Cite as: Olsson, E.: A description of isoAdvector - a numerical method for improved surface sharpness in two-phase flows. In Proceedings of CFD with OpenSource Software, 2017, Edited by Nilsson. H., 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

A description of isoAdvector - a numerical method for improved surface sharpness in two-phase flows

Developed for OpenFOAM v1706

Author:
Elin Olsson
Chalmers University of Technology

Peer reviewed by:
Luofeng Huang
Bercelay Niebles
Mohammad Hossein Arabnejad

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.

January 5, 2018

Learning outcomes

The reader will learn:

How to use it:

- how to use the `interIsoFoam` solver for two-phase flow cases.

The theory of it:

- The theory of the VOF method.
- The theory of some different surface capturing methods used in CFD, with focus on the isoAdvector algorithm

How it is implemented:

- The implementation of the isoAdvector algorithm in OpenFOAM is described.

How to modify it:

- The basic steps for modifying the isoAdvector source code are given.

Prerequisites

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

- Knowledge of fluid dynamics and computational fluid dynamics methods.
- How to run and modify existing tutorial cases in OpenFOAM.

Contents

Chapter 1

Introduction

A popular numerical method for two-phase flows is the volume of fluids (VOF) method. A drawback of the method is that it is difficult to obtain a sharp interface between the phases. Instead a smeared interphase appears between the regions. This problem is avoided by using additional methods for surface capturing. This project is focusing on describing a new such method developed for OpenFOAM, the isoAdvector method.

A main part of the project is to describe the theory behind this method. To give background and context, also the theory for the VOF method is presented, as well as theory behind two other surface capturing methods, MULES and the geometric reconstruction scheme.

The theory is followed by a thorough description of how the theory of isoAdvector is implemented in OpenFOAM. The solver called `interIsoFoam` is a modification of the VOF solver `interFoam` that uses the isoAdvector method. An overview of the source code of the solver is given. This is followed by a short description of how to modify a tutorial case using `interFoam` so that it uses `interIsoFoam` instead.

Finally, the steps for how to modify the source code of the isoAdvector method are provided. This leads to a solver called `interIsoFoamMod`. No further functionality is added in this solver, however this provide a basis for implementing new code and improving the isoAdvector method.

Chapter 2

Theory

In this chapter the theory behind the VOF method is described to give a background and context to the surface capturing methods. The VOF theory is followed by theoretical descriptions of three surface methods, with emphasis on the isoAdvector scheme.

2.1 VOF method

Two-phase flows where the majority of the phases are located in two separate domains, and the surface between them is confined to a small region of the total domain, are referred to as separated free surface flows. Separated free surface flows can be modelled using several different Eulerian methods. In Eulerian methods the fluid properties are calculated in a fixed coordinate system, and the flow is observed at fixed coordinates [1]. The fixed Eulerian coordinate system corresponds to a fixed mesh. The Eularian methods for free surface flows can be classified as surface methods (interface-tracking) or volume methods (interface-capturing).

Surface methods explicitly track the interface between the phases and do in general give a sharp representation of the surface. However, they suffer from drawbacks such as problems with complex surfaces like breaking waves or bubbles [2]. An alternative is to use volume methods, which instead track the two fluid volumes separated by the interface. This enables representation of complex surfaces, however to obtain a sharp surface resolution computationally expensive additional steps are required [3]. A common volume method is the marker and cell (MAC) method [3]. The fluid volume is represented by Lagrangian fluid particles that are distributed over the volume and moved through the Eulerian mesh. The vast number of tracked variables required makes this method significantly more computationally expensive than the surface methods [2]. A computationally inexpensive option that still can handle complex surfaces is the VOF method.

The VOF method is an Eularian volume tracking method where a step function is used to mark the location of the phases (water and air) [2]. This report is limited to two-phase flows with water as the tracked phase and air as the second phase. The step function α marks the volume fraction of the tracked phase in a control volume, so that $\alpha = 1$ corresponds to a control volume entirely occupied by water and $\alpha = 0$ corresponds to a control volume not

containing any water, only air. The value of α is averaged in each of the mesh cells. The interface between the phases is found in cells where $0 < \alpha < 1$.

The function α makes it possible to use only one set of equations in the entire flow domain for describing the local properties, instead of one set of equations for each phase. Fluid properties are calculated using α as a weight. If for example ρ_{water} and ρ_{air} are the densities of the two phases, the density in the entire domain ρ can be described by [3]

$$\rho = \alpha\rho_{water} + (1 - \alpha)\rho_{air} \tag{2.1}$$

Similarly, the dynamic viscosity μ can be obtained by

$$\mu = \alpha\mu_{water} + (1 - \alpha)\mu_{air} \tag{2.2}$$

2.1.1 Governing equations

The flow studied in this project is considered incompressible and the fluids are considered Newtonian. The governing equation for mass (the continuity equation) for a control volume can then be expressed as

$$\frac{\partial\rho}{\partial t} + \nabla \cdot (\rho\boldsymbol{u}) = 0 \tag{2.3}$$

where $\boldsymbol{u}$ is the fluid velocity. The momentum equation is expressed as

$$\frac{\partial(\rho\boldsymbol{u})}{\partial t} + \nabla \cdot (\rho\boldsymbol{u}\boldsymbol{u}) = -\nabla \cdot p + \nabla \cdot \boldsymbol{T} + \rho\boldsymbol{f} + f_\sigma \tag{2.4}$$

where p is the pressure, $\boldsymbol{T}$ is the stress tensor, $\boldsymbol{f}$ represents the body forces and f_σ is the surface tension [4]. In this case, the only present body force is gravity so $\boldsymbol{f}$ can be replaced by $\boldsymbol{g}$, the gravity vector. The VOF method adds one governing equation for the transport of the volume fraction α, the advection equation [3]

$$\frac{\partial\alpha}{\partial t} + \nabla \cdot (\alpha\boldsymbol{u}) = 0 \tag{2.5}$$

In the above equation, the second term is reffered to as the advection term [5]. In OpenFOAM a family of solvers called `interFoam` uses the VOF method. In OpenFOAM version 1706 the group of solvers derived from `interFoam` use the VOF method as described above together with additional methods for surface capturing.

2.2 Surface representation

The main drawback of the VOF method is the smearing of the water surface. Without any additional surface capturing method the surface will be represented as a region where α gradually changes from 1 to 0. The cells containing volume fractions of both phases will not have a sharp surface separating the phase fractions inside the cell, instead the entire cell will be filled with a uniform mixture of the two phases, see Figure **??**. For free surface flows with a sharp interface separating the phases this smeared representation is not physical. After identifying the cells where the surface is present, the challenge lies in determining more precisely where these cells are cut by the surface.

Since the introduction of the VOF method several improvements have been done, especially to the surface resolution. The basic VOF solver `interFoam` uses a scheme called MULES for improving the surface sharpness. In the most recent release of OpenFOAM the newly developed method isoAdvector was introduced, together with the solver `interIsoFoam`. Another common method used in the software ANSYS Fluent is the geometric reconstruction scheme. These methods will be described and compared in more detail in the following sections.

