CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Project work:

Coupled Level-Set with VOF interFoam

Developed for OpenFOAM-2.3.x

Author:
Sankar Menon

Peer reviewed by:
Jethro Nagawkar
Håkan Nilsson

January 15, 2016

1 Introduction

`InterFoam` is the established solver for multiphase flow in OpenFOAM(OF) using the Volume of Fluid (VOF) method. This report explains a new methodology/formulation of coupling the Level Set (LS) method with the VOF method. Currently the implementations are done in the OpenFoam version 2.3.x. The reader is required to have basic understanding of the `interFoam` solver and linux platform to follow this report.

This report starts of with a background and motivation for implementing new solvers in section 2. This section also explains the numerical equations to implement the solver. The numerical algorithm for solvers is shown in section 3. In Section 4, the reader can navigate through the files of the downloaded solvers with the explanation of functionality of each of them. In section 5, case files required to set up a tutorial case for the solvers are explained. In section 6, the solvers are compiled and tutorial case is run with the solvers. Finally, the results and future work, are presented in section 7 and 8 respectively.

2 Background and motivation

Interface capturing of two or more fluids have been a challenge in computational multiphase simulations. Volume of Fluid (VOF) method has been used for many applications of multiphase and still considered to be valid for many computations because of its simplicity and flexibility. One of main drawback of the VOF is smearing of interface. This may have an impact of the results, in particular for cases where where the surface tension is dominant. In VOF method, a volume fraction variable, α, varies from 0 to 1 to represent different phases as shown in Table 1. The physical properties of

Volume fraction	
Phase 1	$\alpha = 1.$
Phase 2	$\alpha = 0.$
Interface	$0 < \alpha < 1$

Table 1: Volume fraction

the two phases are given in `<casefolder>/system/ transportProperties`

The mixture material properties is given as,

$$\begin{aligned} \rho &= \alpha\rho_l + (1-\alpha)\rho_a \\ \mu &= \alpha\mu_l + (1-\alpha)\mu_a \end{aligned} \tag{1}$$

where ρ and μ is the is the density and viscosity, respectively, of the mixture. Here phase 1 is taken as liquid and phase 2 as air with subscript l and a respectively. The advection equation for the interphase capturing is given by,

$$\frac{\partial \alpha}{\partial t} + \frac{\partial \alpha v_j}{\partial x_j} - \alpha\frac{\partial v_j}{\partial x_j} = 0, \tag{2}$$

which can be simplified using the continuity equation as,

$$\frac{\partial \alpha}{\partial t} + \frac{\partial \alpha v_j}{\partial x_j} = 0. \tag{3}$$

The challenging aspect of this advection equation is to have a sharp interface while maintaining both boundedness and mass conservativeness. OF uses an additional counter-gradient convection based term which compresses the interface, while maintaining boundedness and conservativeness [1]. The advection equation (3) is rewritten as

$$\frac{\partial \alpha}{\partial t} + \frac{\partial \alpha v_j}{\partial x_j} + \frac{\partial v_j^c \alpha\beta}{\partial x_j} = 0, \tag{4}$$

where v^c ensures compression ($v^c = v^l - v^g$, l and g stands for liquid and gas, respectively), while the $\partial/\partial x_j$ guarantees conservation and $\alpha\beta$ guarantees boundedness ($\beta = 1 - \alpha$). This counter gradient compression term is implemented in the `alphaEqn.H` of the `interFoam` solver. The fluxes are limited and corrected using the `MULES` algorithm in OF [2].

Another popular interface capturing method is the level set method. The LS method was first developed by Osher and Sethian [3] and later introduced to multiphase flows by Sussman et al. [4]. It is basically a signed distance function, ϕ, to distinguish between two fluids in the mixture. It has a positive value in one fluid and a negative value in the other fluid. The interface is defined by the iso-surface $\phi = 0$. The interface is advected by solving a transport equation with an imaginary time variable. However, the LS function ceases to act as distance function after the first step and thus a re-initialisation process is required to recover it. This makes it mass non-conservative as shown by Sussman et al. [4].

The coupling implemented in the present work takes advantage of the mass conservation of the VOF method and the sharp interface capturing of LS method. Although two separate fields are defined, the VOF advection equation is solved instead of both the VOF and LS equations as required in the standard CLSVOF [5]. The details of the theoretical formulation and numerical procedure are described in Albadawi et al.[6].

In the results below there are two variants of CLSVOF, one where only the surface tension is corrected and another where a heaviside function is used to correct the viscosity and density. For the sake for simplicity and nomenclature, the former is denoted CLSVOFsf and the latter CLSVOF. CLSVOFsf (sf stands for surface force) is derived from Yamamoto [7] and modified to OF 2.3.x.

2.1 Level Set and Heaviside function

The first step is to initialize the value for the LS function from the VOF (α) field, as

$$\phi_0 = (2\alpha - 1)\Gamma, \tag{5}$$

where $\Gamma = 0.75\Delta x$ and Δx is the mesh cell size. The initial value is a signed distance function, with a positive value in the liquid and a negative value in the gas.

The LS is then re-distanced by solving the re-initialisation equation, 6.

$$\begin{gathered} \frac{\partial \phi}{\partial \tau} = S(\phi_0)(1 - \nabla\phi) \\ \phi(x, 0) = \phi(x) \end{gathered} \tag{6}$$

where τ is the artificial time which is chosen as $0.1\Delta x$. The solution converges to the $|\Delta\phi| = 1$. The re-initialisation given a smooth distance function win convergence within a few iterations where iteration number, $\phi_{corr} = \epsilon/\Delta\tau$. Here $\epsilon = 1.5\Delta x$ is the interface thickness needed.

