

Cite as: Hummel, D.: Modifying buoyantPimpleFoam for the Simulation of Solid-Liquid Phase Change with Temperature-dependent Thermophysical Properties. In Proceedings of CFD with OpenSource Software, 2017, Edited by Nilsson. H.,  
[http://dx.doi.org/10.17196/OS\\_CFD#YEAR\\_2017](http://dx.doi.org/10.17196/OS_CFD#YEAR_2017)

## CFD WITH OPENSOURCE SOFTWARE

A COURSE AT CHALMERS UNIVERSITY OF TECHNOLOGY  
TAUGHT BY HÅKAN NILSSON

---

# Modifying buoyantPimpleFoam for the Simulation of Solid-Liquid Phase Change with Temperature-dependent Thermophysical Properties

---

Developed for OpenFOAM 1706+

*Author:*  
Daniel HUMMEL

*Peer reviewed by:*  
Manohar KAMPILI  
Ebrahim GHAHRAMANI

Licensed under CC-BY-NC-SA, <https://creativecommons.org/licenses/>

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

December 22, 2017

# Learning outcomes

The reader will learn:

- How to modify the solver buoyantPimpleFoam for the simulation of solid-liquid phase change
- How to modify thermophysical models to read thermophysical properties from tables
- How to setup a case with the modified solver and thermophysical models

# Prerequisites

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

- Basic thermodynamics with respect to solid-liquid phase change phenomena
- The enthalpy method for the simulation of solid-liquid phase change, Voller [9] provides a thorough overview with respect to derivation and implementation of the method
- The basics behind the implementation of tabulated thermophysical properties in OpenFOAM, Choquet [2] gives detailed instructions

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
<b>2</b>	<b>Theoretical background</b>	<b>7</b>
2.1	The Stefan Problem . . . . .	7
2.2	Enthalpy Method . . . . .	7
2.3	Temperature-dependence of Thermophysical Properties . . . . .	9
<b>3</b>	<b>Creating the Solver solidLiquidPhaseChangeFoam</b>	<b>11</b>
3.1	Copying and Renaming the Solver . . . . .	11
3.2	Modify Momentum Equation . . . . .	11
3.3	Modify Energy Equation . . . . .	12
3.4	Modify createFields . . . . .	13
3.5	Compile the solver . . . . .	15
<b>4</b>	<b>Thermophysical Properties from Tables</b>	<b>16</b>
4.1	General Procedure . . . . .	16
4.2	Interpolation Method . . . . .	17
4.3	Preparation . . . . .	17
4.3.1	Copy basic and specie folder . . . . .	17
4.3.2	Adapt compilation path . . . . .	18
4.3.3	Modify specie/Make/options . . . . .	18
4.4	Transport . . . . .	18
4.4.1	Copy and Rename . . . . .	18
4.4.2	Replace all Instances and Instantiated Type Name . . . . .	18
4.4.3	Modify tableTransportI.H . . . . .	18
4.4.4	Create Interpolation Files . . . . .	19
4.5	Equation of State . . . . .	20
4.5.1	Copy and Rename . . . . .	20
4.5.2	Replace all Instances . . . . .	20
4.5.3	Modify rhoTableI.H . . . . .	20
4.5.4	Create Interpolation Files . . . . .	21
4.6	Thermo . . . . .	22
4.6.1	Copy and Rename . . . . .	22
4.6.2	Replace all Instances and Instantiated Type Name . . . . .	22
4.6.3	Modify hTableThermoI.H . . . . .	22
4.6.4	Create Interpolation Files . . . . .	23
4.7	Modify Temperature Calculation . . . . .	24
4.7.1	Modify thermoI.H . . . . .	24
4.7.2	Create Interpolation Files . . . . .	25
4.8	Declaration of the New Thermophysical Models . . . . .	26
4.8.1	Declaration in thermoPhysicsTypes.H . . . . .	26

4.8.2	Declaration in rhoThermos.H . . . . .	26
4.8.3	Add new model combination to rhoThermos.C . . . . .	26
4.9	Copy Data Files . . . . .	27
4.10	Compile . . . . .	27
<b>5</b>	<b>Tutorial Setup, Run and Post-Processing</b>	<b>28</b>
5.1	Copy Tutorial Files . . . . .	28
5.2	Modify Domain Dimensions and Mesh Resolution . . . . .	28
5.3	Modify Patches . . . . .	29
5.4	Create Mesh . . . . .	29
5.5	Modify Initial and Boundary Conditions . . . . .	29
5.6	Modify Turbulence Properties . . . . .	30
5.7	Modify Thermophysical Properties and Gravity . . . . .	30
5.8	Modify Simulation Settings . . . . .	30
5.9	Add Function Object . . . . .	30
5.10	Add phaseChangeProperties . . . . .	31
5.11	Run Tutorial . . . . .	31
5.12	View Results . . . . .	31
<b>6</b>	<b>Study questions</b>	<b>35</b>

# 1 Introduction

Solid-liquid phase change processes occur in several technical applications such as latent heat thermal energy storages. Latent heat thermal energy storages are an attempt in harvesting the energy that is absorbed during melting and released during freezing. As opposed to sensible heat, the latent heat of a substance (from Latin *latentem*: lying hid, concealed) has no or little effect on the temperature of the substance. Latent heat thermal energy storages bear the potential of higher energy densities when compared to conventional thermal energy storages. For detailed analysis and optimization of such systems, computational models are needed.

This tutorial describes how to pre-process, run and post-process a case involving a material in an enclosure undergoing a solid-liquid phase change. It describes how to copy and modify the solver `buoyantPimpleFoam`. In order to account for the temperature-dependence of the thermophysical properties of the phase change material, thermophysical models are modified in a way that the property values are interpolated from tables.

## 2 Theoretical background

### 2.1 The Stefan Problem

Despite earlier works from different researchers, the mathematical description of solid-liquid phase change problems is predominantly associated with Josef Stefan, also known for his contributions to heat radiation problems.

In the late 19th century he attempted to find analytical solutions to ice growth rate data taken during polar expeditions. His paper about ice formation in the polar seas [7] has drawn the most attention. [10]

In a first model it is assumed that the transport of heat through the ice occurs rapidly. At the air-ice interface the temperature is equally  $\varpi$  degrees below the freezing temperature, whereas at the ice-water interface it is equal to the freezing temperature. The heat removed from the water during a time  $\delta t$  equals the amount of heat required to freeze an amount  $\delta d$  of water. Thus,

$$\frac{k\varpi}{d}\delta t = \rho L \delta d \quad (2.1)$$

where  $k$  is the thermal conductivity of ice,  $d$  is the thickness of the ice layer,  $\rho$  is the density of ice and  $L$  is the latent heat of fusion. In a second model, transient transport of heat in the ice is taken into account. In the ice layer, the temperature difference satisfies the partial differential equation

$$c\rho \frac{\partial T}{\partial t} = k \frac{\partial^2 T}{\partial x^2}, \quad 0 < x < d(t), \quad 0 < t \quad (2.2)$$

where  $c$  is the specific heat of ice,  $T$  is the temperature and  $x$  is the depth below the ice surface. With the initial and boundary conditions

$$d(0) = 0, \quad T(0, t) = f(t), \quad T(d(t), t) = 0 \quad (2.3)$$

and

$$\rho L \frac{\partial d}{\partial t}(t) = -k \frac{\partial T}{\partial x}(d(t), t), \quad 0 < t \quad (2.4)$$

Equations 2.2 and 2.4 are considered the original Stefan Problem [10], where Equation 2.4 is often referred to as Stefan condition.

### 2.2 Enthalpy Method

The idea that lead to the development of the enthalpy method was to reformulate the Stefan problem in a way that the Stefan condition is implicitly bound up in new equations, applicable to the whole computational domain. Prominent contributions have been published

by Voller and co-authors, see Voller [9] for instance.

