Solid and Fluid Mechanics

CFD WITH OPENSOURCE SOFTWARE, ASSIGNMENT 3

Tutorial multiphaseInterFoam

FOR THE DAMBREAK4PHASE CASE

Author: Patrik Andersson

Introduction

Description: multiphaseInterFoam

"Incompressible multi-phase mixture with built in solution for the phase fractions with interface compression for interface-capturing. Derived from transportModel it can be used in conjunction with the incompressible turbulence models. Surface tension and contact-angle is handled for the interface between each phase-pair."

damBreak4phase

Similar to damBreak but now with four phases: Water, air, oil and mercury. All phases are intitially located behind a membrane which is removed at `t=0` and the fluids collapse upon each other and the obstacle. this creates a complicated mixture where the interaction between the phases need to be interpolated and the contact angles calculated.

Getting started

Copy the multiphaseInterFoam tutorial to the run directory (note that the placement of the case might differ between versions).

```
OF17x
cp -r  $FOAM_TUTORIALS/multiphase/multiphaseInterFoam/laminar/\
damBreak4phase $FOAM_RUN
cd $FOAM_RUN/damBreak4phase
```

The folder copied contains:

`/0`– *alphair alphamercury alphaoil alphas alphawater p_rgh U*

`/0.org`– *backup of original files listed above*

`/constant`– *g motionProperties transportProperties turbulenceProperties*

subfolder: `/polymesh`– *blockMeshDict boundary faces neighbour owner points*

`/system` – *controlDict decomposeParDict fvSchemes fvSolution setFieldsDict*

Initial conditions

In `/0`we find all the intial conditions for the different phases and a file called *phases*, which combines all of the phases so that the can be easier shown at the same time when post-processing in paraFoam. For example we can take a look at the reference phase air in *alphaair*.

```
leftWall
{
    type            alphaContactAngle;
    thetaProperties
    (
        ( water air ) 90 0 0 0
        ( oil air ) 90 0 0 0
        ( mercury air ) 90 0 0 0
        ( water oil ) 90 0 0 0
        ( water mercury ) 90 0 0 0
        ( oil mercury ) 90 0 0 0
    );
   value           uniform 0;
}
```

Set contact angle

$\hat{n}$ denotes the normal to the interface at the wall as:

$$\hat{n} = n_w \cos(\theta_{eq}) - n_t \sin(\theta_{eq}) \tag{1}$$

And θ_{eq} is the static contact angle set to 90 degrees.
n_w the unit normal vector to the wall pointing towards the wall.
n_t the unit vector tangential to the wall pointing towards the liquid.

By setting θ_{eq} = 90 the fluid will be we avoid using the surface tension force between the wall and the fluid. If θ_{eq} would be less than 90 degrees, then that would indicate that the fluid wets the wall.

Properties

In /constant:

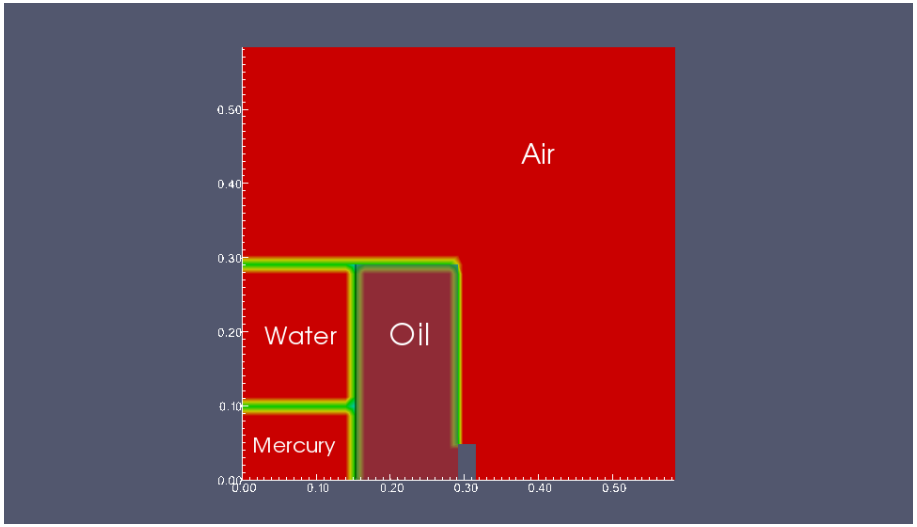
motionProperties: Motion of mesh set to staticFvMesh, since no movement.

transportProperties: the dynamic laminar viscosity, μ, the density ρ, reference phase, surface tension coefficient, σ.

turbulenceProperties: Turbulent model set to laminar.

