Cite as: Khosravifar, P.: Implementation of Solidification Phase Change in a Multiphase Solver. In Proceedings of CFD with OpenSource Software, 2024, Edited by Nilsson. H.,
http://dx.doi.org/10.17196/OS_CFD#YEAR_2024

CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Implementation of Solidification Phase Change in a Multiphase Solver

Developed for OpenFOAM-v2312

Author:
Paria Khosravifar
Luleå University of Technology
paria.khosravifar@ltu.se

Peer reviewed by:
Assoc. Prof. Anna-Lena Ljung
Dr. Saeed Salehi

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

January 28, 2025

Learning outcomes

The main requirements of a tutorial in the course is that it should teach the four points: How to use it, The theory of it, How it is implemented, and How to modify it. Therefore the list of learning outcomes is organized with those headers.

The reader will learn:

How to use it:

- How to use the `interFoam` solver.

The theory of it:

- The theory of enthalpy-porosity solidification model.
- The theory of multiphase simulation.

How it is implemented:

- How the interface of two-phase is tracked using the VOF method.

How to modify it:

- How to implement phase-change into `interFoam` solver.
- How to couple the enthalpy-porosity method with the VOF method.

Prerequisites

- Fundamentals of multiphase flow
- Fundamentals of solid-liquid phase-change
- Basic usage of OpenFOAM
- C++ language and Object-Oriented Programming (OOP)

Contents

Nomenclature

Acronyms

erf	Error Function
Ste	Stefan number
VOF	Volume of Fluid

English symbols

A_{mush}	Mushy zone constant	$\mathrm{kg/m^3 \cdot s}$
C_p	Specific Heat	$\mathrm{J/kg \cdot K}$
H	Enthalpy	J/kg
h	Sensible Enthalpy	J/kg
k	Thermal conductivity	$\mathrm{W/(m \cdot K)}$
L	Latent heat of fusion	J/kg
p	Pressure	Pa
T	Temperature	K
t	Time	s
T_0	Surface temperature	K
u	Velocity	m/s
u_r	Relative velocity	m/s

Greek symbols

α	Volume Fraction of liquid in two-phase fluid	
ΔT_F	phase change temperature range	K
γ	Volume Fraction of liquid in solidification	
κ	curvature of the interface	
ν	Fluid kinematic viscosity	$\mathrm{m^2/s}$
ρ	Fluid density	$\mathrm{kg/m^3}$
σ	surface tension coefficient	N/m

Subscripts

p	pressure
g	Gas
l	Liquid
liq	Liquidus
melt	Melting
ref	reference
s	Solid
sol	Solidus
st	surface tension

Chapter 1

Introduction

Modeling phase-change phenomena, such as solidification, is important for understanding various natural and industrial systems. The main challenge in the simulation of multiphase systems is accurately tracking the interphase between phases, especially when phase-change processes are involved. Several methods have been introduced to track the freezing front in solidification phase change simulations [1], with the enthalpy-porosity method being one of the most widely used approaches. Additionally, in cases involving free surfaces or interactions with other fluids, it becomes essential to consider multiple phases within the simulation framework. For instance, in the freezing of an impinging droplet on a surface, the liquid-gas interface must be tracked accurately, along with the liquid-solid interface during solidification.

This report presents the modification and implementation of existing `interFoam` solver in `OpenFOAM` to include solidification phase-change. `interFoam` is a solver for two incompressible fluids that utilities the volume of fluid (VOF) method for tracking interface between two phases. However, it does not originally support the phase-change phenomena. To address this limitation, the energy equation and the enthalpy-porosity method have been introduced into the `interFoam` solver. The enthalpy-porosity method is applied by adding source terms to the governing equations. As a result, the modified solver becomes capable of simulating three phases—fluid 1, fluid 2, and solid—simultaneously, making it suitable for cases involving complex multiphase systems.

The extended solver has been validated using the Stefan problem [2]. First, a two-phase Stefan problem was simulated to check the solver's capability in simulating phase transitions from liquid to solid. Next, a three-phase Stefan problem was defined to simulate the solidification of a liquid with a free surface in contact with air.

Chapter 2

Theory

In this chapter, the theory of multiphase flow simulation with solidification phase change is discussed. The focus is on simulating the solidification of a liquid phase with a free surface by coupling the volume of fluid (VOF) method with the enthalpy-porosity model. In other words, the interaction between two liquid and gas phases will be achieved using the VOF method, while the enthalpy-porosity method will be utilized to simulate the solidification process of the liquid phase.

2.1 Multiphase Simulation

The volume of fluid (VOF) model has been discussed in several previous reports in this PhD course [3, 4, 5]. Therefore, only the aspects relevant to implementation of the solver in this project are mentioned in this section. The VOF model is a numerical technique for capturing the interface between two phases. It employs a scalar function, known as the volume fraction (α), which is the ratio of the volume of one phase to the volume of the computational cell. The volume fraction function is commonly defined as follows

$$\alpha = \begin{cases} 0 & \text{fluid 2,} \\ 0 < \alpha < 1 & \text{interface,} \\ 1 & \text{fluid 1 .} \end{cases} \tag{2.1}$$

In the volume of fluid method, the transport equation of the volume fraction of the liquid is solved to track the interface between two fluid phases. This equation describes the evolution of the volume fraction (α) of one fluid within each computational cell over time. In OpenFOAM, this equation is defined as

$$\frac{\partial \alpha}{\partial t} + \nabla.(\alpha U) + \nabla.\left[u_r \alpha(1-\alpha)\right] = 0, \tag{2.2}$$

In this equation, the $\nabla.\left[u_r \alpha(1-\alpha)\right]$ term, is an artificial term introduced to sharpen and resist numerical diffusion. This term is valid only at the interface, where u_r is the relative velocity between two phases. The fluid properties of the two-phase mixture in the interface are obtained from the following relation

$$y = \alpha y_1 + (1-\alpha) y_2, \tag{2.3}$$

where 1 and 2 represent the fluids 1 and 2, respectively. y can be any physical property of the fluid. As mentioned previously, the VOF method is used solely to track the interface between two fluids. The formation of the third phase (solid) will be treated by the enthalpy-porosity method.