The enthalpy of a pure substance with distinct melting temperature can be approximated by a piecewise linear function

$$H = \begin{cases} c_s T & T < T_m \\ c_l + (c_s - c_l)T_m + L & T > T_m \end{cases} \quad (2.5)$$

where the respective specific heat capacities in the solid and liquid regions  $c_s$  and  $c_l$  are assumed to be constant. By expressing the one-dimensional transient heat equation in terms of enthalpy, the two involving heat equations of the original Stefan Problem formulation are reduced to

$$\rho \frac{\partial H}{\partial t} = k \nabla^2 T \quad (2.6)$$

By defining the liquid volume fraction  $g_l$  as step function

$$g_l = \begin{cases} 0 & T < T_m \\ 1 & T > T_m \end{cases} \quad (2.7)$$

the enthalpy can be expressed by

$$H = (1 - g_l) \int_{T_{\text{ref}}}^T \rho c_s dT + g_l \int_{T_{\text{ref}}}^T \rho c_l dT + \rho g_l L. \quad (2.8)$$

By substituting Equation 2.8 into Equation 2.6, the nonlinear behaviour associated with the phase change can be isolated into a source term (refer to Voller [9] for details)

$$\rho c \frac{\partial T}{\partial t} = k \nabla^2 T - L \frac{\partial(\rho g_l)}{\partial t} \quad (2.9)$$

In phase change problems that involve fluid flow, the conservation of mass, momentum and energy has to be considered – the set of equations to be solved then reads

$$\frac{\partial \rho}{\partial t} + \nabla(\rho \vec{u}) = 0 \quad (2.10)$$

$$\frac{\partial \rho \vec{u}}{\partial t} + \nabla(\rho \vec{u} \vec{u}) = -\nabla P + \nabla \tau_{ij} + \vec{\Xi}, \quad (2.11)$$

$$\frac{\partial \rho c T}{\partial t} + \nabla(\rho \vec{u} c T) = \nabla \left( \frac{k}{\rho c} \nabla(c T) \right) - L \frac{\partial(\rho g_l)}{\partial t} \quad (2.12)$$

In the momentum equation, the source term

$$\vec{\Xi} = \vec{\Gamma} + Z \vec{u} \quad (2.13)$$

contains the buoyancy term  $\vec{\Gamma}$ . For the assumption of a constant density but yet the consideration of natural convection due to temperature differences, the Boussinesq approximation is often applied. With  $\vec{g}$  being the gravitational acceleration and  $\beta$  being the thermal expansion coefficient, this buoyancy is expressed by a linear function of the deviation of temperature



$$\vec{\Gamma} = \rho \vec{g} b (T - T_{\text{ref}}) \quad (2.14)$$

where  $\mu$  is the dynamic viscosity and  $P$  denotes the pressure. If a temperature-dependent density is considered there is no need for the Boussinesq approximation and the buoyancy force due to density variations can be expressed by

$$\vec{\Gamma} = \vec{g}(\rho - \rho_{\text{ref}}) \quad (2.15)$$

Besides the buoyancy source term, the source term in the momentum equation comprises an additional function

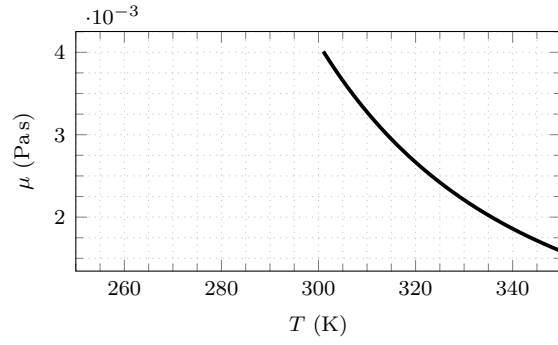
$$Z = C(1 - g_l) \quad (2.16)$$

where constant  $C$  has an arbitrary large value. Multiplied by the velocity vector, the velocity treatment function  $Z$  becomes large if the liquid volume fraction tends to zero and thereby forces the velocity in the solid to zero. For isothermal phase change problems, Brent et al. [1] suggest the simple form as given by Equation 2.16.

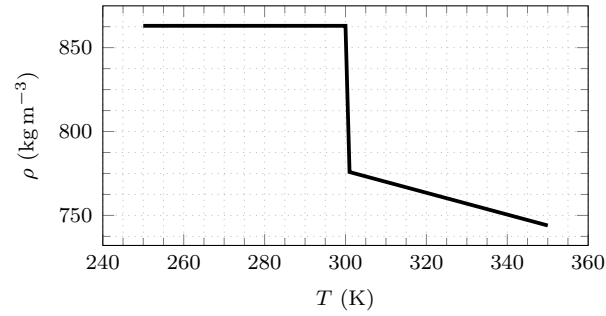
## 2.3 Temperature-dependence of Thermophysical Properties

The thermophysical properties of a substance such as density, viscosity, specific heat capacity and thermal conductivity are often assumed to be constant for simplicity. However, the thermophysical properties of many phase change materials exhibit a strong dependence on temperature. When a phase change occurs, a jump in thermophysical properties can be observed. The temperature-dependence of thermophysical properties is shown for n-octadecane ( $\text{C}_{18}\text{H}_{38}$ ), a paraffin that is frequently used as phase change material.

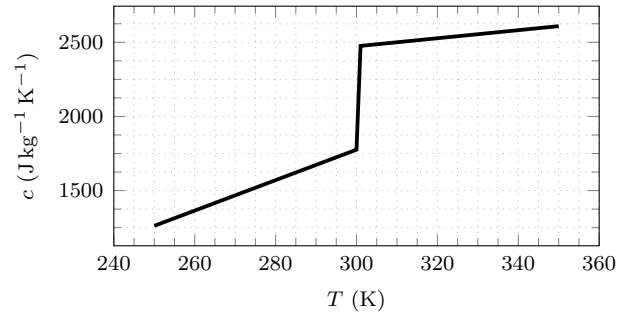
The data shown in Figure 2.1 was taken from different experimental publications ([3], [5], [6] and [8]). The experimental values were interpolated and simplified to enable a description by simple functions. In addition, the intercepts of the specific heat capacity functions (obtained from Marano et al. [5]) were modified so that the resulting values for the latent heat of fusion approximate literature data (e.g. Vélez et al. [8]:  $L = 243.68 \cdot 10^3 \text{ J kg}^{-1}$ ). It is assumed that phase change occurs at a temperature of 300 K.



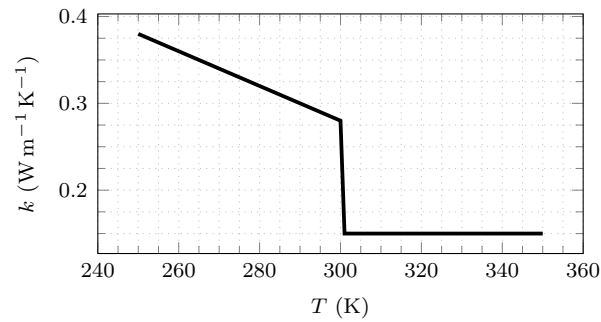
(a) Dynamic viscosity (values originally from Ducoulombier et al. [3], modified)



(b) Density (values originally from Seyer et al. [6] and Vélez et al. [8], modified)



(c) Specific heat capacity (values originally from Marano et al. [5], modified)



(d) Thermal conductivity (values originally from Vélez et al. [8], modified)

Figure 2.1: Variation of thermophysical properties of n-octadecane with temperature

## 3 Creating the Solver

### solidLiquidPhaseChangeFoam

The solver `buoyantPimpleFoam` is a transient solver for buoyant, turbulent flow of compressible fluids for ventilation and heat-transfer. It is close to the desired set of equations (see Equations 2.10–2.12) and therefore constitutes a good basis for the new solver `solidLiquidPhaseChangeFoam`, a solver for the simulation of isothermal solid-liquid phase change with temperature-dependent thermophysical properties. The main difference in comparison to Equations 2.10–2.12 is that the energy equation is based on enthalpy instead of temperature.

