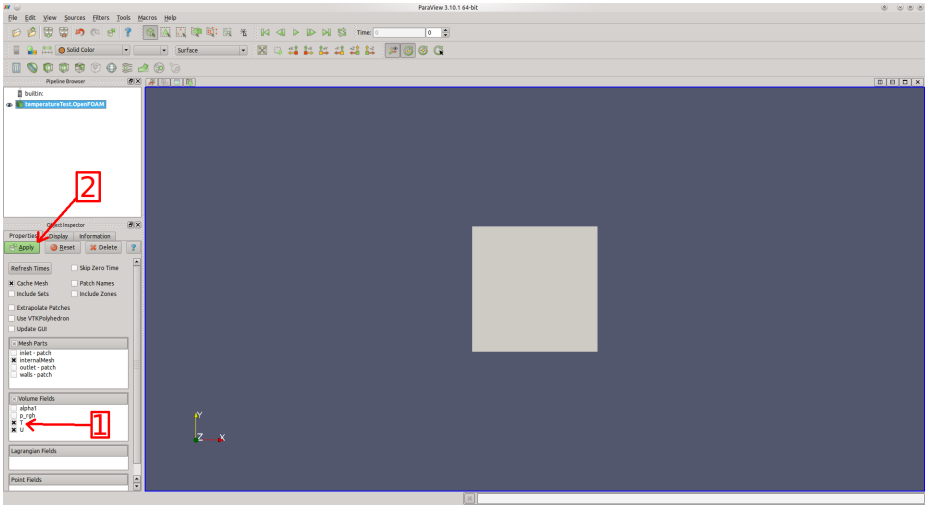
As the boiling point depends on both the pressure and the temperature this modification should be fine, however the algorithm is optimized for cavation at a constant temperature, therefore this modification might not be valid.

The temperature dependensi of the saturation pressure to be implemented here are the *August-Roche-Magnus formula.*

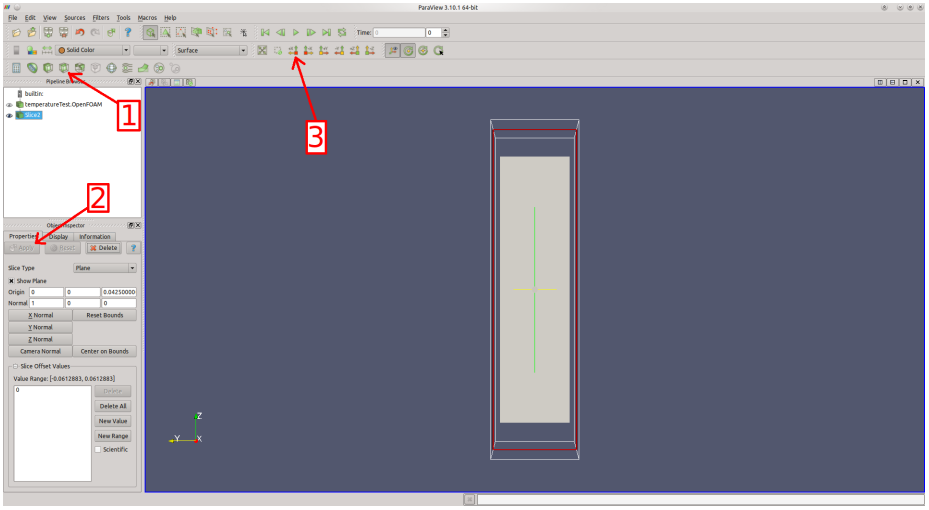
$$\begin{aligned} p_{\text{Sat}} &\approx 610.94 \cdot \exp\left(\frac{17.625 \cdot T_{\text{celsius}}}{T_{\text{celsius}} + 243.04}\right) \\ &= 610.94 \cdot \exp\left(\frac{17.625 \cdot (T - 273.15)}{T - 273.15 + 243.04}\right) \\ &= 610.94 \cdot \exp\left(\frac{17.625 \cdot (T - 273.15)}{T - 30.11}\right) \end{aligned} \tag{3}$$

where:
T is the temperature messured in Kelvin.