2.2.1 MULES scheme

MULES is a numerical scheme where the advection term in Equation 2.5 is modified to compress the surface and thereby reduce the smearing [5]. The scheme is obtained by firstly rewriting the advection equation to integral form

$$\int_{\Omega_i} \frac{\partial \alpha}{\partial t} dV + \int_{\partial \Omega_i} \alpha \boldsymbol{u} \cdot \boldsymbol{n} dS = 0 \tag{2.6}$$

where Ω_i represents each cell, $\partial\Omega_i$ is the cell boundary and $\boldsymbol{n}$ is the cell boundary normal. The equation is then discretized, using any time-stepping scheme for the first term (here forward Euler is used) and writing the second term as a sum over each face of the cell

$$\frac{\alpha_i^{n+1} - \alpha_i^n}{\Delta t} = -\frac{1}{|\Omega_i|} \sum_{f \in \partial \Omega_i} (F_u + \lambda_M F_c)^n \tag{2.7}$$

where F_u and F_c are the advective fluxes and λ_M is the delimiter taking the value 1 at the surface and 0 elsewhere. These advective fluxes are expressed as

$$F_u = \Phi_f \alpha_{f,upwind} \tag{2.8}$$

and

$$F_c = \Phi_f \alpha_f + \Phi_{rf} \alpha_{rf} (1 - \alpha)_{rf} - F_u \tag{2.9}$$

where Φ_f is the volumetric face flux. The subscript f denotes that the quantity is evaluated as the face, *upwind* that an upwind scheme is used and the quantities with the subsrcipt rf are given below. Φ_{rf} is given by

$$\Phi_{rf} = min\left(C_\alpha \frac{|\Phi_f|}{|\boldsymbol{S}_f|}, max\left[\frac{|\Phi_f|}{|\boldsymbol{S}_f|} \right] \right) (\boldsymbol{n}_f \cdot \boldsymbol{S}_f) \tag{2.10}$$

where C_α is a user specified parameter reducing interface smearing, $\boldsymbol{S}_f$ is the cell face area vector, and $\boldsymbol{n}_f$ is the face centered interface normal vector.

α_{rf} is given by

$$\alpha_{rf} = \alpha_P + \frac{\alpha_N - \alpha_P}{2} [1 - \chi(\Phi_f)(1 - \lambda_{\alpha r})] \tag{2.11}$$

where N and P denotes the upwind and downwind neighbour respectively, $\lambda_{\alpha r}$ is a limiter and χ is a step function taking the value 1 where volumetric face flux i positive and -1 where it is negative.

For $\lambda_M = 0$ which is away from the interface, the expression inside the sum in Equation 2.7 reduces to F_u, which uses a simple upwind scheme for the advection term. For $\lambda_M = 1$

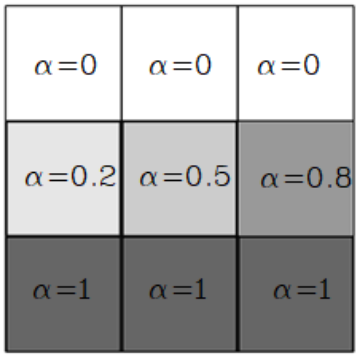
which is at the interface, the expression inside the sum becomes a combination of a higher order scheme for the advection represented by the term $\Phi_f \alpha_f$ and a compressive flux term represented by the term $\Phi_{rf}\alpha_{rf}(1-\alpha)_{rf}$. The higher order scheme gives a more accurate advection at the surface and the compression flux term reduces the surface smearing. At the same time computational effort is reduced away from the surface. Numerical diffusion at the interface is also reduced.

2.2.2 Geometric reconstruction scheme

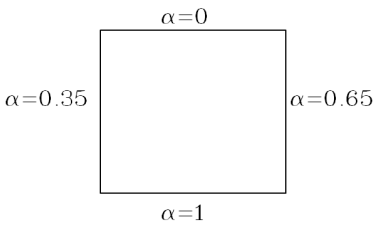
The commercial software ANSYS Fluent provides a couple of different schemes for representing the interface shape. The most general and accurate method is the geometric reconstruction scheme [6]. The interface is represented as a piecewise linear surface, meaning that the surface is linear within each cell. The linear surface of a quadratic mesh in 2 dimensions is constructed as follows [7]

1. The phase fractions for all cells are assumed to be known at the beginning of a time step. The face phase fractions of the faces j of cell i are calculated using the phase fractions of the cells neighbouring each face j. The face phase fraction is calculated as the average of the cell phase fractions. Exceptions are made for faces where this not holds, for example the top and bottom faces in Figure 2.1.
2. The slope of the interface is constructed as the hypotenuse of a right-angled triangle whose other sides are calculated using the face phase fractions. The height is calculated as the difference between the phase fractions of the vertical faces and the width is calculated as the difference between the horizontal faces.
3. Finally the position of the interface is adjusted to match the cell phase fraction.

The above description is illustrated in Figure 2.1 and Figure 2.2.

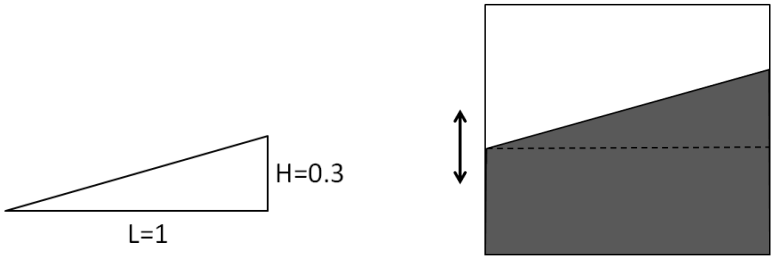


(a) Phase fractions are known in all cells at the beginning of the time step.



(b) Using the known cell phase fractions the face phase fractions are calculated for the faces of each cell. This is the middle cell from (a)

Figure 2.1: The first step of the geometric reconstruction scheme.



(a) The slope of the surface is calculated as the hypotenuse of a triangle using the difference in face phase fractions.

(b) The position of the surface is adjusted to match the known cell phase fraction.

Figure 2.2: The second and third steps of the geometric reconstruction scheme.