#### 3.1 Copying and Renaming the Solver

The entire directory of `buoyantPimpleFoam` is copied to the user directory and renamed to `solidLiquidPhaseChangeFoam`. After renaming the main `.C` file and the entry in `Make/files` accordingly, the path where the executable is compiled is changed to the user directory.

```
cd $WM_PROJECT_USER_DIR

mkdir --parents applications/solvers/heatTransfer

cp -r $FOAM_SOLVERS/heatTransfer/buoyantPimpleFoam $WM_PROJECT_USER_DIR/applications/solvers/heatTransfer/

cd $WM_PROJECT_USER_DIR/applications/solvers/heatTransfer/

mv buoyantPimpleFoam solidLiquidPhaseChangeFoam

cd solidLiquidPhaseChangeFoam

mv buoyantPimpleFoam.C solidLiquidPhaseChangeFoam.C

sed -i s/"buoyantPimpleFoam"/"solidLiquidPhaseChangeFoam"/g Make/files

sed -i s/"FOAM_APPBIN"/"FOAM_USER_APPBIN"/g Make/files

wclean

wmake
```

#### 3.2 Modify Momentum Equation

The buoyancy force and velocity treatment source terms are added to the left hand side of the momentum equation. Added lines are marked by `// added` in the listings.

```
vi UEqn.H
```

Listing 3.1: UEqn.H

```
1 // Solve the Momentum equation
2
3 MRF.correctBoundaryVelocity(U);
4
5 fvVectorMatrix UEqn
6 (
7     fvm::ddt(rho, U) + fvm::div(phi, U)
8     + MRF.DDt(rho, U)
9     + turbulence->divDevRhoReff(U)
```

```

10     + g*(rho-rhoRef) // added
11     + fvm::Sp(C*pow(1-gl, 2)/(pow(gl, 3)+q),U) // added
12     ==
13     fvOptions(rho, U)
14 );
15
16 UEqn.relax();
17
18 fvOptions.constrain(UEqn);
19
20 if (pimple.momentumPredictor())
21 {
22     solve
23     (
24         UEqn
25         ==
26         fvc::reconstruct
27         (
28             (
29                 - ghf*fvc::snGrad(rho)
30                 - fvc::snGrad(p_rgh)
31             ) * mesh.magSf()
32         )
33     );
34
35     fvOptions.correct(U);
36     K = 0.5*magSqr(U);
37 }

```

### 3.3 Modify Energy Equation

The enthalpy source term is added to the right hand side of the energy equation. The liquid volume fraction update is added to the bottom of the energy equation. In order to enable reading of specific heat capacity fields, the values from the thermophysical model are stored in a new variable. The liquid volume fraction and bounding, as described by Voller [9], is added. The specific heat capacity entry is placed outside of the loop. In order to converge the energy equation and the liquid fraction, the energy equation iterated ten times by adding a simple for-loop at the top of the energy equation.

```
vi EEqn.H
```

Listing 3.2: EEqn.H

```

1  for (int i=0; i<10; i++ ) // added
2  {
3      iter++; // added
4
5      volScalarField& he = thermo.he();
6
7      fvScalarMatrix EEqn
8      (
9          fvm::ddt(rho, he) + fvm::div(phi, he)
10         + fvc::ddt(rho, K) + fvc::div(phi, K)
11         + (
12             he.name() == "e"
13             ? fvc::div
14             (
15                 fvc::absolute(phi/fvc::interpolate(rho), U),
16                 p,
17                 "div(phi,p)"
18             )
19             : -dPdt
20         )
21         - fvm::laplacian(turbulence->alphaEff(), he)
22         ==
23         rho*(U&g)
24         - L*(fvc::ddt(rho, gl)) // added
25         + radiation->Sh(thermo, he)
26         + fvOptions(rho, he)
27     );
28
29     EEqn.relax();
30
31     fvOptions.constrain(EEqn);
32
33     EEqn.solve();
34
35     fvOptions.correct(he);
36 }

```

```

37 thermo.correct();
38 radiation->correct();
39
40 volScalarField glNew = gl + glRelax*(thermo.T()-Tmelt)*thermo.Cp()/L; // added
41 gl = max(scalar(0), min(glNew, scalar(1))); // added
42 }
43 Cp = thermo.Cp(); // added

```

## 3.4 Modify createFields

Fields for the liquid volume fraction and the specific heat (only for post-processing purposes) are created. Furthermore, all phase-change related constants are read from the new dictionary phaseChangeProperties.

```
vi createFields.H
```

Listing 3.3: createFields.H

```

1 // No changes until explicitly stated
2
3 Info<< "Reading_thermophysical_properties\n" << endl;
4
5 autoPtr<rhoThermo> pThermo(rhoThermo::New(mesh));
6 rhoThermo& thermo = pThermo();
7 thermo.validate(args.executable(), "h", "e");
8
9 volScalarField rho
10 (
11     IOobject
12     (
13         "rho",
14         runtime.timeName(),
15         mesh,
16         IOobject::NO_READ,
17         IOobject::NO_WRITE
18     ),
19     thermo.rho()
20 );
21
22 volScalarField& p = thermo.p();
23
24 Info<< "Reading_field_U\n" << endl;
25 volVectorField U
26 (
27     IOobject
28     (
29         "U",
30         runtime.timeName(),
31         mesh,
32         IOobject::MUST_READ,
33         IOobject::AUTO_WRITE
34     ),
35     mesh
36 );
37
38 #include "compressibleCreatePhi.H"
39
40
41 Info<< "Creating_turbulence_model\n" << endl;
42 autoPtr<compressible::turbulenceModel> turbulence
43 (
44     compressible::turbulenceModel::New
45     (
46         rho,
47         U,
48         phi,
49         thermo
50     )
51 );
52
53
54 #include "readGravitationalAcceleration.H"
55 #include "readhRef.H"
56 #include "gh.H"
57
58
59 Info<< "Reading_field_p_rgh\n" << endl;
60 volScalarField p_rgh
61 (
62     IOobject
63     (
64         "p_rgh",

```

```

65     runTime.timeName(),
66     mesh,
67     IOobject::MUST_READ,
68     IOobject::AUTO_WRITE
69 ),
70 mesh
71 );
72
73 // Force p_rgh to be consistent with p
74 p_rgh = p - rho*gh;
75
76 mesh.setFluxRequired(p_rgh.name());
77
78 label pRefCell = 0;
79 scalar pRefValue = 0.0;
80
81 if (p_rgh.needReference())
82 {
83     setRefCell
84     (
85         p,
86         p_rgh,
87         pimple.dict(),
88         pRefCell,
89         pRefValue
90     );
91
92     p += dimensionedScalar
93     (
94         "p",
95         p.dimensions(),
96         pRefValue - getRefCellValue(p, pRefCell)
97     );
98 }
99
100 dimensionedScalar initialMass("initialMass", fvc::domainIntegrate(rho));
101
102 #include "createDpdt.H"
103
104 #include "createK.H"
105
106 #include "createMRF.H"
107 #include "createRadiationModel.H"
108
109 // Added from here to end of file
110
111 volScalarField gl
112 (
113     IOobject
114     (
115         "gl",
116         runTime.timeName(),
117         mesh,
118         IOobject::MUST_READ,
119         IOobject::AUTO_WRITE
120     ),
121     mesh
122 );
123
124 IOdictionary phaseChangeProperties
125 (
126     IOobject
127     (
128         "phaseChangeProperties",
129         runTime.constant(),
130         mesh,
131         IOobject::MUST_READ_IF_MODIFIED,
132         IOobject::NO_WRITE
133     )
134 );
135
136 dimensionedScalar Tmelt
137 (
138     phaseChangeProperties.lookup("Tmelt")
139 );
140
141 dimensionedScalar L
142 (
143     phaseChangeProperties.lookup("L")
144 );
145
146 dimensionedScalar C
147 (
148     phaseChangeProperties.lookup("C")
149 );
150
151 dimensionedScalar glRelax
152 (
153     phaseChangeProperties.lookup("glRelax")
154 );
155
156 volScalarField Cp

```

```
157 (
158     IObject
159     (
160         "Cp",
161         runTime.timeName(),
162         mesh,
163         IObject::NO_READ,
164         IObject::AUTO_WRITE
165     ),
166     thermo.Cp()
167 );
```

## 3.5 Compile the solver

The new solver is compiled.

```
wclean
wmake
```

After finishing without any errors, the new solver can be executed. For a first test, the command

```
solidLiquidPhaseChangeFoam -help
```

should return the terminal output

```
Usage: solidLiquidPhaseChangeFoam [OPTIONS]
options:
  -case <dir> specify alternate case directory, default is the cwd
  -decomposeParDict <file>
                    read decomposePar dictionary from specified location
  -noFunctionObjects
                    do not execute functionObjects
  -parallel run in parallel
  -postProcess Execute functionObjects only
  -roots <(dir1 .. dirN)>
                    slave root directories for distributed running
  -srcDoc display source code in browser
  -doc display application documentation in browser
  -help print the usage
```

## 4 Thermophysical Properties from Tables

The implementation is mainly adopted from Choquet [2], who modified thermophysical models in OpenFOAM 2.3.x for the simulation of argon plasma. The implementation is extended by the implementation of a temperature-dependent viscosity and a temperature interpolation. As briefly discussed in Section 2.3, solid-liquid phase change processes can involve sharp jumps in the specific heat capacity; this can lead to similarly sharp jumps in the enthalpy-temperature relation. The Newton-Raphson solver for the temperature calculation (implemented in `specie/thermo/thermo/thermoI.H`) is therefore replaced by an interpolation from a temperature-enthalpy table.