2.2 Solidification Model

The enthalpy-porosity model is a widely used numerical approach in computational fluid dynamics for simulating solidification/melting phase-change processes. This method, presented by Voller and

Prakash [6], utilizes the enthalpy method to account for both sensible and latent heat, along with the porosity concept to represent the transition between the solid and liquid phases. In the porosity concept, the interface between the solid and liquid phases is treated as a porous medium, where the liquid volume fraction represents the porosity of that medium.

In a phase-change problem with fluid flow, the conservation of mass, momentum, and energy need to be considered. The details of the governing equations will be discussed in the next subsection.

2.2.1 Governing Equations

In the simulation of a fluid flow problem using the volume of Fluid (VOF) method, one set of continuity, momentum, and energy equations needs to be solved for the phase mixture. The transient energy equation in terms of enthalpy is defined as

$$\frac{\partial \rho H}{\partial t} + \nabla \cdot (\rho U H) = \nabla \cdot (k \nabla T)\,, \tag{2.4}$$

where

$$H = h + \Delta H. \tag{2.5}$$

Here, h is the sensible enthalpy and ΔH is the latent heat. The sensible enthalpy is defined as

$$h = h_{\text{ref}} + \int_{T_{\text{ref}}}^{T} C_p \mathrm{d}T, \tag{2.6}$$

where h_{ref} is the reference enthalpy at the reference temperature T_{ref}, and C_{p} is the specific heat at constant pressure. In solidification process, considering only the liquid and solid phases, the latent heat is defined as

$$\Delta H = \gamma L, \tag{2.7}$$

where L is the latent heat of fusion and γ is the volume fraction of liquid in a cell; it equals 1 in the liquid phase, 0 in the solid phase, and lies between 0 and 1 in the mushy region, defined by a linear function. In a non-isothermal phase-change, this is described as following step function

$$\gamma = \begin{cases} 0 & T < T_{\text{sol}}, \\ \frac{T - T_{\text{sol}}}{T_{\text{liq}} - T_{\text{sol}}} & T_{\text{sol}} \leq T \leq T_{\text{liq}}, \\ 1 & T > T_{\text{liq}}, \end{cases} \tag{2.8}$$

where T_{liq} is the upper bound (liquidus temperature) and T_{sol} is the lower bound (solidus temperature) of the phase change temperature range (ΔT_{F}), which are defined as follows

$$T_{\text{sol}} = T - \Delta T_{\text{F}}/2 \quad , \quad T_{\text{liq}} = T + \Delta T_{\text{F}}/2. \tag{2.9}$$

However, when considering three phases (two fluids and a solid), the latent heat cannot be used as described in Eq.(2.7), because the existence of two different fluids must also be taken into account. To clarify, the two fluid phases are considered to be gas and liquid (i.e., the solidification of liquid with an interface with air). Consequently, the liquid volume fraction from the VOF model in a cell containing both liquid and gas is defined as α_{l}. Therefore, Eq.(2.7) is extended as the following statement [7]

$$\Delta H = \alpha_{\text{l}} \gamma L. \tag{2.10}$$

In this case, the Eq. (2.10) is valid only in the liquid region of the simulation.

By substituting the definition of H from the above-mentioned relations into the Eq. (2.4), the energy equation that includes the sensible and latent heat parts, which account for the solidification process, is obtained as

$$\frac{\partial (\rho C_p T)}{\partial t} + \nabla \cdot (U \rho C_p T) + \alpha_{\text{l}} L \left[\frac{\partial \rho \gamma}{\partial t} + \nabla \cdot (U \rho \gamma) \right] = \nabla \cdot (k \nabla T). \tag{2.11}$$

Using the piecewise linear function described in the original enthalpy-porosity method [6] (Eq. (2.8)) needs a special iterative solution to couple between the liquid volume fraction and the energy equation. Rosler and Bruggemann [8] presented a continuous liquid fraction relation based on error function (erf) for faster and more stable convergence. Therefore, the liquid fraction is defined as

$$\gamma = 0.5\mathrm{erf}\left(\frac{4(T - T_{\mathrm{melt}})}{T_{\mathrm{liq}} - T_{\mathrm{sol}}}\right) + 0.5, \tag{2.12}$$

where T_{melt} is the melting temperature in isothermal phase change and the mean between T_{liq} and T_{sol} in non-isothermal phase change. Applying Eq. (2.12) to Eq. (2.11) the final energy conservation equation and its source term are given by

$$\frac{\partial(\rho C_p T)}{\partial t} + \nabla \cdot (U \rho C_p T) + S_h = \nabla \cdot (k \nabla T), \tag{2.13}$$

where

$$S_h = \alpha_{\mathrm{l}} \rho L \frac{4\exp((\frac{4(T-T_{melt})}{T_{liq}-T_{sol}})^2)}{(T_{liq} - T_{sol})\sqrt{\pi}} (\frac{\partial T}{\partial t} + U \cdot \nabla T). \tag{2.14}$$

This equation is no longer a function of the liquid fraction and does not require any iterative treatment to update γ. The conservation of momentum is defined as

$$\frac{\partial(\rho U)}{\partial t} + \nabla \cdot (\rho U U) = -\nabla p + \nabla \mu \left[\nabla U + \nabla U^{\mathrm{T}}\right] + \rho g + S_{\mathrm{st}} + S_{\mathrm{u}}. \tag{2.15}$$