2.2.3 isoAdvector scheme

The isoAdvector method uses the concept of isosurfaces to calculate more accurate face fluxes for the cells containing the interface [8]. The value for the phase fraction in cell i at time t, $\alpha_i(t)$, is calculated from a function $H(\boldsymbol{x}, t)$ describing the continuous phase fraction field

$$\alpha_i = \frac{1}{V_i} \int_{\Omega_i} H(\boldsymbol{x}, t) dV \tag{2.12}$$

where V_i is the volume of cell i and Ω_i represents each cell. Knowing the phase fraction in each cell at time t, it is desired to calculate the phase fractions at the next time step using the following equation, where the flux of α over each cell face is integrated in time and added together

$$\alpha_i(t + \Delta t) = \alpha_i(t) - \frac{1}{V_i} \sum_{j \in B_i} s_{ij} \int_t^{t+\Delta t} \int_{F_j} H(\boldsymbol{x}, \tau) \boldsymbol{u}(\boldsymbol{x}, \tau) d\boldsymbol{S} d\tau \tag{2.13}$$

where B_i is the list of all faces F_j belonging to cell i, s_{ij} is used to orient the flux to going out from the cell and τ is the variable of integration used in the time step. $d\boldsymbol{S}$ is the differential area vector pointing out of the volume. s_{ij} is either $+1$ or -1 to ensure that the product $s_{ij} d\boldsymbol{S}$ is always in the direction out form the cell boundary even when the orientation of face j makes $d\boldsymbol{S}$ point into the cell. The integrals inside the sum can be replaced by $\Delta V_j(t, \Delta t)$ which describes the total volume of fluid A transported across face j during one time step

$$\Delta V_j(t, \Delta t) = \int_t^{t+\Delta t} \int_{F_j} H(\boldsymbol{x}, \tau) \boldsymbol{u}(\boldsymbol{x}, \tau) d\boldsymbol{S} d\tau \tag{2.14}$$

This is the quantity that is estimated in the isoAdvector method. It is estimated using the quantities α_i, u_i and Φ_j which are known at time t, where Φ_j is the face flux across face j.

$$\Phi_j(t) = \int_{F_j} \boldsymbol{u}(\boldsymbol{x}, t) d\boldsymbol{S} \tag{2.15}$$

The following algorithm describes the isoAdvector method in more detail.

isoAdvector algorithm

1. For each cell face j, let $\Delta V_j = \alpha_{upwind,j}\Phi_j\Delta t$ at time t. $\alpha_{upwind,j}$ is the face value of α of the upwind cells.
2. Find the surface cells where $0 < \alpha_i(t) < 1$. For all cells not containing the surface the advection is trivial, and the rest of the algorithm described here is used for the surface cells.
3. For each surface cell:

   3.1. Calculate the initial isosurface. The isovalue f which cuts the cell into the correct phase fractions of A and B is sought, so that $\widetilde{\alpha}(f) = \alpha_i$ for each cell. $\widetilde{\alpha}(f)$ is the volume fraction of A when the isovalue f is used to construct the isosurface.

   In order to find f, first the cell volume fractions must be interpolated to the cell vertices. The value is weighed by the inversed cell center-cell vertex distances. These vertex fractions are denoted $f_1...f_N$ for each cell, where N is the number of vertices for the cell. The vertex fractions are used to calculate where the cell is cut by the isosurface f.

   A cell edge is cut if the isovalue f is between the vertex fractions f_k and f_l of that edge, $f_k < f < f_l$. The location where the edge is cut by the isosurface is calculated using linear interpolation:

$$x_{cut} = x_k + \frac{f - f_k}{f_l - f_k}(x_l - x_k) \tag{2.16}$$

   Then $\widetilde{\alpha}$ is calculated for each vertex fraction $f_1...f_N$. The results are used to construct a polynomial expression for $\widetilde{\alpha}$. The sought isovalue can then be found using the iterative Newton's root finding algorithm and $|\widetilde{\alpha}(f) - \alpha_i| < tol$ with a specific tolerance.

   Figure 2.3 shows the initial isosurface in a cell where one corner is submerged in the tracked phase. The locations where the edges are cut by the isosurface are marked.

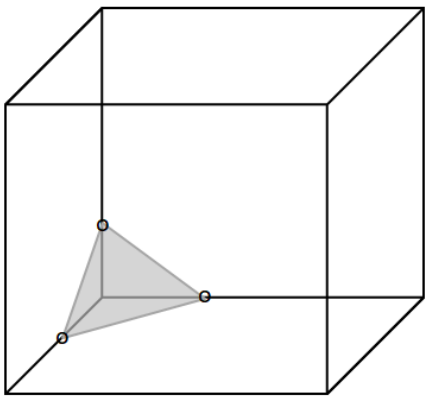


Figure 2.3: The figure shows the initial isosurface in a hexahedral cell where one corner is submerged in the tracked phase. The locations where the edges are cut by the isosurface are marked with circles.

3.2. Estimate the movement of the isosurface during a time step. First, the isosurface center $\boldsymbol{x}_s$ and normal vector $\boldsymbol{n}_s$ are calculated. Then the cell velocity $\boldsymbol{u}_i$ is interpolated to $\boldsymbol{x}_s$, giving the velocity vector $\boldsymbol{U}_s$. Finally, the isosurface motion normal to itself is calculated as

$$U_s = \boldsymbol{U}_s \cdot \boldsymbol{n}_s \tag{2.17}$$

This velocity is assumed to be constant during a time step.

Figure 2.4 shows the propagating isosurface.

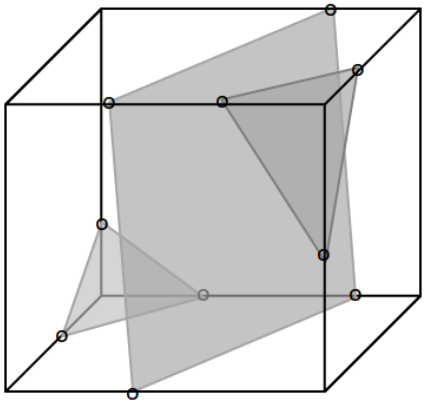


Figure 2.4: The figure shows the propagating isosurface.

3.3. Calculate submerged face area. Using the isosurface velocity calculated above, it can be estimated when the isosurface will reach the face vertex points. The time when vertex k of face j is reached by the isosurface is estimated as

$$t_k \approx t + (\boldsymbol{X}_k - \boldsymbol{x}_s)\frac{\boldsymbol{n}_s}{U_s} \tag{2.18}$$

This can be used to calculate the total face area of one cell that is inside fluid A during one time step $A_j(\tau)$, where τ is the time variable within one time step.

Figure 2.5 shows the propagation of the isosurface on one of the cell faces.

3.4. The time integral of $A_j(\tau)$ over the time step from t to $t + \Delta t$ multiplied by the face flux gives the sought estimate for $\Delta V_j(t, \Delta t)$.

4. To avoid values of α that are < 0 or > 1 a bounding procedure is required. This is done by letting excess of phase A be redistributed to downwind cells in the case of $\alpha > 1$. If $\alpha < 0$, the equations are rewritten in terms of phase B and excess of phase B is redistributed to downwind cells. Redistribution is done to ensure volume conservation. In rare cases strict clipping of the value might be required, however this does not give volume conservation.

After construction the surface, the volumetric flux across each face is divided into flux of A and flux of B proportional to the phase fractions.

In comparative studies isoAdvector is faster and can operate with higher Courant numbers than MULES and still be accurate [8].

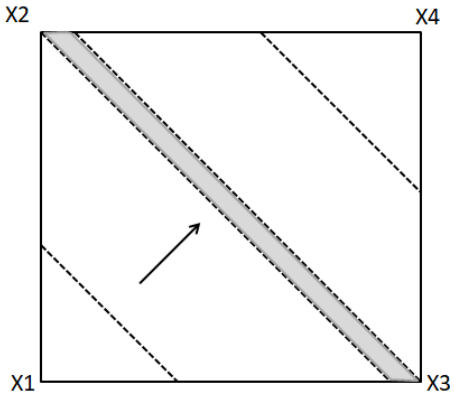


Figure 2.5: The figure shows the propagation of the isosurface on one of the cell faces. Equation 2.18 is used to estimate when the face vertices are reached. The marked region represents the area that is swept between the time when vertex 2 is reached and the time when vertex 3 is reached. This is used to calculate the submerged face area.