### 4.1 General Procedure

The implementation requires that the original basic and specie folders (located in the main folder `thermophysicalModels`) are copied to the user directory. Three main changes are done in the specie folder, where the transport, equationOfState and thermo models are located. For the transport model, the model const is copied and modified, whereas the new equationOfState model is based on perfectGas. The new thermo model is derived from hConst. See Table 4.1 for a description of the original models that are used as the basis of the modifications described in this chapter.

Table 4.1: Description of the original models [4]

<b>Basic thermophysical properties – thermo</b>	
hConstThermo	Constant specific heat $c_p$ model with evaluation of enthalpy $h$ and entropy $s$
<b>Transport properties – transport</b>	
constTransport	Constant transport properties
<b>Equation of State – equationOfState</b>	
perfectGas	Perfect gas equation of state

After some preparatory steps (Section 4.3), the modification of const, perfectGas and hConst is similar. It involves the following steps:

1. Copy existing model and rename folders and files. (Section 4.4.1, 4.5.1, 4.6.1)
2. In the .C, .H and I.H files, replace all instances of
  - const with table,
  - perfectGas with rhoTable and
  - hConst with hTable,



respectively. For thermo and transport models, also replace the instantiated type names in the .H files (Section 4.4.2, 4.5.2, 4.6.2)

3. The member functions in the I.H files are modified (Section 4.4.3, 4.5.3, 4.6.3)
4. The interpolation files that are included in the I.H files are created. (Section 4.4.4, 4.5.4, 4.6.4)

The temperature is calculated from enthalpy in `specie/thermo/thermo/thermoI.H`. For the implementation of the temperature interpolation from tables, steps 3 and 4 also apply. See Section 4.7.1 and 4.7.2.

After declaring the new thermophysical model in `specie/include/thermoPhysicsTypes.H` (Section 4.8.1) and `basic/rhoThermo/rhoThermos.C` (Section 4.8.2), the table value files are inserted (Section 4.9). Finally, the implementation is completed by compiling both `specie` and `basic` libraries (Section 4.10).

## 4.2 Interpolation Method

All interpolations are done in a simple linear manner as suggested by Choquet [2]. Taking the example of the interpolation of dynamic viscosity from a viscosity-temperature table, first an index is computed by

$$i = \left\lfloor \left| \left( \frac{T - T_0}{dT} \right) \right| \right\rfloor \quad (4.1)$$

where  $T$  is the current temperature,  $T_0$  is the temperature that refers to the first viscosity value in the table and  $dT$  is the distance between the table values. In the next step, the temperature at the calculated index is computed by

$$T_{\text{tabulated}} = T_0 + i \cdot dT \quad (4.2)$$

Finally, the interpolation is achieved by

$$\mu = \mu_{\text{tabulated}}(i) + (\mu_{\text{tabulated}}(i+1) - \mu_{\text{tabulated}}(i)) \cdot \left( \frac{T - T_{\text{tabulated}}}{dT} \right) \quad (4.3)$$

where  $\mu_{\text{tabulated}}(i)$  is the  $i$ th value in the viscosity table.

## 4.3 Preparation

### 4.3.1 Copy basic and specie folder

First, the `basic` and `specie` folders are copied into the user directory.

```
cd $WM_PROJECT_USER_DIR
rm -r src/thermophysicalModels
mkdir --parents src/thermophysicalModels
cp -r $WM_PROJECT_DIR/src/thermophysicalModels/specie $WM_PROJECT_USER_DIR/src/thermophysicalModels/specie
cp -r $WM_PROJECT_DIR/src/thermophysicalModels/basic $WM_PROJECT_USER_DIR/src/thermophysicalModels/basic
cd $WM_PROJECT_USER_DIR/src/thermophysicalModels
```

### 4.3.2 Adapt compilation path

Analogous to the steps executed for the adaption of solver described in Section 3.1, the Make/files are adopted. The name of the executable is not changed as executables in the user directory are accessed in priority (see Choquet [2]).

```
sed -i s/"FOAM_LIBBIN"/"FOAM_USER_LIBBIN"/g basic/Make/files
sed -i s/"FOAM_LIBBIN"/"FOAM_USER_LIBBIN"/g specie/Make/files
```

### 4.3.3 Modify specie/Make/options

```
vi basic/Make/options
```

Delete the original path to the specie lnInclude

```
3 -I$(LIB_SRC)/thermophysicalModels/specie/lnInclude \
```

and add one that directs to the new specie model that will be created in the user project directory.

Listing 4.1: basic/Make/options

```
2 -I$(LIB_SRC)/transportModels/compressible/lnInclude \
3 -I$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/specie/lnInclude \
4 -I$(LIB_SRC)/thermophysicalModels/thermophysicalProperties/lnInclude \
```

In order to direct the compiler to the libraries in the user directory first, add the lines

Listing 4.2: basic/Make/options

```
9 LIB_LIBS = \
10 -L$(FOAM_USER_LIBBIN) \
```

## 4.4 Transport

### 4.4.1 Copy and Rename

```
cp -r specie/transport/const/ specie/transport/table
mv specie/transport/table/constTransport.C specie/transport/table/tableTransport.C
mv specie/transport/table/constTransport.H specie/transport/table/tableTransport.H
mv specie/transport/table/constTransportI.H specie/transport/table/tableTransportI.H
```

### 4.4.2 Replace all Instances and Instantiated Type Name

```
sed -i s/"constTransport"/"tableTransport"/g specie/transport/table/*
```

```
sed -i s/"const"/"table"/g specie/transport/table/tableTransport.H
```

### 4.4.3 Modify tableTransportI.H

```
vi specie/transport/table/tableTransportI.H
```

### Include muInterpolation.H

The interpolation procedure for dynamic viscosity, located in `muInterpolation.H` is included.

Listing 4.3: `specie/transport/table/tableTransportI.H`

```

82 template<class Thermo>
83 inline Foam::scalar Foam::tableTransport<Thermo>::mu
84 (
85     const scalar p,
86     const scalar T
87 ) const
88 {
89     //return mu_;
90     #include "muInterpolation.H"
91 }

```

### Include kappaInterpolation.H

The interpolation procedure for thermal conductivity, located in `kappaInterpolation.H` is included.

Listing 4.4: `specie/transport/table/tableTransportI.H`

```

94 template<class Thermo>
95 inline Foam::scalar Foam::tableTransport<Thermo>::kappa
96 (
97     const scalar p,
98     const scalar T
99 ) const
100 {
101     //return this->Cp(p, T)*mu(p, T)*rPr_;
102     #include "kappaInterpolation.H"
103 }

```

### Modify Calculation of alphah

The calculation of thermal diffusivity of enthalpy is modified.

Listing 4.5: `specie/transport/table/tableTransportI.H`

```

106 template<class Thermo>
107 inline Foam::scalar Foam::tableTransport<Thermo>::alphah
108 (
109     const scalar p,
110     const scalar T
111 ) const
112 {
113     //return mu(p, T)*rPr_;
114     return kappa(p,T)/this->Cpv(p,T);
115 }

```

## 4.4.4 Create Interpolation Files

The interpolation for dynamic viscosity and thermal conductivity is implemented according to Section 4.2.

### muInterpolation

```
vi specie/transport/table/muInterpolation.H
```

Listing 4.6: `specie/transport/table/muInterpolation.H`

```

1 int i_index;
2 scalar dT=1;
3 scalar T0=250;

```

```

4      scalar Temp_tabulated;
5      scalar mu_T_tabulated;
6
7      // Dynamic viscosity mus (Pa s):
8
9      scalar mu_tabulated[101]=
10     {
11     #include "mu.H"
12     };
13
14     // linear interpolation to calculate mu(T) (Pa s)
15     i_index = int(floor(fabs((T-T0)/dT)));
16     Temp_tabulated = T0+i_index*dT;
17     mu_T_tabulated = mu_tabulated[i_index]
18                   + (mu_tabulated[i_index+1]-mu_tabulated[i_index])*(T-Temp_tabulated)/dT;
19
20     return mu_T_tabulated;

```

## kappaInterpolation

```
vi specie/transport/table/kappaInterpolation.H
```

Listing 4.7: specie/transport/table/kappaInterpolation.H

```

1      int i_index;
2      scalar dT=1;
3      scalar T0=250;
4      scalar Temp_tabulated;
5      scalar kappa_T_tabulated;
6
7      // Thermal conductivity kappa (W/(m.K)), tabulated :
8
9      scalar kappa_tabulated[101]=
10     {
11     #include "kappa.H"
12     };
13
14     // linear interpolation to calculate kappa(T) W/(m.K)
15     i_index = int(floor(fabs((T-T0)/dT)));
16     Temp_tabulated = T0+i_index*dT;
17     kappa_T_tabulated = kappa_tabulated[i_index]
18                   + (kappa_tabulated[i_index+1]-kappa_tabulated[i_index])*(T-Temp_tabulated)/dT;
19
20     return kappa_T_tabulated;

```

## 4.5 Equation of State

### 4.5.1 Copy and Rename

```

cp -r specie/equationOfState/perfectGas/ specie/equationOfState/rhoTable
mv specie/equationOfState/rhoTable/perfectGas.C specie/equationOfState/rhoTable/rhoTable.C
mv specie/equationOfState/rhoTable/perfectGas.H specie/equationOfState/rhoTable/rhoTable.H
mv specie/equationOfState/rhoTable/perfectGasI.H specie/equationOfState/rhoTable/rhoTableI.H

```

### 4.5.2 Replace all Instances

```
sed -i s/"perfectGas"/"rhoTable"/g specie/equationOfState/rhoTable/*
```

### 4.5.3 Modify rhoTableI.H

```
vi specie/equationOfState/rhoTable/rhoTableI.H
```

## Include rhoInterpolation.H

The interpolation procedure for density, located in `rhoInterpolation.H` is included.