The surface tension is calculated as

$$\mathbf{F}_\sigma = \sigma\kappa(\phi)\delta(\phi)\nabla\phi, \tag{7}$$

where σ is the surface tension coefficient, $\kappa(\phi)$ is the curvature‘ and δ is the Dirac function to limit the influence of surface tension within the interface and takes zero in both fluids, defined as

$$\delta(\phi) = \begin{cases} 0 & \text{if } |\phi| > \epsilon \\ \dfrac{1}{2\epsilon}\left(1 + cos\left(\dfrac{\pi\phi}{\epsilon}\right)\right) & \text{if } |\phi| \leq \epsilon \end{cases} \tag{8}$$

The physical properties can be calculated using the Heaviside function.

$$H(\phi) = \begin{cases} 0 & \text{if } \phi < -\epsilon \\ \dfrac{1}{2}\left[1 + \dfrac{\phi}{\epsilon} + \dfrac{1}{\pi}sin\left(\dfrac{\pi\phi}{\epsilon}\right)\right] & \text{if } |\phi| \geq \epsilon \\ 1 & \text{if } \phi > \epsilon \end{cases} \tag{9}$$

This heaviside function is used to calculate the physical properties instead of the α variable in equation 1.

3 Numerical methodology

The numerical algorithm is quite similar to that in `interFoam` solver, starting off with the initialisation of velocity, pressure and α fields. Then the LS variable, ϕ, and coupling variable, δ and H, are initialized from the initial α before the time loop (equations 5, 6, 8 and 9). Then, the PIMPLE loop starts with advection of α, (equation 4), and correcting the density and viscosity with it. The ϕ field is reconstructed from the α field and Dirac, δ and heaviside function, H are computed.

From here the solvers differ for CLSVOFsf and CLSVOF. In the former solver the surface tension is only corrected (from ϕ as in equation 7) before returning to the main time loop. In the latter solver, the physical properties are also corrected before returning to the main time loop. The solver then advances to the velocity equation or momentum predictor equation and then to the PISO pressure correcting loop till convergence criteria is reached. Figure 1 illustrates the algorithm of the new solvers.

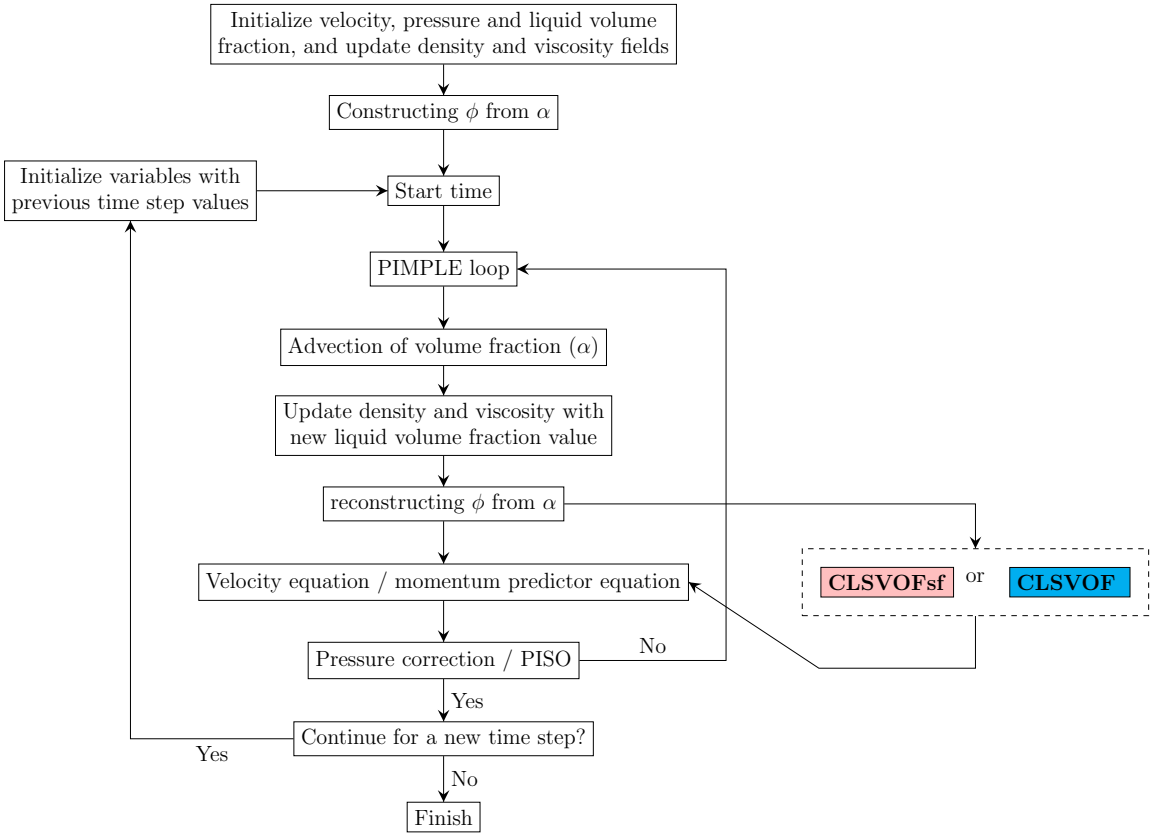


Figure 1: Algorithm for the two solver versions. Here for the CLSVOFsf version of the solver, only the surface tension is corrected from the LS (ϕ) variable and for CLSVOF, the density and viscosity are corrected with the heaviside variable alongside surface tension.