In this equation, S_{st} is the volumetric surface tension force on the interface of two fluids, and S_{u} represents the Darcy source term [6]. As the liquid solidifies, its velocity becomes zero. The Darcy source term adds an artificial momentum source over the interface to represent this velocity sink. The Carman-Kozeny equation is used in the definition of Darcy source term. This term is defined as

$$S_{\mathrm{u}} = -\frac{(1-\gamma)^2}{\gamma^3 + \mathrm{q}} A_{\mathrm{mushy}} U, \tag{2.16}$$

where A_{mushy} is the mushy zone constant and q is a small number to avoid division by zero. The constantA_{mushy} controls how much the flow slows down during the phase change. The continuous surface force (CSF) is used to calculate the surface tension force which is defined as follows

$$S_{\mathrm{st}} = \sigma \kappa \nabla \alpha_{\mathrm{l}}, \tag{2.17}$$

where σ is the surface tension coefficient and κ is the curvature of the interface of two fluids and it is computed as follows

$$\kappa = -\nabla \cdot (\frac{\nabla \alpha_{\mathrm{l}}}{|\nabla \alpha_{\mathrm{l}}|}). \tag{2.18}$$

Finally, as a single set of equations is solved for the mixture of three phases, the material properties of the mixture, incorporating the solid phase properties into Eq. (2.3), are defined as follows

$$y = \alpha_{\mathrm{l}}(\gamma y_{\mathrm{l}} + (1-\gamma) y_{\mathrm{s}}) + (1 - \alpha_{\mathrm{l}}) y_{\mathrm{g}}, \tag{2.19}$$

where l, s, and g represent liquid, solid, and gas, respectively and y can represent any physical property. These properties are assumed to be constant within each phase.

Chapter 3

solidificationInterFoam

In this chapter, the existing implementations related to this project are discussed, along with the modifications made to the `interFoam` solver to simulate solidification phase-change. To enable the simulation of solidification phase-change, the energy equation and solidification source terms have been added to the `interFoam` solver, and the final solver, named `solidificationInterFoam`, has been developed.

3.1 Existing Implementations

3.1.1 interFoam Solver

`interFoam` is a solver for two incompressible, isothermal immiscible fluids using a VOF(volume of fluid) phase-fraction based interface capturing approach. The structure of `interFoam` directory is as follows.

```
|-- Make
|   |-- files
|   `-- options
|-- UEqn.H
|-- alphaSuSp.H
|-- correctPhi.H
|-- createFields.H
|-- initCorrectPhi.H
|-- interFoam.C
|-- pEqn.H
`-- rhofs.H
```

`interFoam.C` is the main file that controls the overall workflow of the solver. It includes the necessary header files, such as `UEqn.H`, `pEqn.H`, and `alphaEqn.H`, which are used to solve the momentum, pressure, and volume fraction equations, respectively. This solver uses the transport equation for the volume fraction of liquid, defined in the `multiphase/VoF` directory as `alphaEqn.H`. This file is explained in detail in the report by Almeland [4]. Additionally, all fields used in the solver, such as velocity, pressure, density, etc., are initialized and declared in the `CreateFields.H` file.

The `interFoam` solver uses the `immiscibleIncompressibleTwoPhaseMixture` class to read and calculate transport properties from the `transportProperties` file in the system directory. This class combines properties for handling two immiscible, incompressible fluids. As shown in Listing 3.1, it inherits and combines functionality from two classes: `incompressibleTwoPhaseMixture`, which manages the transport properties of the two incompressible phases (such as density and viscosity), and `interfaceProperties`, which handles properties related to the interface between the two phases, such as surface tension and curvature.

Listing 3.1: immiscibleIncompressibleTwoPhaseMixture inheritance

```
53 class immiscibleIncompressibleTwoPhaseMixture
54 :
55     public incompressibleTwoPhaseMixture,
56     public interfaceProperties
```

The mixture object from the `immiscibleIncompressibleTwoPhaseMixture` class has defined in `createFields.H` file to have access to functions and properties in the class as follows.

```
Info<< "Reading transportProperties\n" << endl;
immiscibleIncompressibleTwoPhaseMixture mixture(U, phi);
```

3.2 Extending Thermodynamic Properties

The specific heat coefficient (C_p) and thermal conductivity (k) are essential thermodynamic properties used in the energy equation. In this section the modification is made in the solver to consider thermodynamic properties are mentioned.

In order to read these data from `transportProperties` dictionary, the `thermoIncompressibleTwoPhaseMixture` class from `$FOAM_SOLVERS/multiphase/interCondensingEvaporatingFoam/temperaturePhaseChangeTwoPhaseMixtures` is used by changing its name to `myThermoIncompressibleTwoPhaseMixture`. This class inherits from `incompressibleTwoPhaseMixture` class and extends its functionality to read thermodynamic properties, such as C_p and k. In this case, the `myImmiscibleIncompressibleTwoPhaseMixture` class should be the subclass of `myThermoIncompressibleTwoPhaseMixture`. Listing 3.2 shows the inheritance of `myImmiscibleIncompressibleTwoPhaseMixture`. Full files of these two libraries can be found in Appendix A.1.

Listing 3.2: myImmiscibleIncompressibleTwoPhaseMixture inheritance

```
55 class myImmiscibleIncompressibleTwoPhaseMixture
56 :
57     // public incompressibleTwoPhaseMixture,
58     public myThermoIncompressibleTwoPhaseMixture,
59     public interfaceProperties
```

Finally, to get access to mixture properties `mixture` object of `myImmiscibleIncompressibleTwoPhaseMixture` class is defined as following in the `createFields.H` file. Listing 3.3 illustrates the definition of the `mixture` object.

Listing 3.3: mixture object definition in createFields.H file

```
116 Info<< "Reading transportProperties\n" << endl;
117 myImmiscibleIncompressibleTwoPhaseMixture mixture(U, phi);
```

`myImmiscibleIncompressibleTwoPhaseMixture` and `myThermoIncompressibleTwoPhaseMixture` are defined as separate libraries. To use them in solidificationInterFoam solver they should be mentioned as follows in `Make/options` file.

```
    -L$(FOAM_USER_LIBBIN) \
    -lmyImmiscibleIncompressibleTwoPhaseMixture \
    -lmyThermoIncompressibleTwoPhaseMixture
```

3.3 Modifying Governing Equations

In this section, the modifications in the governing equations to consider phase change are discussed. The energy equation introduced in the theory chapter (Eq. (2.13)) has been implemented in the solver. Listing 3.4 shows the modified energy equation. The source term in the energy equation is implemented in the lines 6 and 7 of this listing and added to the `TEqn` in the line 13. The `alpha1`

variable in line 13 represents the volume fraction of liquid (α_l), which is calculated using the transport equation for the liquid's volume fraction, as shown in Eq. (2.2) (VOF method). Its value is 1 in the liquid phase and 0 in the gas phase. Consequently, this ensures that the term in line 13 is valid only in the liquid phase and becomes zero in the gas phase.