Listing 4.8: `specie/equationOfState/rhoTable/rhoTableI.H`

```

73 template<class Specie>
74 inline Foam::scalar Foam::rhoTable<Specie>::rho(scalar p, scalar T) const
75 {
76     //return p/(this->R()*T);
77     #include "rhoInterpolation.H"
78 }

```

## Modify psi calculation

The calculation for compressibility is modified. As incompressibility is assumed, it is set to zero.

Listing 4.9: `specie/equationOfState/rhoTable/rhoTableI.H`

```

102 template<class Specie>
103 inline Foam::scalar Foam::rhoTable<Specie>::psi(scalar p, scalar T) const
104 {
105     //return 1.0/(this->R()*T);
106     return 0;
107 }

```

## Modify Z calculation

The calculation for compressibility factor is modified. As incompressibility is assumed, it is set to zero.

Listing 4.10: `specie/equationOfState/rhoTable/rhoTableI.H`

```

110 template<class Specie>
111 inline Foam::scalar Foam::rhoTable<Specie>::Z(scalar p, scalar T) const
112 {
113     //return 1;
114     return 0;
115 }

```

## Modify CpMCv calculation

The calculation for specific heat capacity at constant pressure minus specific heat capacity at constant volume is modified. As incompressibility is assumed, it is set to zero.

Listing 4.11: `specie/equationOfState/rhoTable/rhoTableI.H`

```

118 template<class Specie>
119 inline Foam::scalar Foam::rhoTable<Specie>::CpMCv(scalar p, scalar T) const
120 {
121     //return this->R();
122     return 0;
123 }

```

## 4.5.4 Create Interpolation Files

### rhoInterpolation

The interpolation for density is implemented according to Section 4.2.

```
vi specie/equationOfState/rhoTable/rhoInterpolation.H
```

Listing 4.12: specie/equationOfState/rhoTable/rhoInterpolation.H

```

1  int i_index;
2  scalar dT=1;
3  scalar T0=250;
4  scalar Temp_tabulated;
5  scalar rho_T_tabulated;
6
7  // Density (kg/m^3), tabulated :
8
9
10 scalar rho_tabulated[101]=
11 {
12     #include "rho.H"
13 };
14
15 // linear interpolation to calculate rho(T) in kg/m^3
16 i_index = int(fabs(floor((T-T0)/dT)));
17 Temp_tabulated = T0+i_index*dT;
18 rho_T_tabulated = rho_tabulated[i_index]
19     + (rho_tabulated[i_index+1]-rho_tabulated[i_index])*(T-Temp_tabulated)/dT;
20
21 return rho_T_tabulated;

```

## 4.6 Thermo

### 4.6.1 Copy and Rename

```

cp -r specie/thermo/hConst/ specie/thermo/hTable

mv specie/thermo/hTable/hConstThermo.C specie/thermo/hTable/hTableThermo.C

mv specie/thermo/hTable/hConstThermo.H specie/thermo/hTable/hTableThermo.H

mv specie/thermo/hTable/hConstThermoI.H specie/thermo/hTable/hTableThermoI.H

```

### 4.6.2 Replace all Instances and Instantiated Type Name

```
sed -i s/"hConstThermo"/"hTableThermo"/g specie/thermo/hTable/*
```

```
sed -i s/"hConst<"/"hTable<"/g specie/thermo/hTable/hTableThermo.H
```

### 4.6.3 Modify hTableThermol.H

```
vi specie/thermo/hTable/hTableThermoI.H
```

#### Include cpInterpolation.H

The interpolation procedure for specific heat capacity at constant pressure, located in `cpInterpolation.H` is included.

Listing 4.13: specie/thermo/hTable/hTableThermoI.H

```

91 template<class EquationOfState>
92 inline Foam::scalar Foam::hTableThermo<EquationOfState>::Cp
93 (
94     const scalar p,
95     const scalar T
96 ) const
97 {
98     //return Cp_ + EquationOfState::Cp(p, T);
99     #include "cpInterpolation.H"
100 }

```

## Include haInterpolation.H

The interpolation procedure for absolute enthalpy, located in `haInterpolation.H` is included.

Listing 4.14: `specie/thermo/hTable/hTableThermoI.H`

```

103 template<class EquationOfState>
104 inline Foam::scalar Foam::hTableThermo<EquationOfState>::Ha
105 (
106     const scalar p, const scalar T
107 ) const
108 {
109     //return Cp_*T + Hf_ + EquationOfState::H(p, T);
110     #include "haInterpolation.H"
111 }
```

## Modify Calculation of Hs

The calculation for sensible enthalpy is modified.

Listing 4.15: `specie/thermo/hTable/hTableThermoI.H`

```

114 template<class EquationOfState>
115 inline Foam::scalar Foam::hTableThermo<EquationOfState>::Hs
116 (
117     const scalar p, const scalar T
118 ) const
119 {
120     //return Cp_*T + EquationOfState::H(p, T);
121     return Ha(p,T)-Hc();
122 }
```

## Modify Calculation of Hc

The calculation for chemical enthalpy is modified. It is assumed to be zero.

Listing 4.16: `specie/thermo/hTable/hTableThermoI.H`

```

125 template<class EquationOfState>
126 inline Foam::scalar Foam::hTableThermo<EquationOfState>::Hc() const
127 {
128     //return Hf_;
129     return 0;
130 }
```

## 4.6.4 Create Interpolation Files

The interpolation for specific heat capacity at constant pressure is implemented according to Section 4.2.

### cpInterpolation.H

```
vi specie/thermo/hTable/cpInterpolation.H
```

Listing 4.17: `specie/thermo/hTable/cpInterpolation.H`

```

1  int i_index;
2  scalar dT=1;
3  scalar T0=250;
4  scalar Temp_tabulated;
5  scalar Cp_T_tabulated;
6
7  // Specific heat at constant pressure (J/(kg K)), tabulated :
8  scalar Cp_tabulated[101]=
9  {
10     #include "cp.H"
11 }
```

```

12
13 // linear interpolation to calculate Cp(T) in J/(kg.K)
14 i_index = int(floor(fabs((T-T0)/dT)));
15 Temp_tabulated = T0+i_index*dT;
16 Cp_T_tabulated = Cp_tabulated[i_index]
17 + (Cp_tabulated[i_index+1]-Cp_tabulated[i_index])*(T-Temp_tabulated)/dT;
18
19 // return cp(T) in J/(kmol.K)
20 return Cp_T_tabulated*this->W();

```

## haInterpolation.H

The interpolation for absolute enthalpy according to Section 4.2 is implemented.

```
vi specie/thermo/hTable/haInterpolation.H
```

Listing 4.18: specie/thermo/hTable/haInterpolation.H

```

1 int i_index;
2 scalar dT=1;
3 scalar T0=250;
4 scalar Temp_tabulated;
5 scalar h_T_tabulated;
6
7 // enthalpy (J/kg), tabulated:
8
9 scalar h_tabulated[101] =
10 {
11 #include "ha.H"
12 };
13
14 // linear interpolation to calculate h(T) in J/kg
15 i_index = int(floor(fabs((T-T0)/dT)));
16 Temp_tabulated = T0+i_index*dT;
17 h_T_tabulated = h_tabulated[i_index]
18 + (h_tabulated[i_index+1]-h_tabulated[i_index])*(T-Temp_tabulated)/dT;
19
20 // return h(T) in J/kmol
21 return h_T_tabulated*this->W();

```

## 4.7 Modify Temperature Calculation

### 4.7.1 Modify thermoI.H

The Newton-Raphson solver for temperature calculation from enthalpy is replaced by a temperature interpolation from tables. In thermoI.H, all respective lines are commented out or deleted.

```
vi specie/thermo/thermo/thermoI.H
```

## Include TInterpolation.H

In the same way as implemented for all temperature-dependent quantities, the temperature interpolation is accomplished. In this case, however, the independent variable is the current enthalpy obtained from the solver, which is accessed by the scalar `f`.