4 Implementation

Quite understandably, CLSVOFsf solver is termed `sclsVOFFoamsf` and CLSVOF is `sclsVOFFoam`. Two solvers as sclsVOFFoam.tar.gz and sclsVOFFoamsf.tar.gz are available from the site [1]. The

[1] http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2015/

files and their functionalities are explained in these solvers will be explained in this section. Make the parent folders for the solver in the user directory with [2],

```
OF23x
mkdir -p $WM_PROJECT_USER_DIR/applications/solvers/multiphase
```

From section 3 it can be realized that the basic algorithm of `sclsVOFFoam` and `sclsVOFFoamsf` is quite similar to `interFoam`. Thus the solvers are created from the basic skeleton of `interFoam` solver. Some additions are made to the `interFoam` solver to achieve `sclsVOFFoamsf` solver, which will be explained in the section 4.1. Some more additions to the files will achieve the `sclsVOFFoam` solver, which will be explained later in the section 4.2.

4.1 CLSVOFsf solver

Supposed that the solver tar packages is downloaded to the `~/Downloads/` folder, it can moved and extracted in the OF user path by,

```
mv ~/Downloads/sclsVOFFoamsf.tar.gz $WM_PROJECT_USER_DIR/applications/solvers/multiphase/
cd $WM_PROJECT_USER_DIR/applications/solvers/multiphase
tar -zxvf sclsVOFFoamsf.tar.gz
rm -f sclsVOFFoamsf.tar.gz
cd sclsVOFFoamsf
```

The linux command `ls -lR` will list the files in the folder, which is shown in Listing 1. When compared to the files in the `interFoam` solver[3], the file `interFoam.C` is renamed to `sclsVOFFoam.C` and four new files are added. The file `mappingPsi.H` is used of initializing the LS field (ϕ_0), `solveLSFunction.H` file reinitialize ϕ and computes δ and H, `calcNewCurvature.H` file computes the new curvature, $\kappa(\phi)$ and finally, `updateFlux.H` file recompute the fluxes. Detailed explanation of these files and other changes from the `interFoam` solver will be addressed in the succeeding sections. However, explanation of functionality of each file that is already existing in the `interFoam` solver is beyond the scope of this report.

Listing 1: Files in sclsVOFFoamsf solver

```
|-- Make
|   |-- files
|   |-- options
|-- alphaCourantNo.H
|-- alphaEqn.H
|-- alphaEqnSubCycle.H
|-- correctPhi.H
|-- createFields.H
|-- pEqn.H
|-- UEqn.H
|-- setDeltaT.H
|-- sclsVOFFoamsf.C
|-- mappingPsi.H
|-- solveLSFunction.H
|-- calcNewCurvature.H
|-- updateFlux.H
```

4.1.1 Initializing the case

Fields and constants required to implement the solver are initialized in `createFields.H` file. The additional fields and constants to implement the new solver is shown in Table 2. They are defined in `createFields.H` file as shown in the Listings 2 and 3.

These Listings show only the additional code that is added to `createFields.H`, in corresponding line numbers shown along the left margin. The fields are initialized with `volScalarField` class, as

[2]The below commands are executed in the terminal of an Linux based OS (ex. Ubuntu)
[3]ls -l $FOAM_SOLVERS/multiphase/interFoam

Fields	
`psi`, ϕ	LS function
`psi0`, ϕ_0	initial LS function
H	heaviside
`delta`, δ	Dirac function
C	curvature
Constants	
`deltaX`, Δx	mesh cell size,
`gamma`, γ	small non-dimensional number of initializing LS,
`epsilon`, ϵ	interface thickness,
`deltaTau`, $\Delta\tau$	artificial time step,
`dimChange`	for ϕ re-initialisation fraction,
`sigma`, σ	recalled since the surface tension is calculated with LS.
`nu1` ν_1 and `nu2`, ν_2	recalled for viscosity correction for CLSVOF

Table 2: Fields and constants initialized for implementing new solver

shown in Listing 2. Only the volume scalar `psi` has `IOobject::MUST_READ` (line 24 in Listing 2), since the values are read from the mesh. Other fields, `psi0`, `delta` and `H`, has `IOobject::NO_READ` (lines 9, 38, 53, 67 from Listing 2), since their values are computed. This means that, the field ϕ should be defined in the starting time directory of the case file that is running this solver, but not for other fields. All the fields have `IOobject::AUTO_WRITE`, to write these field values in all time directories as specified by `writeControl` in `system/controlDict` in the case folder.