Solver setup

In `/system` we find *setFieldsDict* we see the positioning of the phases.



In *fvSchemes* we set all schemes required to solve.

Moving on to *fvSolution* we have the solver setup with solver method specification.

multiphaseMixture.H

Go to `cd $FOAM_SOLVERS/multiphase/multiphaseInterFoam`

Now, lets move on to the solver. We begin in phases.C and .H located within the multiphaseMixture folder. Where one phase at the time is handled and ν and ρ is read.

From phases we go to multiphaseMixture.H. The code initially sets the selected transportmodel and then moves forward by defining interfacePairs for the mixture of phases. This can bee seen in the Constructor and the Friend Operators section of the code.

```
// Constructors
            interfacePair()
            {}

            interfacePair(const word& alpha1Name, const word& alpha2Name)
            :
                Pair<word>(alpha1Name, alpha2Name)
            {}

            interfacePair(const phase& alpha1, const phase& alpha2)
            :
                Pair<word>(alpha1.name(), alpha2.name())
            {}
```

```
    // Friend Operators

        friend bool operator==
        (
            const interfacePair& a,
            const interfacePair& b
        )
        {
            return
            (
                ((a.first() == b.first()) && (a.second() == b.second()))
             || ((a.first() == b.second()) && (a.second() == b.first()))
            );
        }
        friend bool operator!=
        (
            const interfacePair& a,
            const interfacePair& b
        )
        {
            return (!(a == b));
        }
};
```

Furthermore, the code defines terms such as those seen in table 1 on next slide.

Private data terms	
refPhase	The phase chosen as reference
rhoPhi	The volumetric flux
sigmatable	The stresses for the interface pair
deltaN	Stabilisation for the normalisation of the interface normal
alphaTable	Phase-fraction field table for multivariate discretization from multivariateSurfaceInterpolationScheme
Member functions	Returns the:
phases	phases
U	velocity
phi, rhophi	volumetric flux
rho	mixture density
mu	dynamic laminar viscosity
muf	face-interpolated dynamic laminar viscosity
nu	kinematic laminar viscosity
nuf	face-interpolated kinematic laminar viscosity
nearInterface	Indicator of the proximity of the interface-field, values are 1 near and 0 away from the interface
solve	Solve for the mixture phase-fractions
correct	Correct the mixture properties
read	Read base transportProperties dictionary

Table 1: multiphaseMixture.H

Now lets go to the .C file

multiphaseMixture.C

To view type:

```
gedit $FOAM_SOLVERS/multiphase/multiphaseInterFoam/\
multiphaseMixture/multiphaseMixture.C+
```

In the solve part of *multiphaseMixture.C* we have the iteration loops for ρ, μ, μ_f.There are also loops for ν, which is $\frac{\mu}{\rho}$, and one loop for the faceinterpolated ν_f. The loop for the surfacetension-force (refered to as stf in the code) is a bit more extensive, here σ for the interfacepair and an interpolation is performed between the two phases.

The surface tension force is defined as:

$$F_s = \sigma \left(\nabla \cdot \left(\frac{\nabla \alpha}{|\nabla \alpha|} \right) \right) (\nabla \alpha) \tag{2}$$

Where

$\nabla \alpha$ = n the vector normal to the interface
σ = surface tension coefficient

In the code this is represented by:

```
stf += dimensionedScalar("sigma", dimSigma_, sigma())
   *fvc::interpolate(K(alpha1, alpha2))*
    (
        fvc::interpolate(alpha2)*fvc::snGrad(alpha1)
```

```
        - fvc::interpolate(alpha1)*fvc::snGrad(alpha2)
    );
```

Moving on we have the piso-loops for alpha in the code (see table 2 below). Compared to the two-phase case for InterFoam we now have a different setup with, for example, four times the halfing of the time-step.