Listing 4.19: specie/thermo/thermo/thermoI.H

```

30 /*template<class Thermo, template<class> class Type>
31 //inline Foam::species::thermo<Thermo, Type>::thermo
32 //(
33 // const Thermo& sp
34 //)
35 //:
36 // Thermo(sp)
37 //{
38 //
39 //
40 //template<class Thermo, template<class> class Type>
41 //inline Foam::scalar Foam::species::thermo<Thermo, Type>::T

```



```

42  //(
43  // scalar f,
44  // scalar p,
45  // scalar T0,
46  // scalar (thermo<Thermo, Type>::*F)(const scalar, const scalar) const,
47  // scalar (thermo<Thermo, Type>::*dFdT)(const scalar, const scalar)
48  // const,
49  // scalar (thermo<Thermo, Type>::*limit)(const scalar) const
50  //) const
51  //{
52  // if (T0 < 0)
53  // {
54  //   FatalErrorInFunction
55  //   // << "Negative initial temperature T0:" << T0
56  //   // << abort(FatalError);
57  // }
58  //
59  // scalar Test = T0;
60  // scalar Tnew = T0;
61  // scalar Ttol = T0*tol_;
62  // int iter = 0;
63  //
64  // do
65  // {
66  //   Test = Tnew;
67  //   Tnew =
68  //   // (this->*limit)
69  //   // (Test - ((this->*F)(p, Test) - f)/(this->*dFdT)(p, Test));
70  //   //
71  //   // if (iter++ > maxIter_)
72  //   // {
73  //   //   FatalErrorInFunction
74  //   //   // << "Maximum number of iterations exceeded:" << maxIter_
75  //   //   // << abort(FatalError);
76  //   // }
77  //   //
78  // } while (mag(Tnew - Test) > Ttol);
79  //
80  // return Tnew;
81  //}*/
82
83  template<class Thermo, template<class> class Type>
84  inline Foam::species::thermo<Thermo, Type>::thermo
85  (
86      const Thermo& sp
87  )
88  :
89      Thermo(sp)
90  {}
91
92
93  template<class Thermo, template<class> class Type>
94  inline Foam::scalar Foam::species::thermo<Thermo, Type>::T
95  (
96      scalar f,
97      scalar p,
98      scalar T0,
99      scalar (thermo<Thermo, Type>::*F)(const scalar, const scalar) const,
100     scalar (thermo<Thermo, Type>::*dFdT)(const scalar, const scalar) const,
101     scalar (thermo<Thermo, Type>::*limit)(const scalar) const
102 ) const
103 {
104     #include "TInterpolation.H"
105 }

```

## 4.7.2 Create Interpolation Files

### TInterpolation.H

The interpolation for temperature is implemented according to Section 4.2.

```
vi specie/thermo/thermo/TInterpolation.H
```

Listing 4.20: specie/thermo/thermo/TInterpolation.H

```

1  int i_index;
2  scalar dH=5916.4;
3  scalar h0=3.1558e+05;
4  scalar enth_tabulated;
5  scalar Tnew;
6  scalar h = f;
7
8  // Temperature (K), tabulated :
9

```

```

10 scalar T_tabulated[101]=
11 {
12     #include "Th.H"
13 };
14
15 // linear interpolation to calculate T(h) (K)
16 i_index = int(floor(fabs((h-h0)/dH)));
17 enth_tabulated = h0+i_index*dH;
18 Tnew = T_tabulated[i_index] + (T_tabulated[i_index+1]-T_tabulated[i_index])*(h-enth_tabulated)/dH;
19
20 return Tnew;

```

## 4.8 Declaration of the New Thermophysical Models

### 4.8.1 Declaration in thermoPhysicsTypes.H

#### Include

Include hTableThermo.H and rhoTable.H in thermoPhysicsTypes.H

```
vi specie/include/thermoPhysicsTypes.H
```

Listing 4.21: specie/include/thermoPhysicsTypes.H

```

51 #include "tableTransport.H"
52 #include "rhoTable.H"
53 #include "hTableThermo.H"

```

#### Add new models to the type definition

Listing 4.22: specie/include/thermoPhysicsTypes.H

```

59 typedef
60 tableTransport
61 <
62     species::thermo
63     <
64         hTableThermo
65         <
66             rhoTable<specie>
67         >,
68         sensibleEnthalpy
69     >
70 > tableGasHTermoPhysics;

```

### 4.8.2 Declaration in rhoThermos.H

#### Include

```
vi basic/rhoThermo/rhoThermos.C
```

Listing 4.23: basic/rhoThermo/rhoThermos.C

```

54 #include "tableTransport.H"
55 #include "hTableThermo.H"
56 #include "rhoTable.H"

```

### 4.8.3 Add new model combination to rhoThermos.C

Listing 4.24: basic/rhoThermo/rhoThermos.C

```

64 makeThermo
65 (
66     rhoThermo,

```

```
67     heRhoThermo,  
68     pureMixture,  
69     tableTransport,  
70     sensibleEnthalpy,  
71     hTableThermo,  
72     rhoTable,  
73     specie  
74 );
```

## 4.9 Copy Data Files

```
cp ~/Downloads/hummelTables/rho ./specie/equationOfState/rhoTable/rho.H  
cp ~/Downloads/hummelTables/ha ./specie/thermo/hTable/ha.H  
cp ~/Downloads/hummelTables/cp ./specie/thermo/hTable/cp.H  
cp ~/Downloads/hummelTables/mu ./specie/transport/table/mu.H  
cp ~/Downloads/hummelTables/kappa ./specie/transport/table/kappa.H  
cp ~/Downloads/hummelTables/Th ./specie/thermo/thermo/Th.H
```

## 4.10 Compile

```
cd specie  
wclean libso  
wmake libso  
cd ..  
cd basic  
wclean libso  
wmake libso
```

## 5 Tutorial Setup, Run and Post-Processing

A tutorial for the melting of n-octadecane in a rectangular cavity is set up. The tutorial is based on the buoyantSimpleFoam tutorial **buoyantCavity**. Solution and scheme settings are obtained from the buoyantPimpleFoam tutorial **hotRoom**. Figure 5.1 shows the computational domain with initial and boundary conditions for temperature and the dimensions. With an assumed melting temperature of 300 K, the phase change material is initially solid, while being heated from the left boundary (patch hot) and kept at initial temperature on the right boundary (patch cold). The top and bottom boundaries are assumed to be isolated ideally. In order to save computational resources, the assumption of two-dimensional flow has been made.

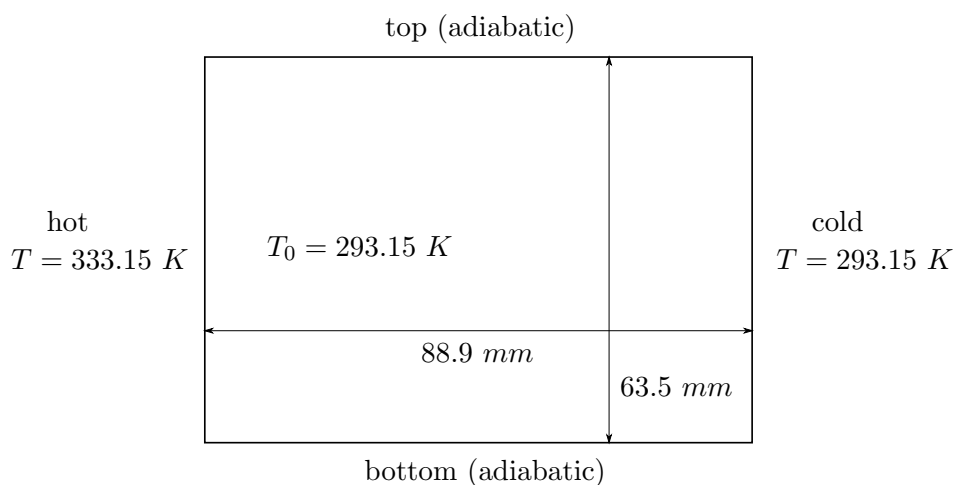


Figure 5.1: Domain with temperature boundaries and dimensions

### 5.1 Copy Tutorial Files

```
cd $WM_PROJECT_USER_DIR/run
cp -r $FOAM_TUTORIALS/heatTransfer/buoyantSimpleFoam/buoyantCavity/ ./phaseChangeCavity/
cd phaseChangeCavity
cp $FOAM_TUTORIALS/heatTransfer/buoyantPimpleFoam/hotRoom/system/controlDict ./system/
cp $FOAM_TUTORIALS/heatTransfer/buoyantPimpleFoam/hotRoom/system/fvSolution ./system/
cp $FOAM_TUTORIALS/heatTransfer/buoyantPimpleFoam/hotRoom/system/fvSchemes ./system/
```