Listing 2: Fields initialized in createFields.H

```
1   Info<< "Reading field psi0\n" << endl;
2   volScalarField psi0
3   (
4       IOobject
5       (
6           "psi0",
7           runTime.timeName(),
8           mesh,
9           IOobject::NO_READ,
10          IOobject::AUTO_WRITE
11      ),
12      mesh,
13      dimensionedScalar("psi0",dimless, 0.0)
14  );
15
16  Info<< "Reading field psi\n" << endl;
17  volScalarField psi
18  (
19      IOobject
20      (
21          "psi",
22          runTime.timeName(),
23          mesh,
24          IOobject::MUST_READ,
25          IOobject::AUTO_WRITE
26      ),
27      mesh
28  );
29
30  Info<< "Reading field delta\n" << endl;
31  volScalarField delta
32  (
33      IOobject
34      (
35          "delta",
36          runTime.timeName(),
```

```
37              mesh,
38              IOobject::NO_READ,
39              IOobject::AUTO_WRITE
40          ),
41          mesh,
42          dimensionedScalar("delta",dimless, 0.0)
43      );
44
45      Info<< "Reading field H\n" << endl;
46      volScalarField H
47      (
48          IOobject
49          (
50              "H",
51              runTime.timeName(),
52              mesh,
53              IOobject::NO_READ,
54              IOobject::AUTO_WRITE
55          ),
56          mesh,
57          dimensionedScalar("H",dimless, 0.0)
58      );
59
60      volScalarField C
61      (
62          IOobject
63          (
64              "C",
65              runTime.timeName(),
66              mesh,
67              IOobject::NO_READ,
68              IOobject::AUTO_WRITE
69          ),
70          mesh,
71          dimensionedScalar("C",dimless/dimLength, 0.0)
72      );
```

The constants in Table 2 are defined with `dimensionedScalar` class, as shown in the Listing 3. Constants `gamma`,`deltaTau` and `dimChange` are "hard-coded" in the solver with giving them a specific value. However, for constants `deltaX`, `epsilon` and `sigma` it is specified with `transportProperties.lookup` to look up for respective values in `constant/transportProperties` of the case folder. So the user needs to specify that when running the case (line 36 and 37 in Listing 20).

Listing 3: New constants initialized in createFields.H

```
222
223     dimensionedScalar deltaX
224     (
225         transportProperties.lookup("deltaX")
226     );
227
228     dimensionedScalar gamma
229     (
230         dimensionedScalar(deltaX*0.75)
231     );
232
233     dimensionedScalar epsilon
234     (
235         dimensionedScalar(deltaX*3.5)
236     );
237
238     dimensionedScalar deltaTau
239     (
240         dimensionedScalar(deltaX*0.1)
241     );
242
```

```
243     dimensionedScalar dimChange
244     (
245         dimensionedScalar("dimChange",dimLength, 1.0)
246     );
247
248     dimensionedScalar sigma
249     (
250         transportProperties.lookup("sigma")
251     );
```

The files `mappingPsi.H`, `solveLSFunction.H`, `calcNewCurvature.H` are called twice in the main file, `sclsVOFFoamsf.C`. Firstly it is called to initialize the fields and constants for LS computations. The initialization lines are included before the time loop in the solver in which the new fields are computed from the initial values of VOF field, α. Then again, these three files along with `updateFlux.H`, are called inside the PIMPLE loop to recompute them. These new fields are computed after the solving the advection of α (`alphaControls.H` and `alphaEqnSubCycle.H`) and before the momentum equation, as suggested in the section 3.

The LS function, ϕ, is initialized using α, as written in the equation 5, with the file `mappingPhi.H`, shown in Listing 4.

Listing 4: mappingPsi.H

```
1  // mapping alpha value to psi0
2     psi0 == (double(2.0)*alpha1-double(1.0))*gamma;
```

The LS function ϕ is then re-initialized in the file `solveLSFunction.H`, as per equation 6, as shown in Listing 5.

Listing 5: solveLSFunction.H

```
5     psi == psi0;
6
7     for (int corr=0; corr<int(epsilon.value()/deltaTau.value()); corr++)
8     {
9        psi = psi + psi0/mag(psi0)*(double(1)-mag(fvc::grad(psi)*dimChange))*
             deltaTau;
10       psi.correctBoundaryConditions();
11    }
```

After computing ϕ, the Dirac and heaviside functions are computed in same file as shown in Listing 6.

Listing 6: solveLSFunction.H

```
13
14 // update Dirac function
15    forAll(mesh.cells(),celli)
16    {
17       if(mag(psi[celli]) > epsilon.value())
18          delta[celli] = double(0);
19       else
20          delta[celli] = double(1.0)/(double(2.0)*epsilon.value())*(double(1.0)+
                Foam::cos(M_PI*psi[celli]/epsilon.value()));
21    };
22
23 // update Heaviside function
24    forAll(mesh.cells(),celli)
25    {
26       if(psi[celli] < -epsilon.value())
27          H[celli] = double(0);
28       else if(epsilon.value() < psi[celli])
29          H [celli] = double(1);
30       else
31          H[celli] = double(1.0)/double(2.0)*(double(1.0)+psi[celli]/epsilon.value
                ()+Foam::sin(M_PI*psi[celli]/epsilon.value())/M_PI);
32    };
```

Curvature is calculated based on the LS function by the file `calcNewCurvature.H`, shown in Listing 7.

Listing 7: calcNewCurvature.H

```
// calculate normal vector
volVectorField gradPsi(fvc::grad(psi));
surfaceVectorField gradPsif(fvc::interpolate(gradPsi));
surfaceVectorField nVecfv(gradPsif/(mag(gradPsif)+scalar(1.0e-6)/dimChange));
surfaceScalarField nVecf(nVecfv & mesh.Sf());

// calculate new curvature based on psi (LS function)
   C == -fvc::div(nVecf);
```

The counter-gradient convective fluxes are recomputed using new LS normal vector in the file `updateFlux.H` as shown in Listing 8.