piso-loops	fvSolution settings	Description
nAlphaSubCycles	4	Number of subsycles for α_n for each timestep
nAlphaCorr	4	Number of correction for α, to improve quality of solution via fixed point iteration
cycleAlpha	yes	Cycling of alpha turned on
cAlpha	2	Compression of the interface, above one equals to enhanced compression

Table 2: Piso-loops in *multiphaseMixture.C*

Contact Angle

A large and interesting section in the solve part of *multiphaseMixture.C* is the correction for the boundary condition, on the unit normal nHat on walls, in order to produce a correct contact angle here. The dynamic contact angle is calculated by using the component of the velocity U on the direction of the interface, parallell to the wall. Before we go in to this we need to take a quick look at the definitions of the angles in *alphaContactAngleFvPatchScalarField.C*.

class interfaceThetaProps		
theta0	θ_0 (θ_C)	Equilibrium contact angle
uTheta	u_θ	Dynamic contact angle velocity scale
thetaA	θ_A	Limiting advancing contact angle
thetaR	θ_R	Limiting receeding contact angle

Table 3: Angles

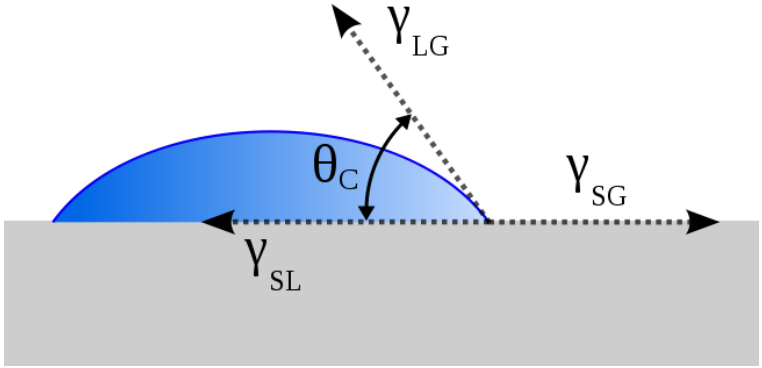


Figure 1: Contact angle and interface energies

In figure 1 above, we see the contact angle of a droplet which will represent our phase. γ_{SL} denotes the solid liquid energy, γ_{SG} the solid vapor energy and γ_{LG} the liquid vapor energy, i.e. the surface tension. This is governed by youngs equation (3) which will be satisfied at equilibrium.

$$0 = \gamma_{SG} - \gamma_{SL} - \gamma_{LG}\cos(\theta_C) \tag{3}$$

Where θ_C is dependent on the highest (advancing) contact angle θ_A and the lowest (receding) contact angle θ_R, written as:

$$\theta_C = \arccos\left(\frac{r_A\cos(\theta_A) + r_R\cos(\theta_R)}{r_A + r_R}\right) \tag{4}$$

where

$$r_A = \sqrt[3]{\frac{\sin^3(\theta_A)}{2 - 3\cos(\theta_A) + \cos(\theta_A)}} \tag{5}$$

$$r_R = \sqrt[3]{\frac{\sin^3(\theta_R)}{2 - 3\cos(\theta_R) + \cos(\theta_R)}} \tag{6}$$

θ_A is the contact angle when increasing the volume of, for example, the droplet. θ_R is the contact angle when decreasing the volume. In other words, when there is a relative motion of the droplet over a solid surface or another phase-interface, a different angle than the equilibrium contact angle will appear. It depends upon the direction of the previous motion, that is, if it was a advancing or receeding motion of the surface/interface (see figure 2 for explanation).

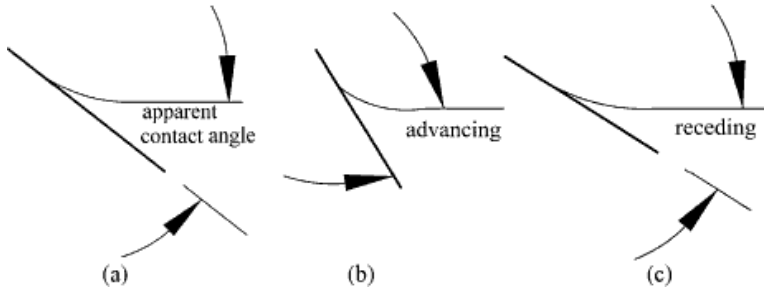


Figure 2: Schematic of equilibrium contact angle: θ (a) stationary liquid, (b) liquid flows upward, (c) liquid flows downward.