Listing 3.4: TEqn

```
6     volScalarField expArg = sqr(4.0 * (T - Tmelt) / (Tliq - Tsol));
7     volScalarField Sh_erf = -rho * L * ((4.0 * exp(-expArg) / ((Tliq - Tsol) * ::sqrt(Foam::constant::
       mathematical::pi))) * (fvc::ddt(T) + (U & fvc::grad(T))));
8
9     fvScalarMatrix TEqn
10    (
11         fvm::ddt(rhoCp, T)
12       + fvm::div(rhoCpPhi, T)
13       + alpha1*Sh_erf
14       - fvm::Sp(fvc::ddt(rhoCp) + fvc::div(rhoCpPhi), T)
15       - fvm::laplacian(kappaEff, T)
16    );
```

To track the liquid volume fraction during the solidification process, the `volScalarField gamma` is defined in `createFields.H` file. The code Listing 3.5 defines this field. In line 129 the amount of `gamma` is initialized by defining the Eq. 2.12. This variable will be updated after solving T in the TEqn file, and written in the time directories during run (lines 30 to 33 in Listing 3.6), to be able to display in post-processing.

Listing 3.5: Solidification volume fraction field (γ) defination

```
114 volScalarField gamma
115 (
116     IOobject
117     (
118         "gamma",
119         mesh.time().timeName(),
120         mesh,
121         IOobject::NO_READ,
122         IOobject::AUTO_WRITE
123     ),
124     mesh,
125     dimensionedScalar("gamma", dimless, 1)
126 );
127
128 // Update gamma using the erf function
129 gamma = 0.5 * Foam::erf(4.0 * (T - Tmelt) / (Tliq - Tsol)) + 0.5;
```

Listing 3.6: The update of γ in TEqn.H

```
24    gamma = 0.5 * Foam::erf(4.0 * (T - Tmelt) / (Tliq - Tsol)) + 0.5;
25    gamma.correctBoundaryConditions();
26    Info << "min/max(gamma) = " << min(gamma).value() << ", "
27     << max(gamma).value() << endl;
28
29    // Write the field to the time directory
30    if (runTime.writeTime())
31    {
32        gamma.write();
33    }
```

Finally, the momentum equation is modified to consider the artificial source term as mentioned in the theory chapter. The Listing 3.7 displays momentum equation with added source terms. In line 3 of this code the Carman-Kozeny relation is calculated and added to the UEqn in line 8 to include Darcy source term of Eq. (2.16). Same as the comment in TEqn, alpha1 in line 8 makes this term to be valid only in the liquid phase.

Listing 3.7: UEqn

```
    volScalarField prosityFunc = A_mushy *sqr(1.0 - gamma)/(pow3(gamma) + q);

    fvVectorMatrix UEqn
    (
        fvm::ddt(rho, U) + fvm::div(rhoPhi, U)
      + alpha1 * fvm::Sp(prosityFunc, U) //Add
      + MRF.DDt(rho, U)
      + turbulence->divDevRhoReff(rho, U)
     ==
        fvOptions(rho, U)
    );

    UEqn.relax();

    fvOptions.constrain(UEqn);

    if (pimple.momentumPredictor())
    {
        solve
        (
            UEqn
         ==
            fvc::reconstruct
            (
                (
                    mixture.surfaceTensionForce()
                  - ghf*fvc::snGrad(alpha1*rho1+alpha2*rho2)
                  - fvc::snGrad(p_rgh)
                ) * mesh.magSf()
            )
        );

        fvOptions.correct(U);
    }
```

The properties are required in the equations of solidification phase change such as latent heat of fusion (L), mushy zone constant (A_{mushy}), melting temperature (T_{melt}), etc will define in the `phaseChangeProperties` directory in the system folder of the test case. This file is mentioned in the `createFields.H` file as the Listing 3.8.

Listing 3.8: phaseChangeProperties defination

```
Info<< "Reading phaseChangeProperties" << endl; //Add
IOdictionary phaseChangeProperties
(
    IOobject
    (
        "phaseChangeProperties",
        runTime.constant(),
        mesh,
        IOobject::MUST_READ,
        IOobject::NO_WRITE
    )
);
```

The full script of the modified solver is represented in Appendix A.

Chapter 4

Test Case

In this chapter, the accuracy of the modified `solidificationInterFoam` solver is verified in two sections. Firstly, the movement of the freezing front is validated by comparing the results with the analytical solution of the two-phase Stefan problem [2]. Secondly, a three-phase Stefan problem is simulated to check the solver's ability to handle solidification phase-change in the presence of two fluids.

4.1 Two-phase Stefan problem

The Stefan problem is a classical model in heat transfer problems with phase change. It is also known as a moving boundary problem. In this problem, the development of the solid-liquid interface is determined by solving the conservation of energy. One of the analytical solutions of the Stefan problem is known as Neumann's solution. This solution is applied to the one-dimensional solidification of a semi-infinite domain, in which the development of the interface position over time is presented in the following expression [9]

$$x(t) = 2\lambda\sqrt{\frac{tk_s}{C_s\rho}}, \tag{4.1}$$

where t represents time, while k_s, C_s, and ρ_s denote the thermal conductivity, heat capacity, and density of the solid phase, respectively. The constant λ is obtained via solving the following transcendental equation

$$\lambda \mathrm{e}^{\lambda^2} = \frac{\mathrm{Ste}}{\sqrt{\pi}}, \tag{4.2}$$

where Ste is Stefan's number, defined as follows

$$\mathrm{Ste} = \frac{C_s(T_{\mathrm{melt}} - T_0)}{\mathrm{L}}, \tag{4.3}$$

where T_0 refers to surface temperature.