Listing 8: updateFlux.H

```
{
    word alphaScheme("div(phi,alpha)");
    word alpharScheme("div(phirb,alpha)");
        // Standard face-flux compression coefficient
    surfaceScalarField phic(mixture.cAlpha()*mag(phi/mesh.magSf()));
    //surfaceScalarField phic(mag(phi/mesh.magSf()));

    // Add the optional isotropic compression contribution
    if (icAlpha > 0)
    {
        phic *= (1.0 - icAlpha);
        phic += (mixture.cAlpha()*icAlpha)*fvc::interpolate(mag(U));
    }

    // Do not compress interface at non-coupled boundary faces
    // (inlets, outlets etc.)
    forAll(phic.boundaryField(), patchi)
    {
        fvsPatchScalarField& phicp = phic.boundaryField()[patchi];

        if (!phicp.coupled())
        {
            phicp == 0;
        }
    }

    surfaceScalarField phir(phic*nVecf);

    surfaceScalarField phiAlpha
    (
        fvc::flux
        (
            phi,
            alpha1,
            alphaScheme
        )
      + fvc::flux
        (
            -fvc::flux(-phir, scalar(1) - alpha1, alpharScheme),
            alpha1,
            alpharScheme
        )
    );

        MULES::explicitSolve(alpha1, phi, phiAlpha, 1, 0);

        rhoPhi = phiAlpha*(rho1 - rho2) + phi*rho2;
}
```

Inside the files `UEqn.H` and `pEqn.H` in the solver folder, the new surface tension force is recomputed using the new curvature as shown in Listings 9 and 12.

Listing 9: UEqn.H

```
            UEqn
         ==
            fvc::reconstruct
            (
                (
//LS surface tension
                    sigma*fvc::snGrad(psi)*fvc::interpolate(C)*fvc::interpolate(
                        delta)
                  - ghf*fvc::snGrad(rho)
                  - fvc::snGrad(p_rgh)
                ) * mesh.magSf()
            )
        );
         fvOptions.correct(U);
    }
```

Listing 10: pEqn.H

```
    surfaceScalarField phig
    (
        (
//LS surface tension
            sigma*fvc::snGrad(psi)*fvc::interpolate(C)*fvc::interpolate(delta)
          - ghf*fvc::snGrad(rho)
        )*rAUf*mesh.magSf()
    );
```

4.2 CLSVOF solver

In this section, solver CLSVOF solver is explained, wherein the viscosity and density are corrected. Here the additional code that is added when comparing to the previous CLSVOFsf is only explained. Download the solver file `sclsVOFFoam.tar.gz` in the same in the path as the previous solver i.e , `$WM_PROJECT_USER_DIR/applications/solvers/multiphase` and extract the solver.

```
cd $WM_PROJECT_USER_DIR/applications/solvers/multiphase
tar -zxvf sclsVOFFoam.tar.gz
rm -f sclsVOFFoam.tar.gz
cd  sclsVOFFoam
```

The viscosity fields for the both the phases are initialized in the `createFields.H` file, as shown in Listing 11

Listing 11: Viscosity initialized in createFields.H

```
    dimensionedScalar nu1
    (
        transportProperties.subDict("water").lookup("nu")
    );

    dimensionedScalar nu2
    (
        transportProperties.subDict("air").lookup("nu")
```

The density is recomputed by adding,

```
const volScalarField limitedH
   (
       "limitedH",
```

```
        min(max(H, scalar(0)), scalar(1))
    );
rho == limitedH*rho1 + (1.0 - limitedH)*rho2;
```

at the end of `solveLSFunction.H`. Here `volScalarField limitedH` is used to limit the value of `H` within zero and unity, as defined inside it.

The viscosity is recomputed by adding,

```
volScalarField& nuTemp = const_cast<volScalarField&>(mixture.nu()());
nuTemp == limitedH*nu1 + (1.0 - limitedH)*nu2;
```

at the end of file `solveLSFunction.H`.

The heaviside is kept in the limited region of $0 < H < 1$ by adding
`H == limitedH;`
in `solveLSFunction.H`.

The flux in the advection equation 4, needs to be computed from the heaviside function. The flux `PhiH` is computed in the `updateFlux.H` file by adding following instead of `Phialpha`.

Listing 12: updateFlux.H

```
32      surfaceScalarField phiH
33      (
34          fvc::flux
35          (
36              phi,
37              H,//alpha1,
38              alphaScheme
39          )
40        + fvc::flux
41          (
42              -fvc::flux(-phir, scalar(1) - H, alpharScheme),
43              H,//alpha1,
44              alpharScheme
45          )
46      );
47
48      //MULES::explicitSolve(alpha1, phi, phiAlpha, 1, 0);
49      MULES::explicitSolve(H, phi, phiH, 1, 0);
50
51      rhoPhiH = phiH*(rho1 - rho2) + phi*rho2;
52  }
```

Notice the difference compared to Listing 8 for CLSVOFsf. This `PhiH` is then fed into the `UEqn.H` file as shown in Listing 13

Listing 13: UEqn.H

```
3           fvm::ddt(rho, U)
4         + fvm::div(rhoPhiH, U)
5         + turbulence->divDevRhoReff(rho, U)
```

Sometimes when there is too much smearing of α, especially during coalescence of two bubbles, computing the heaviside becomes almost impossible for the solvers. To avoid that α is overwritten with the heaviside at small intervals of time steps as,

```
// reInitialise the alpha equation
if (runTime.outputTime())
  {
    Info<<"Overwriting alpha" << nl << endl;
    alpha1 = H;
        volScalarField& alpha10 = const_cast<volScalarField&>(alpha1.oldTime());
    alpha10 = H.oldTime();
  }
```

in `sclsVOFFoam.C` just after the PIMPLE loop. There is also a need to save the old time values for the heaviside function by adding `H.storeOldTime();` before the PIMPLE loop but inside the time loop. Look inside `sclsVOFFoam.C` to see these implementation.