Chapter 3

Description of the source code

This chapter describes how the isoAdvector method is implemented in the OpenFOAM solver `interIsoFoam`. `interIsoFoam` is a modified version of the solver `interFoam` which is a solver for incompressible, immiscible and isothermal two-phase flows using the VOF method. In `interIsoFoam` the surface method isoAdvector is added. The implementation is presented with extracts from the code in OpenFOAM and compared with the theoretical description in section 2.2.3.

The source code of the solver `interIsoFoam` is found in the directory `$FOAM_SOLVERS/multiphase/interIsoFoam`.

The following files can be found in this directory:

`alphaControls.H`	`correctPhi.H`	`Make`
`alphaCourantNo.H`	`createFields.H`	`pEqn.H`
`alphaEqn.H`	`createIsoAdvection.H`	`setDeltaT.H`
`alphaEqnSubCycle.H`	`interIsoFoam.C`	`UEqn.H`

The file `interIsoFoam.C` begins with including a number of files in the header

Listing 3.1: interIsoFoam.C

```
56 #include "isoAdvection.H"
57 #include "fvCFD.H"
58 #include "subCycle.H"
59 #include "immiscibleIncompressibleTwoPhaseMixture.H"
60 #include "turbulentTransportModel.H"
61 #include "pimpleControl.H"
62 #include "fvOptions.H"
63 #include "CorrectPhi.H"
```

Some of these require further explanation. The class `isoAdvection.H` calculates the new VOF `alpha` field (i.e. the phase distribution) after a time step given the VOF field `alpha`, velocity field `U` and face fluxes `phi` at the beginning of the time step. It uses the isoAdvector algorithm described in section 2.2.3. The implementation of the algorithm in OpenFOAM will be described further later. The `isoAdvection.H` class is located in `$WM_PROJECT_DIR/src/finiteVolume/fvMatrices/solvers/isoAdvection/isoAdvection`

The class `immiscibleIncompressibleTwoPhaseMixture.H` contains member functions for calculating transport and interface properties. This class is a model for a mixture of two phases that are immiscible and incompressible, and can be found in `$WM_PROJECT_DIR/src/transportModels/immiscibleIncompressibleTwoPhaseMixture`

`turbulentTransportModel.H` contains typedefs for the laminar, RAS and LES turbulence models and is located in `$WM_PROJECT_DIR/src/TurbulenceModels/incompressible/turbulentTransportModels`

`pimpleControl.H` provides member functions for modifying the pimple algorithm in the `fvSolutions` dictionary. This file is found in `$WM_PROJECT_DIR/src/finiteVolume/cfdTools/general/solutionControl/pimpleControl/`

After the header, the main function in `interIsoFoam.C` begins with initializing the case

Listing 3.2: interIsoFoam.C

```
67 int main(int argc, char *argv[])
68 {
69     #include "postProcess.H"
70
71     #include "setRootCase.H"
72     #include "createTime.H"
73     #include "createMesh.H"
74     #include "createControl.H"
75     #include "createTimeControls.H"
76     #include "initContinuityErrs.H"
77     #include "createFields.H"
78     #include "createFvOptions.H"
79     #include "correctPhi.H"
80
81     turbulence->validate();
82
83     #include "readTimeControls.H"
84     #include "CourantNo.H"
85     #include "setInitialDeltaT.H"
```

Most of these files are not specific for the `interIsoFoam` solver, but are general files for initializing variables required for e.g. the time stepping. The file `createFields.H` is included from the `interIsoFoam` solver directory. This file initializes all the variables. The content of this file is presented below

Listing 3.3: createFields.H

```
1    Info<< "Reading field p_rgh\n" << endl;
2 volScalarField p_rgh
3 (
4     IOobject
5     (
6         "p_rgh",
7         runTime.timeName(),
8         mesh,
9         IOobject::MUST_READ,
10        IOobject::AUTO_WRITE
11    ),
12    mesh
```

```
13 );
14
15 Info<< "Reading field U\n" << endl;
16 volVectorField U
17 (
18     IOobject
19     (
20         "U",
21         runTime.timeName(),
22         mesh,
23         IOobject::MUST_READ,
24         IOobject::AUTO_WRITE
25     ),
26     mesh
27 );
28
29 #include "createPhi.H"
30
31
32 Info<< "Reading transportProperties\n" << endl;
33 immiscibleIncompressibleTwoPhaseMixture mixture(U, phi);
34
35 volScalarField& alpha1(mixture.alpha1());
36 volScalarField& alpha2(mixture.alpha2());
37
38 const dimensionedScalar& rho1 = mixture.rho1();
39 const dimensionedScalar& rho2 = mixture.rho2();
40
41
42 // Need to store rho for ddt(rho, U)
43 volScalarField rho
44 (
45     IOobject
46     (
47         "rho",
48         runTime.timeName(),
49         mesh,
50         IOobject::READ_IF_PRESENT
51     ),
52     alpha1*rho1 + alpha2*rho2
53 );
54 rho.oldTime();
55
56
57 // Mass flux
58 surfaceScalarField rhoPhi
59 (
60     IOobject
61     (
62         "rhoPhi",
63         runTime.timeName(),
64         mesh,
65         IOobject::NO_READ,
66         IOobject::NO_WRITE
67     ),
68     fvc::interpolate(rho)*phi
69 );
70 // Construct incompressible turbulence model
71 autoPtr<incompressible::turbulenceModel> turbulence
```

```
72 (
73     incompressible::turbulenceModel::New(U, phi, mixture)
74 );
75
76
77 #include "readGravitationalAcceleration.H"
78 #include "readhRef.H"
79 #include "gh.H"
80
81
82 volScalarField p
83 (
84     IOobject
85     (
86         "p",
87         runTime.timeName(),
88         mesh,
89         IOobject::NO_READ,
90         IOobject::AUTO_WRITE
91     ),
92     p_rgh + rho*gh
93 );
94
95 label pRefCell = 0;
96 scalar pRefValue = 0.0;
97 setRefCell
98 (
99     p,
100     p_rgh,
101     pimple.dict(),
102     pRefCell,
103     pRefValue
104 );
105
106 if (p_rgh.needReference())
107 {
108     p += dimensionedScalar
109     (
110         "p",
111         p.dimensions(),
112         pRefValue - getRefCellValue(p, pRefCell)
113     );
114     p_rgh = p - rho*gh;
115 }
116
117 mesh.setFluxRequired(p_rgh.name());
118 mesh.setFluxRequired(alpha1.name());
119
120 #include "createMRF.H"
121 #include "createIsoAdvection.H"
```

First, the dynamic pressure `p_rgh` and the velocity `U` are initialized in the domain. Then the file `createPhi` which calculates and initializes the relative face-flux field `phi` is included. After that an object of the class `immiscibleIncompressibleTwoPhaseMixture` called `mixture` is created. Then `alpha1` and `alpha2` are created, which are references to `alpha1_` and `alpha2_`. `alpha1_` and `alpha2_` are the phase fraction of each phase 1 and 2. References are also created for the density of each phase. These are then used to calculate the density

`rho` for the entire mixture as seen in line 52:

```
alpha1*rho1 + alpha2*rho2
```

The rest of the file `createFields.H` contains code for calculating the mass flux, constructing the turbulence model and calculating the absolute pressure `p`. In the final line the file `createIsoAdvection.H` is included. This file is also located in the `interIsoFoam` solver directory and creates an object of the class `isoAdvection` called `advector`.