### 5.2 Modify Domain Dimensions and Mesh Resolution

The domain dimensions and mesh resolution are modified for the test case.

```
sed -i s/"76"/"88.9"/g system/blockMeshDict
sed -i s/"260"/"31.75"/g system/blockMeshDict
sed -i s/"(35 150 15)"/"(40 1 30)"/g system/blockMeshDict
```

## 5.3 Modify Patches

In `blockMeshDict`, the patch type of `frontAndBack` is changed to `empty` and the mesh is created.

```
vi system/blockMeshDict
```

## 5.4 Create Mesh

The mesh is created by utilizing the `blockMesh` utility. See Figure 5.2.

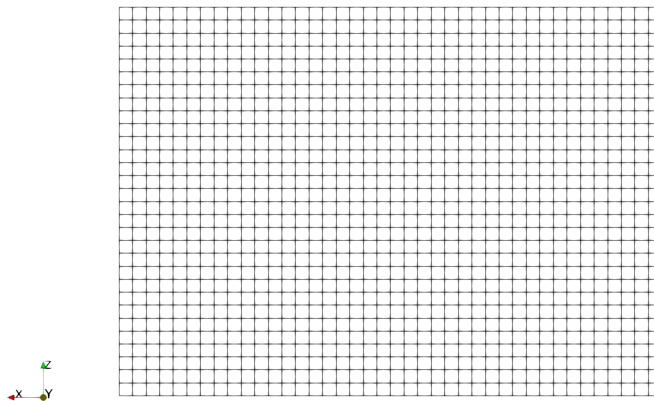


Figure 5.2: Mesh for the tutorial case

```
blockMesh
```

## 5.5 Modify Initial and Boundary Conditions

The boundary conditions of patch `frontAndBack` are changed to `empty` for `p`, `p_rgh`, `T` and `U`.

```
gedit 0/p 0/p_rgh 0/T 0/U
```

The temperature values are changed to fit the test case:

```
sed -i s/"293"/"293.15"/g 0/T
sed -i s/"288.15"/"293.15"/g 0/T
sed -i s/"307.75"/"333.15"/g 0/T
```

Boundary and initial conditions for the liquid volume fraction field `gl` are generated:

```
vi 0/gl
```

Listing 5.1: 0/gl

```
1 FoamFile
2 {
3   version 2.0;
4   format ascii;
5   class volScalarField;
6   object gl;
```

```

7 }
8 dimensions [0 0 0 0 0 0];
9 internalField uniform 0;
10 boundaryField
11 {
12 frontAndBack {type empty;}
13 topAndBottom {type zeroGradient;}
14 hot {type zeroGradient;}
15 cold {type zeroGradient;}
16 }

```

## 5.6 Modify Turbulence Properties

As laminar flow is assumed, the turbulence model is disabled.

```
sed -i s/"simulationType RAS;"/"simulationType laminar;"/g constant/turbulenceProperties
```

## 5.7 Modify Thermophysical Properties and Gravity

The entries in thermophysicalProperties are replaced by the table values at the initial temperature. The acceleration due to gravity is set in the negative z coordinate.

```

sed -i s/"28.96"/"254.5"/g constant/thermophysicalProperties
sed -i s/"1004.4"/"1700"/g constant/thermophysicalProperties
sed -i s/"1.831e-05"/"0.0041058"/g constant/thermophysicalProperties
sed -i s/"0.705"/"23.794"/g constant/thermophysicalProperties
sed -i s/"(0 -9.81 0)"/"(0 0 -9.81)"/g constant/g

```

The new thermophysical model is used by changing the keywords for transport, equationOfState and thermo models.

```

sed -i s/"const"/"table"/g constant/thermophysicalProperties
sed -i s/"hConst"/"hTable"/g constant/thermophysicalProperties
sed -i s/"perfectGas"/"rhoTable"/g constant/thermophysicalProperties

```

## 5.8 Modify Simulation Settings

The simulation is run for 1200 s. The first time step is set to 0.0001 and then controlled by a Courant Number of 1. Solution directories are written for every 60 seconds.

```

sed -i s/"2000;"/"1200;"/g system/controlDict
sed -i s/"2;"/"1e-4;"/g system/controlDict
sed -i s/"timeStep;"/"adjustableRunTime;"/g system/controlDict
sed -i s/"100;"/"60;"/g system/controlDict
sed -i s/"no;"/"yes;"/g system/controlDict
sed -i s/"0.5;"/"1;"/g system/controlDict

```

## 5.9 Add Function Object

Fields for density, thermal diffusivity, dynamic viscosity and compressibility are written every 60 seconds.

```
vi system/controlDict
```

Listing 5.2: system/controlDict

```

52 functions
53 {
54     writeThermophysicalProperties
55     {
56         type writeObjects;
57         libs ("libutilityFunctionObjects.so");
58         objects (thermo:alpha thermo:mu thermo:psi thermo:rho thermo:Cp);
59         writeControl adjustableRunTime;
60         writeInterval 60;
61     }
62 }

```

## 5.10 Add phaseChangeProperties

The values for latent heat of fusion, melting temperature, reference density, velocity treatment function constant and liquid volume update relaxation factor are set.

```
vi constant/phaseChangeProperties
```

Listing 5.3: constant/phaseChangeProperties

```

1 FoamFile
2 {
3     version 2.0;
4     format ascii;
5     class dictionary;
6     object transportProperties;
7 }
8 L L [0 2 -2 0 0 0 0] 212490;
9 Tmelt Tmelt [0 0 0 1 0 0 0] 300;
10 beta beta [0 0 0 -1 0 0 0] 9.1e-4;
11 rhoRef rhoRef [1 -3 0 0 0 0 0] 755.05;
12 C C [1 -3 -1 0 0 0 0] 1e5;
13 q q [0 0 0 0 0 0 0] 1e-3;
14 glRelax glRelax [0 0 0 0 0 0 0] 0.9;
15
16 minEnergyIter 2;
17 maxEnergyIter 50;
18 glTol 1e-2;

```

## 5.11 Run Tutorial

```
solidLiquidPhaseChangeFoam >log&
```

## 5.12 View Results

```
paraFoam -builtin
```

Figures 5.3 to 5.8 show the results after 1200 s. The thermophysical properties density, specific heat capacity, dynamic viscosity and thermal diffusivity (Figures 5.5 to 5.8) vary with temperature in the expected ranges in accordance with the input table values (cf. Figure 2.1). The melting front in Figure 5.4 appears to be plausible and shows the influence of natural convection.

However, validation is necessary to confirm that the solver in conjunction with the implemented thermophysical models gives physically sound results.