5 Bubble column tutorial

Download the accompanying case folder to the OF run directory (`$FOAM_RUN`) and extract it.

```
tar -zxvf bubblecol.tar.gz
cd bubblecol
```

The case is taken from quantitative experiments conducted by Hysing et al. [8]. The geometry, initial configuration and boundary conditions is shown in Figure 2. The case is tested for laminar flow. The physical parameters used are shown in Table 3

Test case	ρ_l	ρ_a	μ_l	μ_a	g	σ
	1000	1	10	0.1	0.98	1.96

Table 3: Physical parameters defining the test case

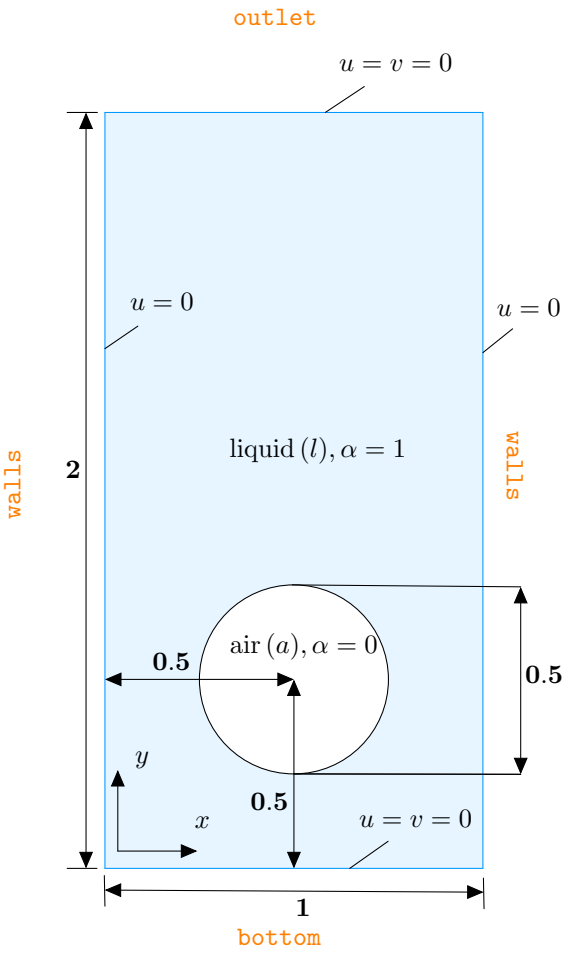


Figure 2: Initial configuration and boundary conditions for the test case.

5.1 Geometry and Mesh

The geometry of the case is given in `constant/polyMesh/blockMeshDict`, shown in Listing 14. The mesh has 160 cells in the x direction and 160×2 cells in the y direction. From Figure 2, the left and right sides are named `walls`, the top is named `outlet` and the bottom is named `bottom`.

Listing 14: blockMeshDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.3.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
    (0 0 0)
    (1 0 0)
    (1 2 0)
    (0 2 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 2 0.1)
    (0 2 0.1)
);

blocks
(
     hex (0 1 2 3 4 5 6 7) (160 320 1) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
    bottom
    {
        type wall;
        faces
        (
            (1 5 4 0)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (3 7 6 2)
        );
    }
    walls
    {
        type wall;
        faces
        (
            (0 4 7 3)
            (2 6 5 1)
        );
    }
```

```
67 );
68
69 mergePatchPairs
70 (
71 );
72
73 // ************************************************************************* //
```

5.2 Initialisation and boundary condition

Boundary condition and initial values are specified in the `0` directory. The `0` directory has `alpha.water`, `alpha.water.org`, `p_rgh`,`psi` ,`U`.

`alpha.water` has `zeroGradient` for all sides, as shown in Listing 15. Only the sides in the z direction (perpendicular to the plane in Figure 2) are given as `empty` to have a 2D case.

Listing 15: alpha.water

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  2.3.0                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       volScalarField;
13     object      alpha.water;
14 }
15 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17 dimensions      [0 0 0 0 0 0 0];
18
19 internalField   uniform 0;
20
21 boundaryField
22 {
23     bottom
24     {
25         type            zeroGradient;
26     }
27
28     outlet
29     {
30         type            zeroGradient;
31     }
32
33     walls
34     {
35         type            zeroGradient;
36     }
37
38     defaultFaces
39     {
40         type            empty;
41     }
42 }
43
44 // ************************************************************************* //
```

The pressure, `p_rgh`, is given as shown in Listing 16. The left, right and bottom walls of the column are given as `zeroGradient` and the top wall (`outlet`) has a fixed value of zero.