To simulate isothermal solidification in Stefan problem, the computational domain is illustrated in the Figure 4.1, which is a rectangular domain with dimensions of 100cm × 25cm. The initial temperature inside the domain is 4°C and the melting temperature is 0°C. For the isothermal solidification, phase change occurs at a constant temperature (T_{melt}). To use the continuous liquid fraction Eq. (2.12), the phase change temperature range set to small value of 0.4 K ($T_{\mathrm{liq}} = 273.35$ K, $T_{\mathrm{sol}} = 272.95$ K). The properties of materials are represented in Table 4.1. The value of λ obtained from Eq. (4.2) is equal to 0.56742. In this step of simulation, the initial phase of the entire domain is defined as liquid. For this purpose, the `setFieldsDict` dictionary is defined as Listing 4.2, to apply liquid phase to the whole domain in the initial step.

Listing 4.1: setFieldsDict in two-phase case

```
17 defaultFieldValues
18 (
19     volScalarFieldValue alpha.water 0
20 );
21
22 regions
23 (
24     boxToCell
25     {
26         box (0 -0.001 0) (0.025 0.001 0.100);
27         fieldValues
28         (
29             volScalarFieldValue alpha.water 1
30         );
31     }
32 );
```

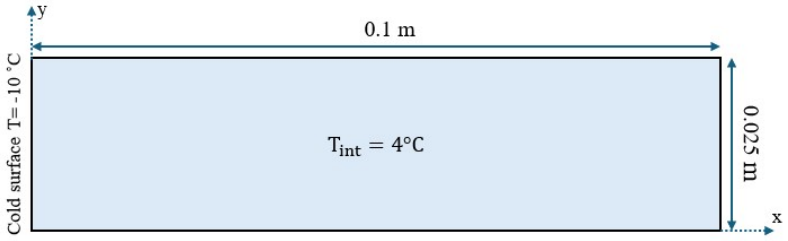

Figure 4.1: Schematic of computational domain.

Movement of the solid-liquid interface during phase change, compared to the results of the analytical results of Eq. (4.1) and is displayed Figure 4.2. To track the interface in simulation, the `isoSurface functionObject` is used to obtain the location of $T = 273.15$ K iso-surface. Figure 4.2 shows that the simulation results agree with the analytical solution and confirms the solver's ability in the simulation of solidification phase change.

Table 4.1: Material's properties

Properties	Solid	Liquid	Gas	Interface
Heat capacity (J/kg·K)	2050	2590	1	
Thermal conductivity (W/m·K)	4.02	2.89	0.025	
Density (kg/m^3)	1000	1000	1	
Melting temperature (K)				273.15
Latent heat of fusion (J/kg)				80332

4.2 Three-phase Stefan problem

In order to check the ability of the solver to simulate solidification in the presence of two fluids, a three-phase Stefan problem has been defined. In this case, half of the computational domain is defined as liquid and the other half as air. The only difference in `setFieldsDict` is in line 26, dimensions of the box. Figure 4.3 shows the schematic of this test case. Natural convection is neglected to have a flat liquid-gas interface.

Listing 4.2: setFieldsDict in three-phase case

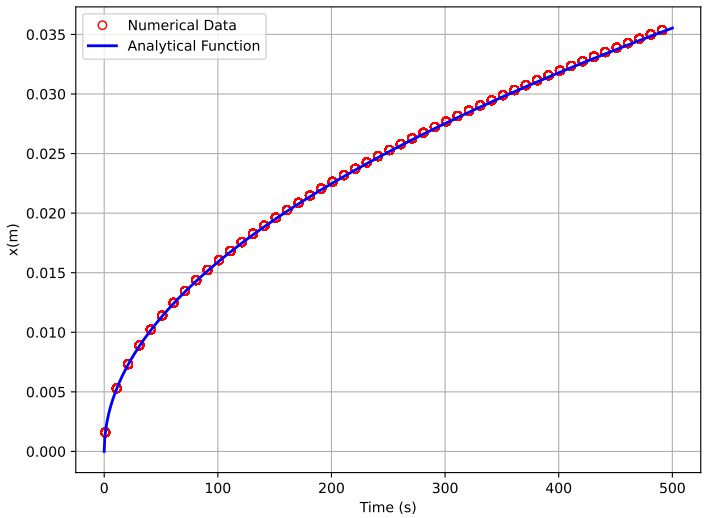

Figure 4.2: Comparison of numerical data and analytical solution of freezing front location.

```
26        box (0 -0.001 0) (0.025 0.001 0.050);
```

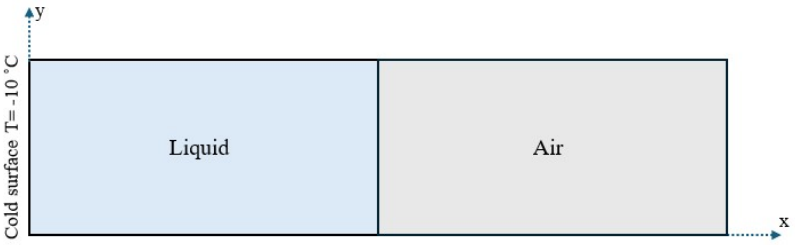

Figure 4.3: Schematic of computational domain

Figure 4.4 shows the evolution of the freezing front in the three-phase case. The freezing front moves upward until it reaches the interface between air and liquid at x=0.05 m.

4.3 Conclusion

The `solidificationInterFoam` solver successfully simulated the solidification phase change and effectively handled three-phase interactions during the solidification of a liquid in the presence of another fluid.