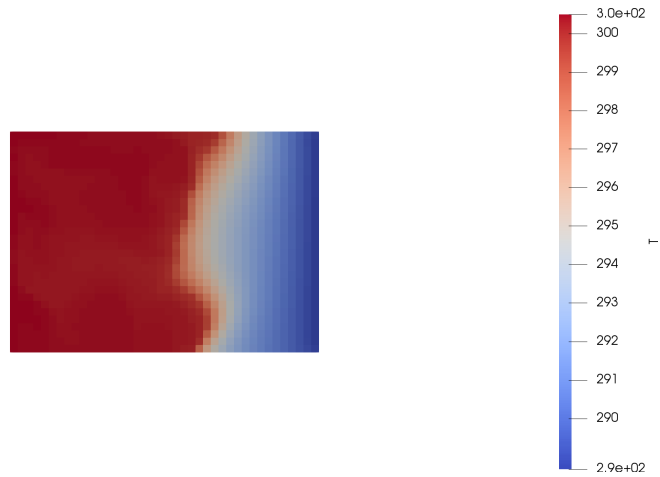


Figure 5.3: Temperature Field after 1200 s

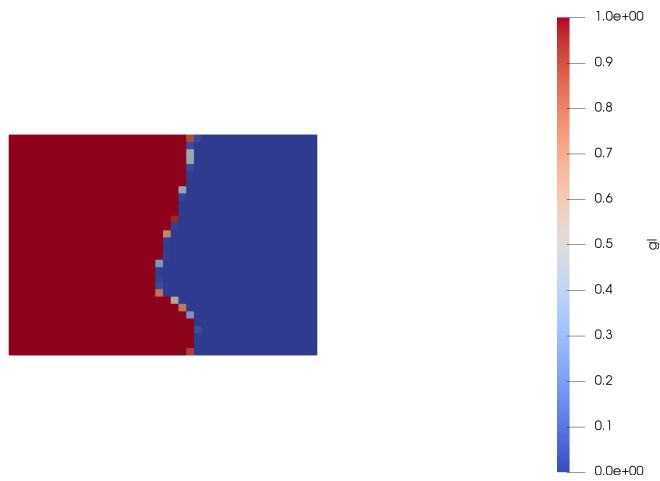


Figure 5.4: Liquid Volume Fraction Field after 1200 s

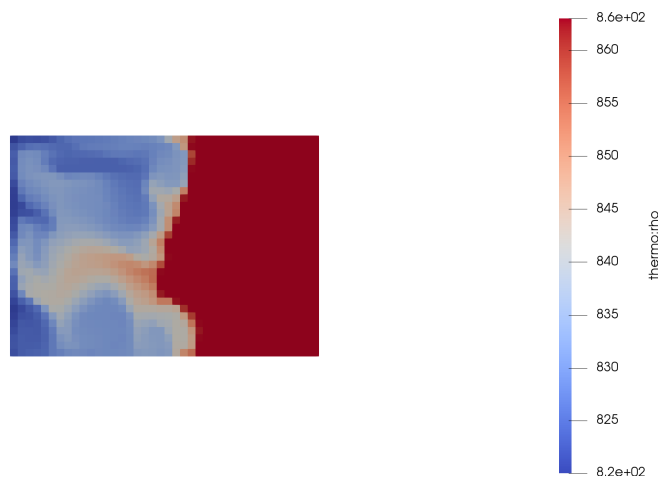


Figure 5.5: Density Field After 1200 s



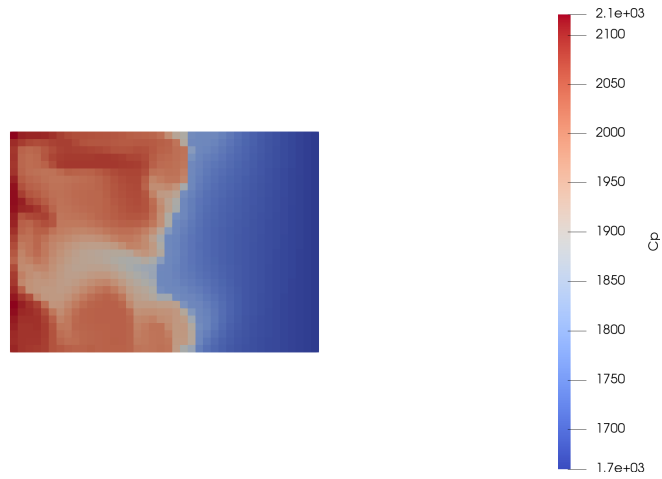


Figure 5.6: Specific Heat Capacity field after 1200 s

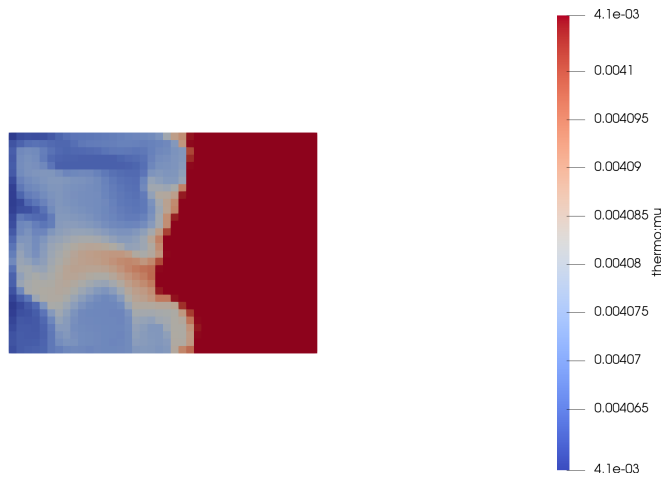


Figure 5.7: Dynamic Viscosity Field after 1200 s

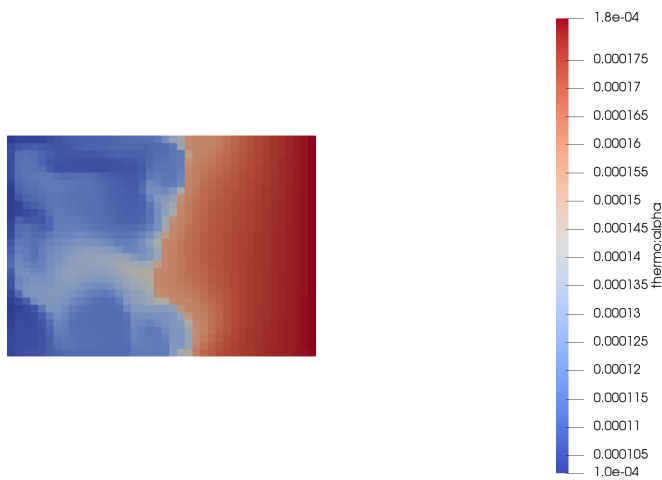


Figure 5.8: Thermal Diffusivity Field after 1200 s

# References

- [1] Brent, A. D. et al. “Enthalpy-Porosity Technique for Modeling Convection-Diffusion Phase Change”. In: *Numerical Heat Transfer* 13.3 (1988), 297–318.
- [2] Choquet, I. *ThermophysicalModels library in OpenFOAM-2.3.x – How to implement a new thermophysical model*. 2014. URL: [www.tfd.chalmers.se/~hani/kurser/OS\\_CFD\\_2014/isabelleChoquet/thermophysicalModels-OF-2.3-updated\\_HN141009.pdf](http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2014/isabelleChoquet/thermophysicalModels-OF-2.3-updated_HN141009.pdf).
- [3] Ducoulombier, D. et al. “Pressure (1-1000 bars) and Temperature (20-100 °C) Dependence of the Viscosity of Liquid Hydrocarbons”. In: *The Journal of Physical Chemistry* 90.8 (1986), 1692–1700.
- [4] ESI Group. *User Guide OpenFOAM v1706, Thermophysical models*. URL: <https://openfoam.com/documentation/user-guide/thermophysical.php#x19-770005.2>.
- [5] Marano, J. J. and Holder, G. D. “A General Equation for Correlating the Thermophysical Properties of n-Paraffins, n-Olefins, and other Homologous Series. 3. Asymptotic Behavior Correlations for Thermal and Transport Properties”. In: *Industrial & Engineering Chemistry Research* 36.6 (1997), 2399–2408.
- [6] Seyer, W. F. et al. “The Density and Transition Points of the n-Paraffin Hydrocarbons”. In: *Journal of the American Chemical Society* 66.2 (1944), 179–182.
- [7] Stefan, J. “Ueber die Theorie der Eisbildung, insbesondere über die Eisbildung im Polarmeere”. In: *Annalen der Physik* (1891).
- [8] Vélez, C. et al. “Temperature-Dependent Thermal Properties of Solid/Liquid Phase Change Even-Numbered n-Alkanes: n-Hexadecane, n-Octadecane and n-Eicosane”. In: *Applied Energy* 143 (2015), 383–394.
- [9] Voller, V. “An Overview of Numerical Methods for Solving Phase Change Problems”. In: *Advances in Numerical Heat Transfer*. 1996.
- [10] Vuik, C. “Some Historical Notes About The Stefan Problem”. In: *Nieuw Archief voor Wiskunde*. 4th ser. 11 (1993), pp. 157–167.

## 6 Study questions

1. What is `g1`?
2. Why is the Boussinesq approximation, as e.g. presented by Brent et al. [1], not implemented in this tutorial?
3. What is the reason for adding the line `-L$(FOAM_USER_LIBBIN)` to `basic/Make/options`?
4. What is `alphah`?
5. Where is temperature calculated from enthalpy in the thermophysical models in OpenFOAM v1706?