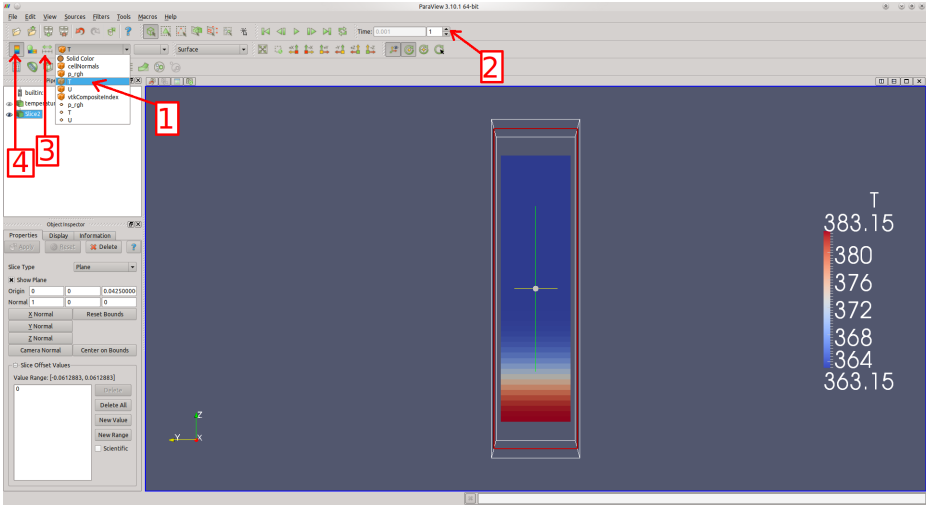
This in a non-linear equation. The saturation pressure is used in two loops in the solver. To speed-up the execution, the implementation will include a new field for the saturation pressure. This will be created so that the saturation pressure only is calculated when the temperature field in updated, and not every time the saturation pressure is used in the solver.



(a) Select the Volume Field *T*. Click *Apply*



(b) Click *Slice*. Click *Apply*. Click *Set view direction to +X*



(c) Select the Volume Field *alpha1* from the drop down menu. Step one time step up. Click *Rescale to Data Range*. Click *Toggle Color Legend Visibility*.

Figure 2: Screenshot

6.1 Create a Saturation Pressure Field

First the *August-Roche-Magnus formula* will be implemented in a source file. Create a file named *calcPSatField.H* and include the following in the file:

```
{
    const dimensionedScalar t30_11("30.11", dimensionSet(0,0,0,1,0,0,0), 30.11);
    const dimensionedScalar t273_15("273.15", dimensionSet(0,0,0,1,0,0,0), 273.15);
    const dimensionedScalar t1("1", dimensionSet(0,0,0,1,0,0,0), 1);
    const dimensionedScalar p610_94("610.94", dimensionSet(1,-1,-2,0,0,0,0), 610.94);
        // dimensionSet( [kg], [m], [s], [K], [kg*mol], [A], [cd]), [kg/(m*S^2)]=[Pa]


    // August-Roche-Magnus formula
    pSatField = p610_94 * exp(  17.625*(T-t273_15) / max(t1, T-t30_11) );
        //max(1,...) is included to avoid problems with devision by 0
}
```

The field *pSatField* is not defined yet, but it will be done now. Open the file *createFields.H* and remove the line

```
const dimensionedScalar& pSat = twoPhaseProperties->pSat();
```

And include this the following lines in file after *volScalarField T* and *volScalarField p_rgh* but before *Creating phaseChangeTwoPhaseMixture*

```
    volScalarField pSatField
    (
        IOobject
        (
            "pSatField",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        p_rgh  // initial value will be overwritten by calcPSatField.H
    );
    #include "calcPSatField.H"
```

FIXME: are there a better way to create the field than to initiate it to be equal to *p_rgh*?

Also update the new field each time the temperature field is re-calculated. Include the line

```
#include "calcPSatField.H"
```

In *myInterPhaseChangeFoam.C* after *TEqn.solve();*.

6.2 Remove the Static Saturation Pressure Variable

In the folder *phaseChangeTwoPhaseMixtures/phaseChangeTwoPhaseMixture/* open the file *phaseChangeTwoPhaseMixture.H* and remove the lines

```
 //- Saturation vapour pressure
dimensionedScalar pSat_;
```

And remove the lines

```
//- Return const-access to the saturation vapour pressure
const dimensionedScalar& pSat() const
{
    return pSat_;
}
```

And instead insert the following lines:

```
//- Return const-access to the saturation vapour pressure
const volScalarField& pSat() const
{
    const volScalarField& pSat = alpha1_.db().lookupObject<volScalarField>("pSat");
    return pSat;
}
```

In the same folder open the file *phaseChangeTwoPhaseMixture.C* and remove everything corresponding to *pSat*. That is remove the third line in following, and also remove the comma at the end of the line just above

```
:
    twoPhaseMixture(U, phi, alpha1Name),
    phaseChangeTwoPhaseMixtureCoeffs_(subDict(type + "Coeffs")),
    pSat_(lookup("pSat"))
{}
```

And remove the line

```
lookup("pSat") >> pSat_;
```

6.3 Compile and Enjoy

The saturation pressure will now be calculated from the temperature, making the phase change temperature dependent. Clean up and compile the solver.

```
wclean
wmake
```

The solver can be tested using the test case from Section 5. Or by downloading the test case. The temperatur distribution and coresponding liquid/vapor distribution can be seen in Figure 3 and 4.