The remaining part of `interIsoFoam` is the following time loop

Listing 3.4: interIsoFoam.C

```
89    Info<< "\nStarting time loop\n" << endl;
90
91     while (runTime.run())
92     {
93         #include "readTimeControls.H"
94
95         #include "CourantNo.H"
96         #include "alphaCourantNo.H"
97         #include "setDeltaT.H"
98
99         runTime++;
100
101        Info<< "Time = " << runTime.timeName() << nl << endl;
102
103        // --- Pressure-velocity PIMPLE corrector loop
104        while (pimple.loop())
105        {
106            #include "alphaControls.H"
107            #include "alphaEqnSubCycle.H"
108
109            mixture.correct();
110
111            if (pimple.frozenFlow())
112            {
113                continue;
114            }
115
116            #include "UEqn.H"
117
118            // --- Pressure corrector loop
119            while (pimple.correct())
120            {
121                #include "pEqn.H"
122            }
123
124            if (pimple.turbCorr())
125            {
126                turbulence->correct();
127            }
128        }
129
130        runTime.write();
131
132        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
133            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
```

```
134             << nl << endl;
135     }
136
137     Info<< "End\n" << endl;
138
139     return 0;
140 }
```

The time loop begins with the inclusion of some more header files, the most interesting in this case is `alphaCourantNo.H`. It calculates the interface Courant number. Then the current time is printed to the screen.

Within each time step the pressure-velocity PIMPLE corrector loop is run. In the beginning, two files are included for calculating the phase fractions `alpha1` and `alpha2`. The first, `alphaControls.H`, is used to look up the number of sub-cycles that are specified for the `alpha` calculation. In the second, `alphaEqnSubCycle.H`, `alpha` is calculated using the specified number of sub-cycles. The calculation is done in the file `alphaEqn.H` (included in `alphaEqnSubCycle.H`) which is also located in the `interIsoFoam` directory. `alphaEqnSubCycle.H` ends with updating the density `rho`.

The file `alphaEqn.H` contains the following lines for updating the phase fraction `alpha1` and the mass flux field `rhoPhi`:

Listing 3.5: alphaEqn.H

```
16     // Update alpha1
17 advector.advect();
18
19 // Update rhoPhi
20 rhoPhi = advector.getRhoPhi(rho1, rho2);
21
22 alpha2 = 1.0 - alpha1;
```

Two member functions of the `advector` object are called, `advector.advect()` and `advector.getRhoPhi(rho1, rho2)`. Both are found in the `isoAdvection.H` class

Listing 3.6: isoAdvection.H

```
301         //- Advect the free surface. Updates alpha field, taking into account
302         //  multiple calls within a single time step.
303         void advect();
```

.
.
.

```
345             //- Return mass flux
346             tmp<surfaceScalarField> getRhoPhi
347             (
348                 const dimensionedScalar rho1,
349                 const dimensionedScalar rho2
350             ) const
351             {
352                 return tmp<surfaceScalarField>
```

```
353                 (
354                     new surfaceScalarField
355                     (
356                         "rhoPhi",
357                         (rho1 - rho2)*dVf_/mesh_.time().deltaT() + rho2*phi_
358                     )
359                 );
360             }
```

The function `getRhoPhi` returns the calculated mass flux. The `advect` function which calculates the `alpha1` field is however only declared in the `isoAdvection.H` file. The definition can be found in the part of the file `isoAdvection.C` presented below.

Listing 3.7: isoAdvection.C

```
973 void Foam::isoAdvection::advect()
974 {
975     DebugInFunction << endl;
976 
977     scalar advectionStartTime = mesh_.time().elapsedCpuTime();
978 
979     // Initialising dVf with upwind values
980     // i.e. phi[facei]*alpha1[upwindCell[facei]]*dt
981     dVf_ = upwind<scalar>(mesh_, phi_).flux(alpha1_)*mesh_.time().deltaT();
982 
983     // Do the isoAdvection on surface cells
984     timeIntegratedFlux();
985 
986     // Synchronize processor patches
987     syncProcPatches(dVf_, phi_);
988 
989     // Adjust dVf for unbounded cells
990     limitFluxes();
991 
992     // Advect the free surface
993     alpha1_ -= fvc::surfaceIntegrate(dVf_);
994     alpha1_.correctBoundaryConditions();
995 
996     // Apply non-conservative bounding mechanisms (clipping and snapping)
997     // Note: We should be able to write out alpha before this is done!
998     applyBruteForceBounding();
999 
1000         // Write surface cell set and bound cell set if required by user
1001     writeSurfaceCells();
1002     writeBoundedCells();
1003 
1004     advectionTime_ += (mesh_.time().elapsedCpuTime() - advectionStartTime);
1005 }
```

Inside this function step 1 of the algorithm in section 2.2.3 can be found on line 981, that is, the initialization of the transported volume `dVf`. The bounding and clipping taking place in step 4 can be found on line 990 and line 998, where `limitFluxes` is the bounding function and `applyBruteForceBounding` is the clipping function. Both these functions are defined in the current file, `isoAdvection.C`. The rest of the algorithm steps can be found in the function called by the `advect` function on line 984, the `timeIntegratedFlux` function. It

is defined earlier in the file `isoAdvection.C`. This function is however too long to include here, and its content is instead summarized below.

The `timeIntegratedFlux` function starts at the beginning of the time step with interpolating the cell center phase fraction `alpha1` to the cell vertices, using the `volPointInterpolation` class. This is corresponding to the beginning of step 3.1 of the algorithm. The isosurface is constructed using functions from the file `isoCutCell.C` located in

`$WM_PROJECT_DIR/src/finiteVolume/fvMatrices/solvers/isoAdvection/isoCutCell`

This file is included in `isoAdvection.H` which is included in `isoAdvection.C`. `isoCutCell.C` is also too long to include here. This file defines the functions used by `timeIntegratedFlux` to construct the isosurface.