The data is grouped into the member function thetaProps for the interface-pairs.

Back to multiphaseMixture.C

Lets go back to multiphaseMixture.C and have a look at the correction of the contact angle Then finally, corrects the angle θ by

$$\theta = (\theta_A - \theta_R) * tanh\left(\frac{u_{wall}}{u_\theta}\right) \tag{7}$$

This new angle is then used to reset nHatPatch (the direction of the contact interface) so that it corresponds to the contact angle.

Finally, we are ready for the alpha-equation which will be used when calculating the new volumetric flux. The equation reads:

$$\frac{d\alpha}{dt} + mvconvection- > fvmdiv(\phi, \alpha) \tag{8}$$

Where fvmdiv is the divergence of the flux and the α-field. The flux calculated from the alpha-equation is then calculated and later used for rhophi_.

UEqn, PEqn and multiphaseInterFoam

The multiphaseInterFoam.C code is in it self relatively short. Since the major solving is done in previously discussed multiphaseMixture section.

the U-equation:

$$\left(\frac{d}{dt}(\rho, U) + \nabla \cdot (\rho\phi_{mix}, U) - \nabla^2(\mu_{Eff}, U) - \nabla U \cdot \nabla \mu_{Eff}\right)_{implicit} - (\nabla \cdot (\mu_{Eff}))_{explicit} \quad (9)$$

where:

$$\begin{aligned} \mu_{Eff} &= \mu_f + \text{interpolate}(\rho * \nu_t) \\ \mu_f &= mixture.muf() \\ \nu_t &= \text{turbulent viscosity} \end{aligned}$$

$$UEqn = (F_s - g * \nabla \rho_{explicit} - \nabla P_{rgh,explicit}) * cellfacevectors \quad (10)$$

where:

$$\begin{aligned} F_s &= \text{the surface-tension-force derived from the mixture} \\ \nabla &= \text{facenormal-gradient} \end{aligned}$$

Summarizing the solution steps

Onno Ubbink ”Numerical prediction of two fluid systems with sharp interfaces”.

The solution sequence is as follows:

1. Initialise all the variables.
2. Calculate the Courant number and adjust the time step if necessary.
3. Solve the equation by using the old time level’s volumetric fluxes.
4. Use the new values together with the constitutive relations to obtain an estimate for the new viscosity, density and the face densities.
5. Use the above values to do a momentum prediction and continue with the PISO algorithm.
6. If the final time has not yet been reached advance to the next time level and return to step 2.

Running the case

Finally time to run the case! Go to the case-folder in the run directory and execute blockMesh. When done, write setFields. setFields now sets the specified α-values for the different phases in their respective boxes as previously described.

Now it is time to initialize the solver, type: `multiphaseInterFoam | tee log`

Post-processing

Now we can view the results by typing paraFoam. The best way to visualize it is my simple choosing the alphas parameter to visualize the 4 different phases.

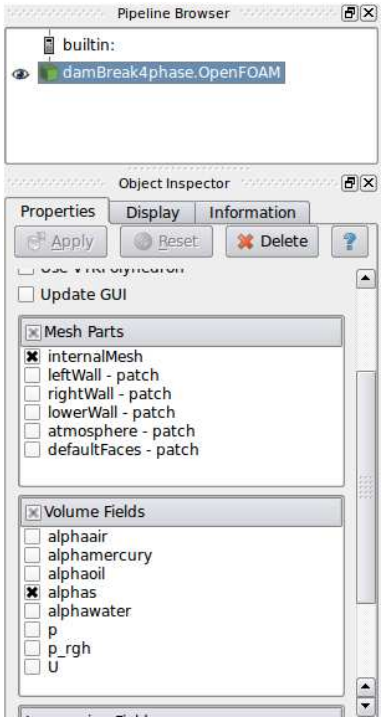


Figure 3: Setup in paraFoam