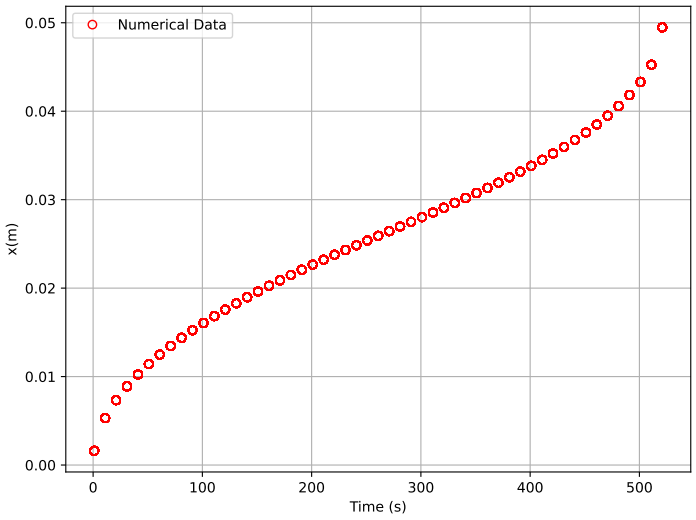

Figure 4.4: Freezing front evolution in three-phase Stefan problem.

Bibliography

[1] S. Akhtar, M. Xu, M. Mohit, and A. P. Sasmito, "A comprehensive review of modeling water solidification for droplet freezing applications," *Renewable and Sustainable Energy Reviews*, vol. 188, p. 113768, 2023. [Online]. Available: https://doi.org/10.1016/j.rser.2023.113768

[2] D. W. Hahn and M. N. Özisik, *Heat conduction*. John Wiley & Sons, 2012.

[3] S. Menon, "Coupled level-set with vof interfoam," *Proceedings of CFD with OpenSource Software, 2015, Edited by Nilsson H.*, 2015. [Online]. Available: https://dx.doi.org/10.17196/OS_CFD#YEAR_2015

[4] S. Almeland, "Implementation of an air-entrainment model in interfoam," *In Proceedings of CFD with OpenSource Software, 2018, Edited by Nilsson. H.*, 2018. [Online]. Available: http://dx.doi.org/10.17196/OS_CFD#YEAR_2018

[5] Y. Sun, "Description of intercondensatingevaporatingfoam and implementation of sgs term into volume fraction equation," *In Proceedings of CFD with OpenSource Software, 2022, Edited by Nilsson. H.*, 2022. [Online]. Available: http://dx.doi.org/10.17196/OSCFD#YEAR2022

[6] V. Voller and C. Prakash, "A fixed grid numerical modelling methodology for convection-diffusion mushy region phase-change problems," *International Journal of Heat and Mass Transfer*, vol. 30, no. 8, pp. 1709–1719, 1987. [Online]. Available: https://doi.org/10.1016/0017-9310(87)90317-6

[7] Y. Yao, C. Li, Z. Tao, and R. Yang, "Numerical simulation of water droplet freezing process on cold surface," in *ASME International Mechanical Engineering Congress and Exposition*, vol. 58431. American Society of Mechanical Engineers, 2017, p. V008T10A072. [Online]. Available: https://doi.org/10.1115/IMECE2017-71175

[8] F. Rösler and D. Brüggemann, "Shell-and-tube type latent heat thermal energy storage: numerical analysis and comparison with experiments," *Heat and mass transfer*, vol. 47, no. 8, pp. 1027–1033, 2011. [Online]. Available: https://doi.org/10.1007/s00231-011-0866-9

[9] M. J. de Lemos and A. J. Hodierne, "A fully implicit enthalpy-porosity model for phase-change," *ASME Journal of Heat and Mass Transfer*, vol. 146, no. 1, 2024. [Online]. Available: https://doi.org/10.1115/1.4063732

Study questions

1. How can the `interFoam` solver be modified to simulate solidification phase change?
2. How does the enthalpy-porosity method model solidification phase change?
3. How can a three-phase system be incorporated into the simulation of liquid solidification with a free surface?

Appendix A

Developed codes

A.1 Libraries

myThermoIncompressibleTwoPhaseMixture.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2016-2020 OpenCFD Ltd.
-------------------------------------------------------------------------------

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

#include "myThermoIncompressibleTwoPhaseMixture.H"


// * * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * * * //

namespace Foam
{
    defineTypeNameAndDebug(myThermoIncompressibleTwoPhaseMixture, 0);
}


// * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * //

Foam::myThermoIncompressibleTwoPhaseMixture::myThermoIncompressibleTwoPhaseMixture
(
    const volVectorField& U,
    const surfaceScalarField& phi
)
:
    incompressibleTwoPhaseMixture(U, phi),

    kappa1_
    (
        "kappa1",
        dimEnergy/dimTime/dimLength/dimTemperature,
        subDict(phase1Name_),
        "kappa"
    ),
    kappa2_
    (
        "kappa2",
        kappa1_.dimensions(),
```

```
        subDict(phase2Name_),
        "kappa"
    ),

    Cp1_
    (
        "Cp1",
        dimEnergy/dimTemperature/dimMass,
        subDict(phase1Name_),
        "Cp"
    ),
    Cp2_
    (
        "Cp2",
        dimEnergy/dimTemperature/dimMass,
        subDict(phase2Name_),
        "Cp"
    )

{

}

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

bool Foam::myThermoIncompressibleTwoPhaseMixture::read()
{
    if (incompressibleTwoPhaseMixture::read())
    {
        subDict(phase1Name_).readEntry("kappa", kappa1_);
        subDict(phase2Name_).readEntry("kappa", kappa2_);

        subDict(phase1Name_).readEntry("Cp", Cp1_);
        subDict(phase2Name_).readEntry("Cp", Cp2_);

        return true;
    }

    return false;
}


// ************************************************************************* //
```

myThermoIncompressibleTwoPhaseMixture.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2014-2017 OpenFOAM Foundation
-------------------------------------------------------------------------------

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

#include "myImmiscibleIncompressibleTwoPhaseMixture.H"


// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * //

Foam::myImmiscibleIncompressibleTwoPhaseMixture::
myImmiscibleIncompressibleTwoPhaseMixture
(
    const volVectorField& U,
    const surfaceScalarField& phi
```

```
)
:
    // incompressibleTwoPhaseMixture(U, phi),
    myThermoIncompressibleTwoPhaseMixture(U, phi),
    interfaceProperties(alpha1(), U, *this)
{}


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

bool Foam::myImmiscibleIncompressibleTwoPhaseMixture::read()
{
    return
        // incompressibleTwoPhaseMixture::read()
        myThermoIncompressibleTwoPhaseMixture::read()
     && interfaceProperties::read();
}


// ************************************************************************* //
```