Listing 16: p_rgh

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.3.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p_rgh;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    bottom
    {
        type            zeroGradient;
    }

    outlet
    {
        type            fixedValue;
        value           uniform 0;
    }

    walls
    {
        type            zeroGradient;
    }

    defaultFaces
    {
        type            empty;
    }
}

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

The velocity is initialized as illustrated in Figure 2 and shown in Listing 17. It should be noted that the side walls are given a `slip` condition, to exclude boundary layer effects.

Listing 17: U

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.3.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

```

```
dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    bottom
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    outlet
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    walls
    {
        type            slip;
    }
    defaultFaces
    {
        type            empty;
    }
}


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

The boundary conditions of the LS function are shown in Listing 18, which is similar to those of `alpha.water`

Listing 18: psi

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.3.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      psi;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    walls
    {
        type            zeroGradient;
    }

    bottom
    {
      type            zeroGradient;
    }

    outlet
    {
```

```
35          type            zeroGradient;
36      }
37
38      defaultFaces
39      {
40          type            empty;
41      }
42  }
43
44  // ************************************************************************* //
```

5.3 Solver settings

The solvers settings (`constant/fvSchemes` and `constant/fvSolution`) are similar to those in `damBreak` tutorial case in `$FOAM_TUTORIAL/multiphase/interFoam/laminar/damBreak`. The counter-gradient term is switched off by setting `cAlpha` to zero in `system/fvSolution`. The air-bubble is initialized as given in Figure 2 using the `setFields` utility. To implement that a `setFieldsDict` is required in the `system/` folder. The `setFieldsDict` is shown in Listing 19.

Listing 19: setFieldsDict

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  2.3.0                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      location    "system";
14      object      setFieldsDict;
15  }
16  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18  defaultFieldValues
19  (
20      volScalarFieldValue alpha.water 1
21  );
22
23  regions
24  (
25      cylinderToCell
26      {
27          p1 (0.25 0.5 -1);
28          p2 (0.75 0.5 1);
29          radius 0.25;
30
31          fieldValues
32          (
33              volScalarFieldValue alpha.water 0
34          );
35      }
36  );
37
38  // ************************************************************************* //
```

The cell size, Δx needs to be mentioned in `constant/transportProperties` as requested in `createFields.H` in the solver. Also, the interface thickness, ϵ, needs to be given in `constant/transportProperties`

Listing 20: transportProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.3.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

phases (water air);

water
{
    transportModel  Newtonian;
    nu              nu [ 0 2 -1 0 0 0 0 ] 0.01;
    rho             rho [ 1 -3 0 0 0 0 0 ] 1000;
}

air
{
    transportModel  Newtonian;
    nu              nu [ 0 2 -1 0 0 0 0 ] 0.1;
    rho             rho [ 1 -3 0 0 0 0 0 ] 1;
}

sigma           sigma [ 1 0 -2 0 0 0 0 ] 1.96;

deltaX          deltaX [ 0 0 0 0 0 0 0 ] 0.00625; //0.006667;
epsilon     epsilon [ 0 0 0 0 0 0 0 ] 0.009375; //1.5*deltaX;

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

6 Running the case

The solver needs to be compiled to make it available in the OF lib solvers. Run `wmake` inside both solvers `sclsVOFFoam` and `sclsVOFFoamsf` as,

```
cd $WM_PROJECT_USER_DIR/applications/solvers/multiphase/sclsVOFFoamsf
wmake
cd ../sclsVOFFoam
wmake
```

Once they are complied, move to the run folder with `cd $FOAM_RUN` and make two copies of the case for running solvers, `interFoam, sclsVOFFoamsf` with

```
cp -r bubblecol bubblecolinter
cp -r bubblecol bubblecolsf
```

The current case of `bubblecol` will serve as case for `sclsVOFFoam`. Change the solver names in `./Allrun` for the respective cases and run the solvers by

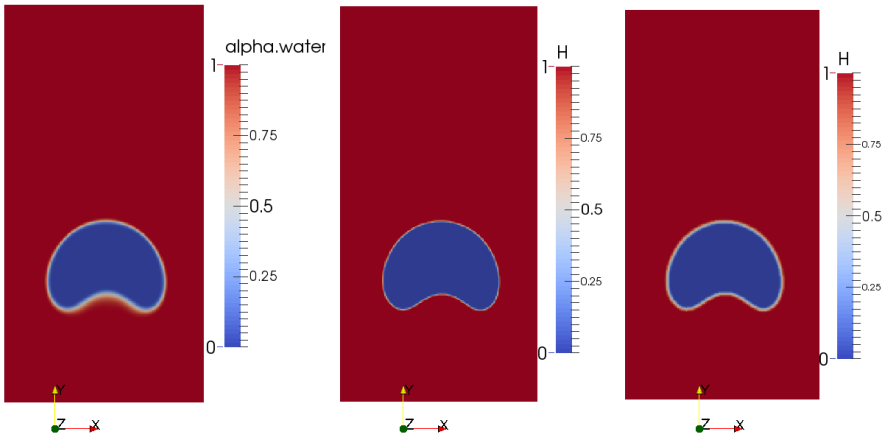
```
sed -i s/sclsVOFFoam/sclsVOFFoamsf/g bubblecolsf/Allrun
sed -i s/sclsVOFFoam/interFoam/g bubblecolinter/Allrun
./bubblecol/Allrun
./bubblecolinter/Allrun
./bubblecolsf/Allrun
```