The cells which are cut by the surface are identified using the `vofCutCell` function on `isoCutCell_` which is an object of the class `isoCutCell`

Listing 3.8: isoAdvection.C

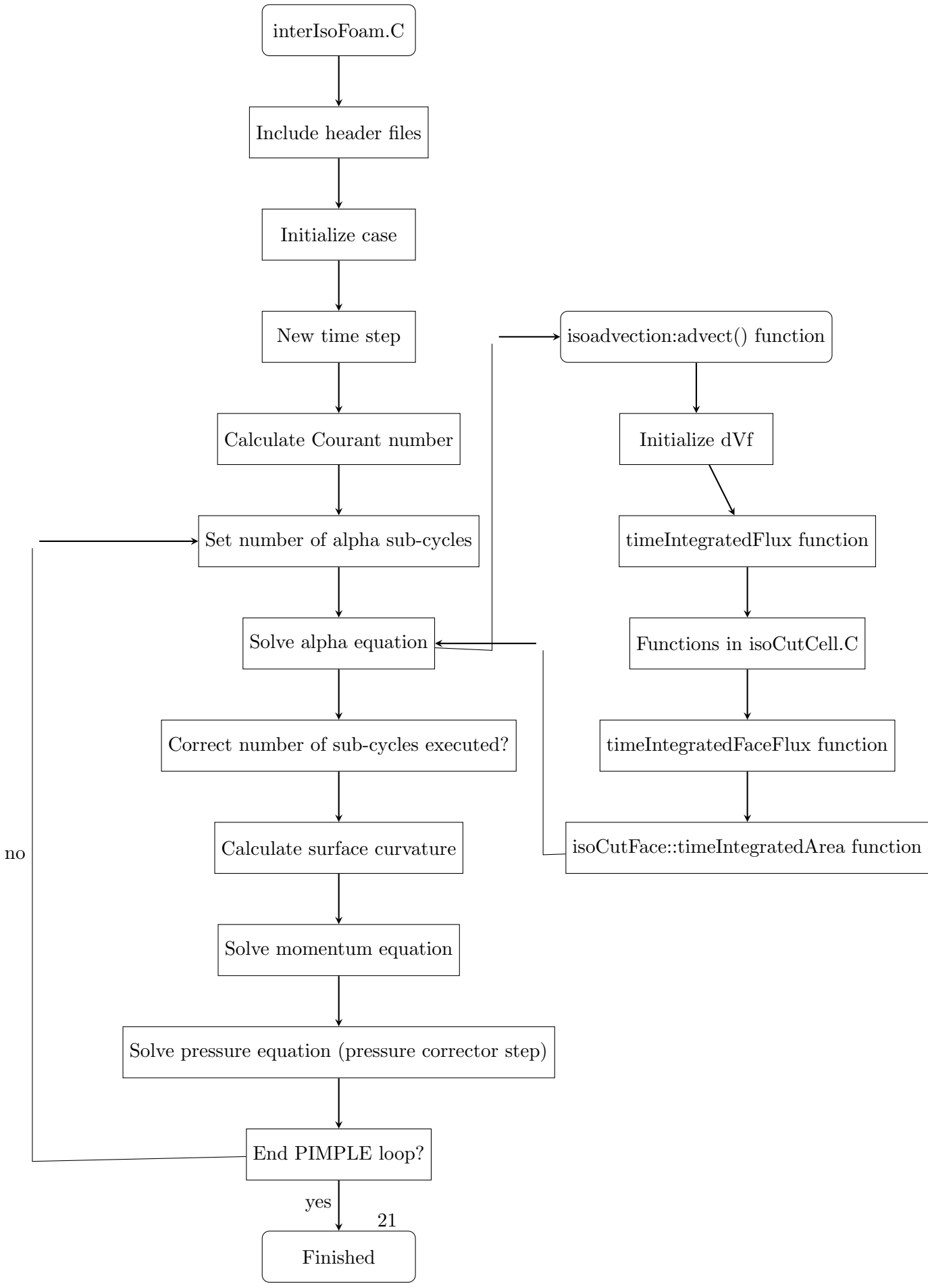
```
263 // Calculate cell status (-1: cell is fully below the isosurface, 0:
264             // cell is cut, 1: cell is fully above the isosurface)
265             label cellStatus = isoCutCell_.vofCutCell
```

The `vofCutCell` function is also the function where the construction of the isosurface, as described in step 3.1, is done. The `vofCutCell` function constructs the polynomial which is used to find the desired value of `alpha1` so that the cell is cut into the right volume fractions. This function returns the `cellStatus`, that is, whether a cell is cut or not by the isosurface. The isovalue itself is accessed by the function `isoValue` of the `isoCutCell.C` file.

For the cells that are cut and thereby located on the isosurface, the isosurface center and normal are calculated. This is done using the functions `isoFaceCentre` and `isoFaceArea` in the `isoCutCell.C` file. These functions returns the result of the function `calcIsoFaceCentreAndArea`.

Then the motion of the isosurface is calculated as the dot product between the velocity and the isosurface normal, as described in step 3.2. This is done inside the `timeIntegratedFlux` function. The time integrated face flux, corresponding to ΔV_j is then calculated by the function `timeIntegratedFaceFlux` as in step 3.4. Inside this function, the `timeIntegratedArea` function of the `isoCutFace.C` file is called. `isoCutFace` is found next to the `isoCutCell` files and is also included in the `isoAdvection.H` file. The `timeIntegratedArea` function calculates the submerged face area by estimating when the face vertex points are reached by the isosurface, as described in step 3.3.

Returning to the pimple loop in `interIsoFoam.C`, the phase fractions have now been updated. The function `mixture.correct()` on line 109 calculates the interface curvature and the laminar viscosity of the mixture. In the rest of the loop the velocity and pressure fields are calculated in the files `UEqn.H` and `pEqn.H`. These are both found in the `interIsoFoam` solver directory. The entire process is summarized in the flowchart below.


interIsoFoam.C
Include header files
Initialize case
New time step
Calculate Courant number
Set number of alpha sub-cycles
Solve alpha equation
Correct number of sub-cycles executed?
Calculate surface curvature
Solve momentum equation
Solve pressure equation (pressure corrector step)
End PIMPLE loop?
yes
no
Finished
isoadvection:advect() function
Initialize dVf
timeIntegratedFlux function
Functions in isoCutCell.C
timeIntegratedFaceFlux function
isoCutFace::timeIntegratedArea function

Chapter 4

Set-up of tutorial case

The solver `interIsoFoam` is used for incompressible, immiscible and isothermal two-phase flow cases. It can be applied to similar cases as the original `interFoam` solver. This chapter provides instructions for how to modify a case using the solver `interFoam` so that it uses `interIsoFoam` instead.