A.2 solidificationInterFoam solver

solidificationInterFoam.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2011-2017 OpenFOAM Foundation
    Copyright (C) 2020 OpenCFD Ltd.
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Application
    SolidificationInterFoam

Group
    grpMultiphaseSolvers

Description
    A solver for two incompressible, immiscible fluids using a Volume of Fluid
    (VOF) phase-fraction-based interface capturing approach, with solidification
    phase change in one of the fluids. The solidification phase change is
    incorporated into the model as a momentum porosity contribution, and the
    energy associated with the phase change is added as an enthalpy contribution
    using the enthalpy-porosity method.
```

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

#include "fvCFD.H"
#include "fvMesh.H"
#include "dynamicFvMesh.H"
#include "CMULES.H"
#include "EulerDdtScheme.H"
#include "localEulerDdtScheme.H"
#include "CrankNicolsonDdtScheme.H"
#include "subCycle.H"
#include "myImmiscibleIncompressibleTwoPhaseMixture.H"
#include "incompressibleInterPhaseTransportModel.H"
#include "turbulentTransportModel.H"
#include "pimpleControl.H"
#include "fvOptions.H"
#include "CorrectPhi.H"
#include "fvcSmooth.H"

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

int main(int argc, char *argv[])
{
    argList::addNote
    (
        "A solver for two incompressible, immiscible fluids using a Volume of Fluid"
        "(VOF) phase-fraction-based interface capturing approach, with solidification"
        "phase change in one of the fluids. The solidification phase change is"
        "incorporated into the model as a momentum porosity contribution, and the"
        "energy associated with the phase change is added as an enthalpy contribution"
        "using the enthalpy-porosity method."
    );

    #include "postProcess.H"

    #include "addCheckCaseOptions.H"
    #include "setRootCaseLists.H"
    #include "createTime.H"
    #include "createDynamicFvMesh.H"
    #include "initContinuityErrs.H"
    #include "createDyMControls.H"
    #include "createFields.H"
    #include "createAlphaFluxes.H"
    #include "initCorrectPhi.H"
    #include "createUfIfPresent.H"

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

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

    while (runTime.run())
    {
        #include "readDyMControls.H"

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

```
        ++runTime;

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

        // --- Pressure-velocity PIMPLE corrector loop
        while (pimple.loop())
        {
            if (pimple.firstIter() || moveMeshOuterCorrectors)
            {
                mesh.update();

                if (mesh.changing())
                {
                    // Do not apply previous time-step mesh compression flux
                    // if the mesh topology changed
                    if (mesh.topoChanging())
                    {
                        talphaPhi1Corr0.clear();
                    }

                    gh = (g & mesh.C()) - ghRef;
                    ghf = (g & mesh.Cf()) - ghRef;

                    MRF.update();

                    if (correctPhi)
                    {
                        // Calculate absolute flux
                        // from the mapped surface velocity
                        phi = mesh.Sf() & Uf();

                        #include "correctPhi.H"

                        // Make the flux relative to the mesh motion
                        fvc::makeRelative(phi, U);

                        mixture.correct();
                    }

                    if (checkMeshCourantNo)
                    {
                        #include "meshCourantNo.H"
                    }
                }
            }

            #include "alphaControls.H"
            #include "alphaEqnSubCycle.H"

            mixture.correct();

            if (pimple.frozenFlow())
            {
                continue;
            }

            #include "UEqn.H"
            #include "TEqn.H"

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

            if (pimple.turbCorr())
            {
```

```
                turbulence->correct();
            }
        }

        runTime.write();

        runTime.printExecutionTime(Info);
    }

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

    return 0;
}


// ************************************************************************* //
```

createFields.H

```
#include "createRDeltaT.H"

Info<< "Reading phaseChangeProperties" << endl; //Add
IOdictionary phaseChangeProperties
(
    IOobject
    (
        "phaseChangeProperties",
        runTime.constant(),
        mesh,
        IOobject::MUST_READ,
        IOobject::NO_WRITE
    )
);
dimensionedScalar Prt //prantdl
(
    "Prt",
    dimless,
    phaseChangeProperties
);

dimensionedScalar L //latent heat
(
    "L",
    dimensionSet(0, 2, -2, 0, 0, 0, 0),
    phaseChangeProperties
);

dimensionedScalar A_mushy //mushy zone constant
(
    "A_mushy",
    dimensionSet(1, -3, -1, 0, 0, 0, 0),
    phaseChangeProperties
);

dimensionedScalar q //constant in Darcy term
(
    "q",
    dimless,
    phaseChangeProperties.lookupOrDefault<scalar>("q", 0.001)
);

dimensionedScalar Tmelt //melting T
(
    "Tmelt",
    dimensionSet(0, 0, 0, 1, 0, 0, 0),
    phaseChangeProperties
);
```

```
dimensionedScalar Tliq //liquidus T
(
    "Tliq",
    dimensionSet(0, 0, 0, 1, 0, 0, 0),
    phaseChangeProperties
);

dimensionedScalar Tsol //solidus T
(
    "Tsol",
    dimensionSet(0, 0, 0, 1, 0, 0, 0),
    phaseChangeProperties
);

dimensionedScalar rhoS //solid rho
(
    "rhoS",
    dimDensity,
    phaseChangeProperties
);

dimensionedScalar CpS //solid Cp
(
    "CpS",
    dimEnergy/dimTemperature/dimMass,
    phaseChangeProperties
);

dimensionedScalar kappaS //solid kappa
(
    "kappaS",
    dimEnergy/dimTime/dimLength/dimTemperature,
    phaseChangeProperties
);