7 Results

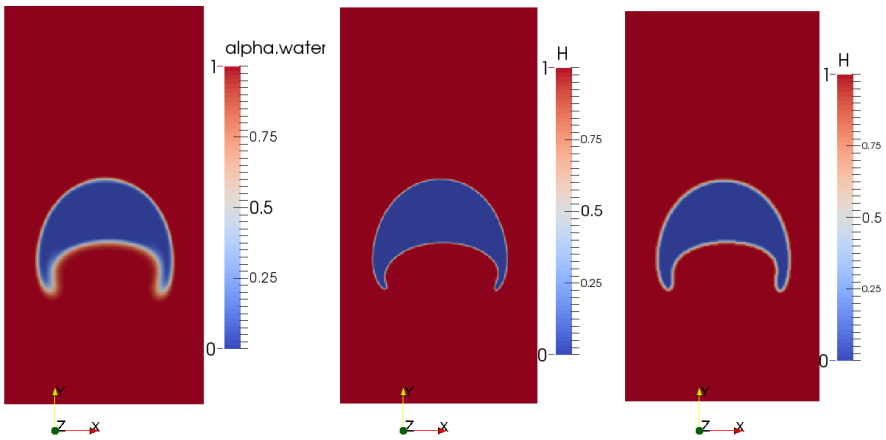
The bubble column is tested for all three solvers, `interFoam, sclsVOFFoam` and `sclsVOFFoamsf`. The bubble position at three different times are shown in Figure 3. Leftmost in the figure is results from the `interFoam` solver using the VOF advection equation. Middle column shows the results from the `sclsVOFFoam` solver and the rightmost column shows the results from the `sclsVOFFoamsf` solver. The α field represents the bubble for the `interFoam` solver, while the heaviside function, H, represents the bubble for the `sclsVOFFoam` and `sclsVOFFoamsf` solvers. For `interFoam` solver, VOF variable, α, smears the bubble. However, `sclsVOFFoam` solvers gives a sharp interface.

8 Future work

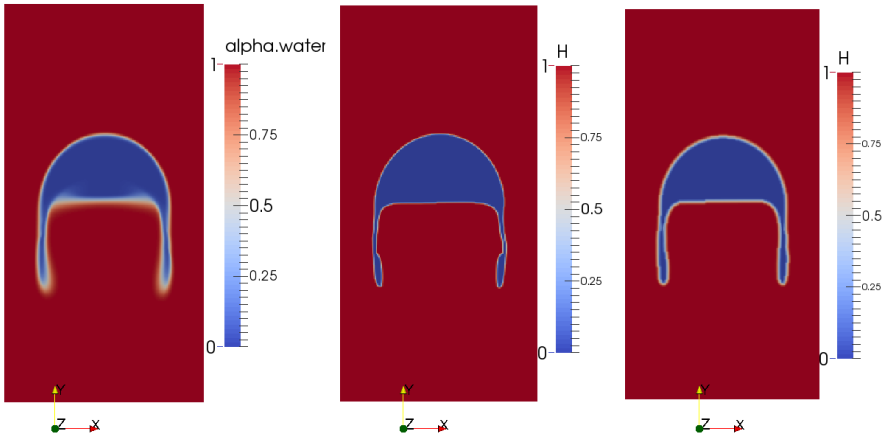
- The solvers are only able to handle the `zeroGradient` boundary condition, which needs to be expanded.
- The solvers are is quite sensitive to the variable `deltaX`, that is given by the user. This needs to be implicitly computed from the solver.



(a) Time = 1s



(b) Time = 2s



(c) Time = 3s

Figure 3: Bubble positions at different times t = 1, 2 and 3s. Staring from left, bubble captured with solvers interFoam, sclsVOFFoam and sclsVOFFoamsf.

References

[1] H. G. Weller, "A new approach to vof-based interface capturing methods for incompressible and compressible flows," Tech. Rep. TR/HGW/04, 2008.

[2] S. T. Zalesak, "Fully multidimensional flux-corrected transport algorithms for fluids," *Journal of Computational Physics*, vol. 31, no. 3, pp. 335 – 362, 1979.

[3] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations," *Journal of Computational Physics*, vol. 79, no. 1, pp. 12 – 49, 1988.

[4] M. Sussman, P. Smereka, and S. Osher, "A level set approach for computing solutions to incompressible two-phase flow," *Journal of Computational physics*, vol. 114, no. 1, pp. 146–159, 1994.

[5] M. Sussman and E. G. Puckett, "A coupled level set and volume-of-fluid method for computing 3d and axisymmetric incompressible two-phase flows," *Journal of Computational Physics*, vol. 162, no. 2, pp. 301 – 337, 2000.

[6] A. Albadawi, D. Donoghue, A. Robinson, D. Murray, and Y. Delauré, "Influence of surface tension implementation in volume of fluid and coupled volume of fluid with level set methods for bubble growth and detachment," *International Journal of Multiphase Flow*, vol. 53, no. 0, pp. 11 – 28, 2013.

[7] T. Yamamoto, "Setting and usage of openfoam multiphase solver (s-clsvof)," June 2014. Available at http://www.slideshare.net/takuyayamamoto1800/s-clsvofsolver-35845718.

[8] S. Hysing, S. Turek, D. Kuzmin, N. Parolini, E. Burman, S. Ganesan, and L. Tobiska, "Quantitative benchmark computations of two-dimensional bubble dynamics," *International Journal for Numerical Methods in Fluids*, vol. 60, no. 11, pp. 1259–1288, 2009.