4.1 weirOverflow tutorial case

The `weirOverflow` tutorial case can be found in `$WM_PROJECT_DIR/tutorials/multiphase/interFoam/RAS` and uses the RAS turbulence model `kEpsilon`. Both `interFoam` and `interIsoFoam` can be run as laminar or using LES and RAS turbulence models. This tutorial is copied to the run directory and renamed to `weirOverflowIso`

```
cp -R $WM_PROJECT_DIR/tutorials/multiphase/interFoam/RAS/weirOverflow/ $FOAM_RUN
mv weirOverflow/ weirOverflowIso
```

The directory structure of the tutorial case is as follows:

```
weirOverflowIso
├── 0.orig
│   ├── alpha.water.orig
│   ├── epsilon
│   ├── include
│   │   └── initialConditions
│   ├── k
│   ├── nut
│   ├── p_rgh
│   └── U
├── Allclean
├── Allrun
└── constant
    ├── g
    └── polyMesh
        ├── boundary
        └── faces
```

```
        ├── neighbour
        ├── owner
        └── points
    ├── transportProperties
    └── turbulenceProperties
└── system
    ├── blockMeshDict
    ├── controlDict
    ├── fvSchemes
    ├── fvSolution
    └── setFieldsDict
```

In the `0.orig` directory initial and boundary conditions for the variables can be found. These are not affected by the choice of surface method and can be kept as they are. Before running the case the content of the directory must be copied to a directory named `0` in the case directory. This is just to ensure that clean versions of these files are kept in case modifications are done. The file `alpha.water.orig` must also be copied and renamed to `alpha.water` inside the `0` directory. The original `alpha.water.orig` file is kept so that the boundary and initial conditions used for `alpha.water` can be found after the `setFields` command has been executed. This command adds the initial `alpha.water` field at the top of the file, placing the boundary conditions far down in the file. IF changes are required for the `alpha.water` conditions, these are made in the file `alpha.water.orig` which is the copied and renamed to `alpha.water` before `setFields` is executed. The content of `alpha.water.orig` can be seen below just to give an example of what the files in the `0` directory look like.

Listing 4.1: alpha.water.orig

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  plus                                  |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      alpha.water;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#include        "include/initialConditions"

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
```

```
26      {
27          type            variableHeightFlowRate;
28          lowerBound      0;
29          upperBound      1;
30          value           uniform 0;
31      }
32
33      outlet
34      {
35          type            zeroGradient;
36              }
37
38      lowerWall
39      {
40          type            zeroGradient;
41      }
42
43      atmosphere
44      {
45          type            inletOutlet;
46          inletValue      uniform 0;
47          value           uniform 0;
48      }
49
50      defaultFaces
51      {
52          type            empty;
53      }
54  }
55
56  // ************************************************************************* //
```

Moving on to the `constant` directory, also the content of this directory is unmodified. As mentioned laminar, RAS or LES turbulence models can be chosen and here the RAS model `kEpsilon` is used. The turbulence model is chosen in the file `turbulenceProperties`.

Inside the `system` directory the files `blockMeshDict`, `decomposeParDict` and `setFieldsDict` are kept unmodified as they are not solver dependent. Changes must however be done to the remaining files in this directory.

In the `controlDict` the `application` must be changed to `interIsoFoam`.

In `fvSchemes` a `requiredFlux` object must be added, see the following lines:

```
    fluxRequired
{
    default     no;
    p_rgh;
    pcorr;
    alpha.water;
}
```

The `fluxRequired` section makes the face fluxes of the listed variables available to the solver after the solution of the transport equation. This is necessary for the isoAdvection scheme.

In `fvSolution` some variables must be added to `alpha.water`

```
isofaceTol      1e-6; // Error tolerance on alpha when cutting surface
                      // cells into sub-cells
surfCellTol     1e-6; // Only cells with surfCellTol < alpha < 1-
                      // surfCellTol are treated as surface cells
nAlphaBounds    3; // Number of times the ad-hoc bounding step should
                   // try to correct unboundedness. Strictly volume
                   // conserving (provided that sum(phi) = 0 for a cell).
snapTol         1e-12; // Optional: cells with alpha < snapAlphaTol are
                       // snapped to 0 and cells with 1 - alpha <
                       //  snapAlphaTol are snapped to 1
clip            true; // Optional: clip remaining unboundedness
```

however these variables have default values and the solver runs without them being specified as well. If present, the following variables can be removed from `alpha.water` in `fvSolution`

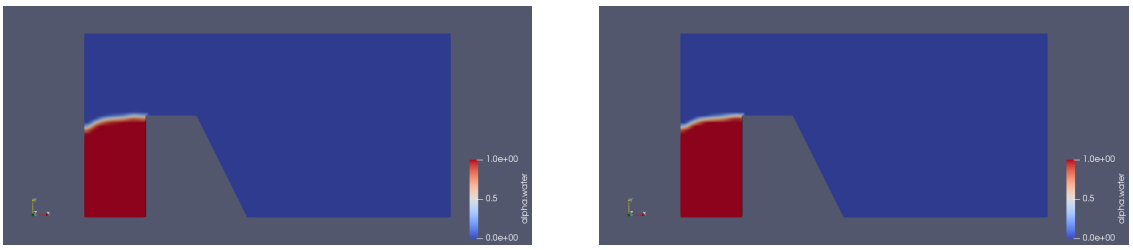
```
nAlphaCorr
MULESCorr
nLimiterIter
solver
smoother
tolerance
relTol
```

as they were used for the MULES scheme. This was the minimum required alterations necessary to run a tutorial case with the `interIsoFoam` solver. More optional changes can be done to e.g. the type of solvers used in `fvSolution` to obtain better results. To run the case, run `blockMesh` and `setFields` to generate the mesh and initial `alpha.water` field, followed by executing `interIsoFoam`.

Both the `weirOverflowIso` tutorial case and the original `weirOverflow` tutorial case were run using default settings for the solvers `interIsoFoam` and `interFoam` respectively. Apart from using different solvers all settings were the same for the tutorial cases. The results for three different time steps are shown below. This is to illustrate the difference between the surface method MULES used by `interFoam` and isoAdvector used by `interIsoFoam`.

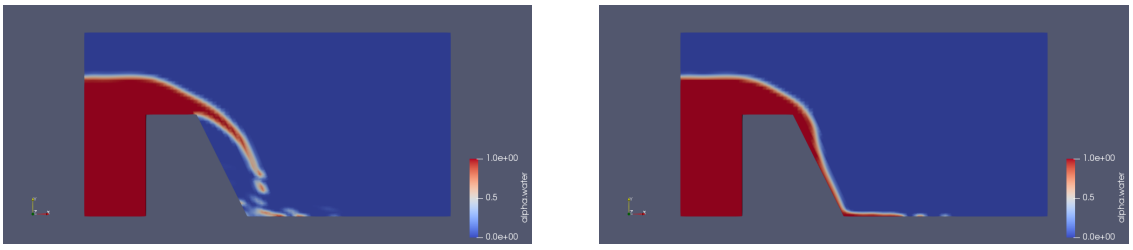
In Figures 4.1, 4.2 and 4.3 the phase fraction field is shown for the two solvers at the times 2s, 8 s and 60 s. No obvious improvement of the surface resolution is visible for the figures obtained from the `weirOverflowIso` case. To improve the surface sharpness further fine tuning of the solver parameters are probably required. It is however interesting to note that for the `interIsoFoam` solver the water flows along the angled wall after passing the weir whereas for the `interFoam` solver it flows out from the wall. Also, some small regions of phase fractions below 1 are observed near the inlet to the left for the `weirOverflowIso` case at later time steps. For many time steps the `interFoam` solver shows regions detached from the rest of the water domain with phase fractions of water between 0 and 1, as can be seen in Figure 4.2. This occurs more rarely for the `interIsoFoam` solver, indicating that the surface sharpening method acts to remove such regions.

More thorough comparisons between MULES and isoAdvector, as well as some other methods, can be found in [8].



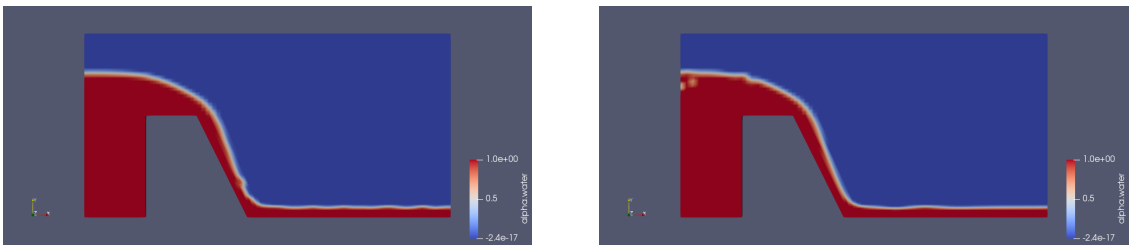
(a) MULES (b) isoAdvector

Figure 4.1: Distribution of the phases at time t=2 s using (a) MULES (interFoam) and (b) isoAdvector (interIsoFoam).



(a) MULES (b) isoAdvector

Figure 4.2: Distribution of the phases at time t=8 s using (a) MULES (interFoam) and (b) isoAdvector (interIsoFoam).



(a) MULES (b) isoAdvector

Figure 4.3: Distribution of the phases at time t=60 s using (a) MULES (interFoam) and (b) isoAdvector (interIsoFoam).

Chapter 5

Modification of source code

As previously mentioned, the source code of the isoAdvector method is located in the directory `$WM_PROJECT_DIR/src/finiteVolume/fvMatrices/solvers/isoAdvection`

Inside this directory, the subdirectories `isoAdvection`, `isoCutFace` and `isoCutCell` are located. Each subdirectory contains the .H and .C files for classes with the same name as te directory. A modification of the solver could be achieved by making a change inside the `isoAdvection.C` file. First, some preparatory work must be done.

To keep the original source code, a copy of the code to be modified is made. Since the `Make` directory is located at a higher level, in the `finiteVolume` directory, this entire directory is copied to the user src directory, and renamed.