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

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

volScalarField gamma
(
    IOobject
    (
```

```
        "gamma",
        mesh.time().timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    mesh,
    dimensionedScalar("gamma", dimless, 1)
);

// Update gamma using the erf function
gamma = 0.5 * Foam::erf(4.0 * (T - Tmelt) / (Tliq - Tsol)) + 0.5;

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

#include "createPhi.H"

Info<< "Reading transportProperties\n" << endl;
myImmiscibleIncompressibleTwoPhaseMixture mixture(U, phi);

volScalarField& alpha1(mixture.alpha1());
volScalarField& alpha2(mixture.alpha2());

const dimensionedScalar& rho1 = mixture.rho1();
const dimensionedScalar& rho2 = mixture.rho2();

// Need to store rho for ddt(rho, U)
volScalarField rho
(
    IOobject
    (
        "rho",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT
    ),
    alpha1* (gamma*rho1+(1-gamma)*rhoS) + alpha2*rho2

);
rho.oldTime();

// Mass flux
surfaceScalarField rhoPhi
(
    IOobject
    (
        "rhoPhi",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
    ),
    fvc::interpolate(rho)*phi
);

const dimensionedScalar& Cp1 = mixture.Cp1(); //Add
```

```
const dimensionedScalar& Cp2 = mixture.Cp2();

volScalarField Cp //Add
(
    IOobject
    (
        "Cp",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT
    ),
    alpha1* (gamma*Cp1+(1-gamma)*CpS) + alpha2*Cp2
);
Cp.oldTime();

autoPtr<incompressible::turbulenceModel> turbulence
(
    incompressible::turbulenceModel::New(U, phi, mixture)
);

#include "readGravitationalAcceleration.H"
#include "readhRef.H"
#include "gh.H"

volScalarField p
(
    IOobject
    (
        "p",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    p_rgh + rho*gh
);

label pRefCell = 0;
scalar pRefValue = 0.0;
setRefCell
(
    p,
    p_rgh,
    pimple.dict(),
    pRefCell,
    pRefValue
);

if (p_rgh.needReference())
{
    p += dimensionedScalar
    (
        "p",
        p.dimensions(),
        pRefValue - getRefCellValue(p, pRefCell)
    );
    p_rgh = p - rho*gh;
}

mesh.setFluxRequired(p_rgh.name());
mesh.setFluxRequired(alpha1.name());

#include "createMRF.H"
#include "createFvOptions.H"

const dimensionedScalar& kappa1 = mixture.kappa1(); //Add
const dimensionedScalar& kappa2 = mixture.kappa2();
```

```
volScalarField kappa //Add
(
    IOobject
    (
        "kappa",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT
    ),
    alpha1* (gamma*kappa1+(1-gamma)*kappaS) + alpha2*kappa2
);
kappa.oldTime();

volScalarField kappaEff //Add
(
    IOobject
    (
        "kappaEff",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
    ),
    kappa
);

// Need to store rho for ddt(rhoCp, U)
volScalarField rhoCp
(
    IOobject
    (
        "rhoCp",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
    ),
    rho*Cp
);
rhoCp.oldTime();

volScalarField deltaT
(
    IOobject
    (
        "deltaT",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
    ),
    mesh,
    dimensionedScalar(dimTemperature, Zero)
);
```

TEqn.H

```
{
    rhoCp = rho*Cp;
    kappaEff = kappa + rho*Cp*turbulence->nut()/Prt;
    const surfaceScalarField rhoCpPhi = fvc::interpolate(rho * Cp) * phi;

    volScalarField expArg = sqr(4.0 * (T - Tmelt) / (Tliq - Tsol));
    volScalarField Sh_erf = -rho * L * ((4.0 * exp(-expArg) / ((Tliq - Tsol) * ::sqrt(Foam::constant::
     mathematical::pi))) * (fvc::ddt(T) + (U & fvc::grad(T))));

    fvScalarMatrix TEqn
    (
```

```
          fvm::ddt(rhoCp, T)
       + fvm::div(rhoCpPhi, T)
       + alpha1*Sh_erf
       - fvm::Sp(fvc::ddt(rhoCp) + fvc::div(rhoCpPhi), T)
       - fvm::laplacian(kappaEff, T)
    );

    TEqn.relax();
    TEqn.solve();

    Info<< "min/max(T) = " << min(T).value() << ", "
        << max(T).value() <<endl;

    gamma = 0.5 * Foam::erf(4.0 * (T - Tmelt) / (Tliq - Tsol)) + 0.5;
    gamma.correctBoundaryConditions();
    Info << "min/max(gamma) = " << min(gamma).value() << ", "
     << max(gamma).value() << endl;

    // Write the field to the time directory
    if (runTime.writeTime())
    {
        gamma.write();
    }
}
```

UEqn.H

```
    MRF.correctBoundaryVelocity(U);

    volScalarField prosityFunc = A_mushy *sqr(1.0 - gamma)/(pow3(gamma) + q);

    fvVectorMatrix UEqn
    (
        fvm::ddt(rho, U) + fvm::div(rhoPhi, U)
      + alpha1 * fvm::Sp(prosityFunc, U) //Add
      + MRF.DDt(rho, U)
      + turbulence->divDevRhoReff(rho, U)
     ==
        fvOptions(rho, U)
    );

    UEqn.relax();

    fvOptions.constrain(UEqn);

    if (pimple.momentumPredictor())
    {
        solve
        (
            UEqn
         ==
            fvc::reconstruct
            (
                (
                    mixture.surfaceTensionForce()
                  - ghf*fvc::snGrad(alpha1*rho1+alpha2*rho2)
                  - fvc::snGrad(p_rgh)
                ) * mesh.magSf()
            )
        );

        fvOptions.correct(U);
    }
```