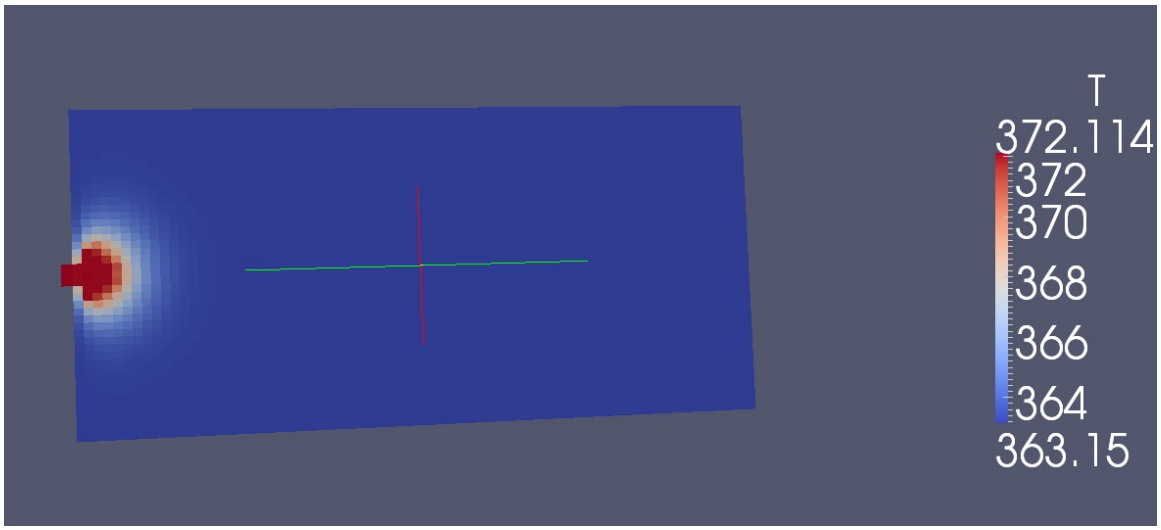


Figure 3: Temperature

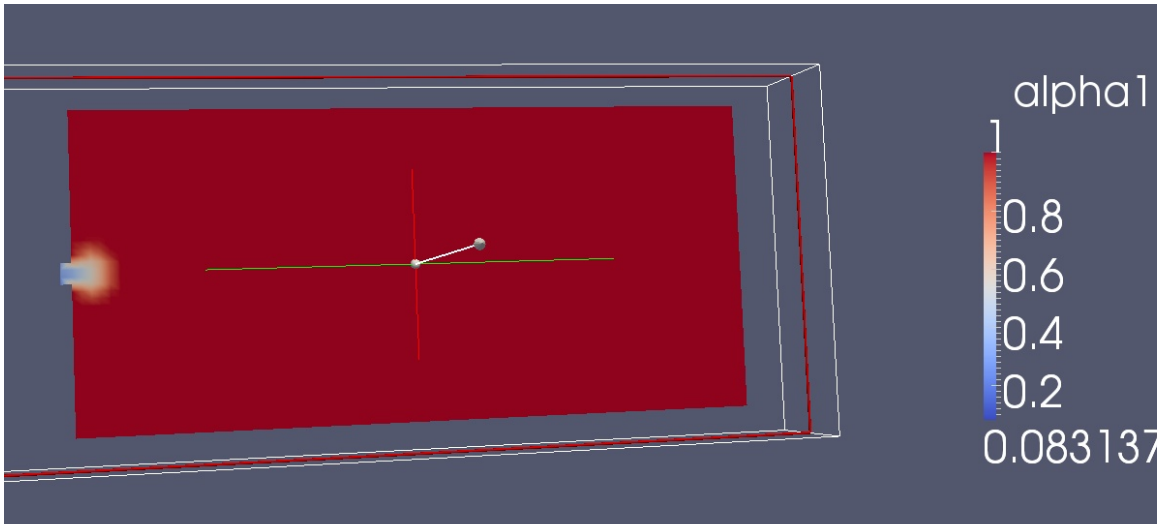


Figure 4: Alpha