```
cd $WM_PROJECT_USER_DIR/src
cp -R $WM_PROJECT_DIR/src/finiteVolume .
mv finiteVolume myFiniteVolume
cd myFiniteVolume
```

Inside the `Make/files` file in the `myFiniteVolume` directory, the final row must be changed to:

```
LIB = $(FOAM_USER_LIBBIN)/libmyFiniteVolume
```

Then, compile using `wmake` while standing in the `myfFiniteVolume` directory. This takes a long time since this directory contains many files.

Descend into the `isoAdvection` directory and copy and rename the `isoAdvection` source code:

```
cd fvMatrices/solvers/isoAdvection/isoAdvection
cp -R isoAdvection isoAdvectionMod
```

Inside the `isoAdvectionMod` directory, rename all files so that the key string "isoAdvection" is followed by "mod"

```
mv isoAdvection.C isoAdvectionMod.C
mv isoAdvection.H isoAdvectionMod.H
mv isoAdvectionTemplates.C isoAdvectionModTemplates.C
```

Then make sure the same change is done inside all of these files:

```
sed -i s/isoAdvection/isoAdvectionMod/g isoAdvection*
```

Go back to the `myFiniteVolume` directory. Inside `Make/files`, add the following line below the line `fvMatrices/solvers/isoAdvection/isoAdvection/isoAdvection.C`:

```
fvMatrices/solvers/isoAdvection/isoAdvectionMod/isoAdvectionMod.C
```

Compile with `wmake libso`.

Changes can now be made to the file `isoAdvectionMod.C`, followed by recompiling with `wmake`. To be able to use the modified source code, the string `isoAdvection` must be altered to `isoAdvectionMod` in the files that use this class. These are all located in the `interIsoFoam` solver directory. It is suitable to create a new solver to keep the original source code. This is done by copying the original `interIsoFoam` solver to the user directory:

```
foam
cp -r --parents  applications/solvers/multiphase/interIsoFoam $WM_PROJECT_
USER_DIR
cd $WM_PROJECT_USER_DIR/applications/solvers/multiphase
mv interIsoFoam interIsoFoamMod
cd interIsoFoamMod
wclean
mv interIsoFoam.C interIsoFoamMod.C
```

Use again the following command to change the name of the modified source code inside all files in the solver directory:

```
sed -i s/isoAdvection/isoAdvectionMod/g *
```

Inside `Make/files` (in the solver directory), change `isoAdvection` to `isoAdvectionMod` so that the file content is:

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

Inside `Make/options`, some more changes need to be done. Add the following line at the top to define a new environment variable:

```
LIB_USER_SRC = $(WM_PROJECT_USER_DIR)/src
```

In `EXE_INC`, add the line

```
-I$(LIB_USER_SRC)/myFiniteVolume/lnInclude \
```

In `EXE_LIBS`, add the lines

```
-L$(FOAM_USER_LIBBIN) \ -lmyFiniteVolume\
```

Finally, compile with `wmake` standing inside the solver directory.

Study questions

1. In short, how does the VOF method simulate two phases?
2. How is the correct isovalue found for each cell in the isoAdvector method?
3. How is the total volume of fluid A transported across a face during one time step calculated in the isoAdvector method, theoretically?
4. How are the files `alphaEqn.H` and `alphaEqnSubCycle.H` related?
5. What is `advector` that appears in the source code used by the `interIsoFoam` solver?
6. What is done by the function `timeIntegratedFaceFlux` and where is it called for?
7. Where can settings for the isoAdvector method be specified for an OpenFOAM case?
8. In what directory is the source code located, that must be modified to modify the isoAdvector method?

Bibliography

[1] M. Ishii and T. Hibiki. *Thermo-fluid dynamics of two-phase flow*. 9th ed. New York, USA: Springer Science+Business Media, Inc., 2006. ISBN: 0-387-28321-8.

[2] C. W. Hirt and B. D. Nichols. Volume of fluid (VOF) method for the dynamics of free boundaries. *Journal of Computational Physics* **39** (1981), 201–25.

[3] V. R. Gopala and B. G. van Wachem. Volume of fluid methods for immiscible-fluid and free-surface flows. *Chemical Engineering Journal* **141**.1 (2008), 204–21.

[4] J. H. Ferziger and M. Perić. *Computational methods for fluid dynamics*. 5th ed. New York, USA: Springer, 2002. ISBN: 3-540-42074-6.

[5] S. S. Deshpande, L. Anumolu, and M. F. Trujillo. Evaluating the performance of the two-phase flow solver interFoam. *Computational Science Discovery* 5 (2012).

[6] *FLUENT 6.3 Documentation*. 2006. URL: `https://www.sharcnet.ca/Software/Fluent6/index.htm` (visited on 10/10/2017).

[7] D. L. Youngs. "Time-dependent multi-material flow with large fluid distortion". *Numerical methods for fluid dynamics*. Ed. by K. W. Morton and M. J. Baines. Academic Press, 1982.

[8] J. Roenby, H. Bredmose, and H. Jasak. A computational method for sharp interface advection. *Royal Society Open Science* **11**.3 (2016).