Lehrstuhl für THERMODYNAMIK

Garching, 29.01.2015

Interfacial Forces in Multi-Scale Two-Phase Flow

OpenFOAM User Group Meeting, Garching

Stefan Wenzel
Lehrstuhl für Thermodynamik, Technische Universität München

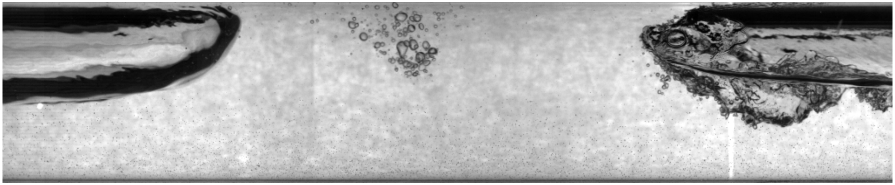
Supported by:

Federal Ministry for Economic Affairs and Energy

on the basis of a decision by the German Bundestag

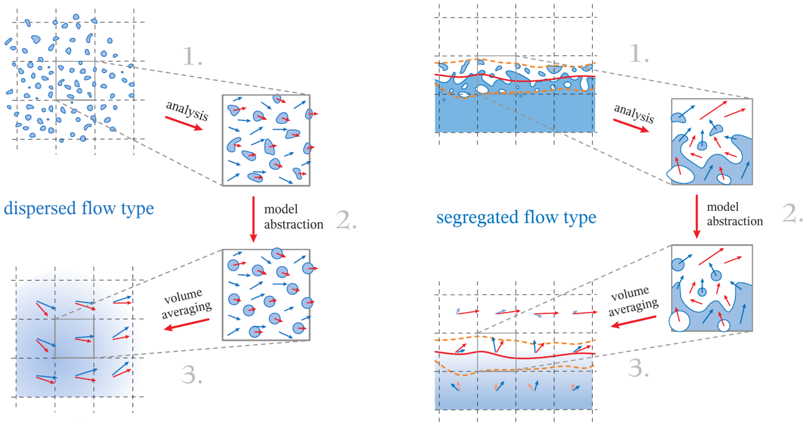
Introduction

- two-phase flows problems often appear to be multi-scale problems
- to describe the flow, accurate calculation of interfacial mass and momentum transfer is crucial
- this is inherent feature of front tracking methods (VoF, Level-Set)
- must be achieved by additional closure in context of Two-Fluid Methodologies (TFM)


slug flow as multi-scale two-phase problem

Partial Penetrating Continua

a TFM appraoch in multi-scale problems

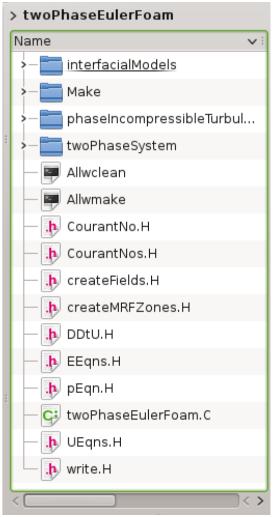


SOURCE: Marschall, H., 2011, *Towards the Numerical Simulation of Multi-Scale Two-Phase Flows*,PhD-Thesis, Technische Universität München, Germany

Interfacial Forces in Multi-Scale Two-Phase Flow

$\frac{\partial \alpha_k \rho_k \overline{\mathbf{U}}_k}{\partial t} + \nabla \bullet (\alpha_k \rho_k \overline{\mathbf{U}}_k \overline{\mathbf{U}}_k) - \nabla \bullet (\alpha_k \overline{\mathbf{R}}_k^{\text{eff}}) =$	
$- \alpha_k \nabla \overline{p} + \alpha_k \rho_k \mathrm{g}$	pressure and gravitation
$+ \Gamma_{\text{fs}} \cdot \sigma \kappa \nabla \alpha$	surface Tension
$+ \Gamma_{\text{fs}} \cdot \alpha_l \alpha_g K^{\text{s}} (\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l)$	drag - segregated flow
$+ (1 - \Gamma_{\text{fs}}) \cdot \alpha_l \alpha_g K^{\text{d}} (\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l)$	drag - dispersed flow
$+ (1 - \Gamma_{\text{fs}}) \cdot \alpha_g \rho_l C_L (\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l) \times (\nabla \times \overline{\mathbf{U}}_l)$	lateral Lift
$+ (1 - \Gamma_{\text{fs}}) \cdot \rho_l k_l C_{td} \nabla \alpha$	turbulent dirpersion
$+ (1 - \Gamma_{\text{fs}}) \cdot \alpha_g \rho_l \left(\frac{D\ \overline{\mathbf{U}}_g}{Dt} - \frac{D\ \overline{\mathbf{U}}_l}{Dt} \right)$	virtual mass
$+ (1 - \Gamma_{\text{fs}}) \cdot \rho_l \alpha_g C_W \lvert \overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l \rvert^2 \vec{n}_W$	wall lubrication

Calculation of Interfacial Forces in OpenFOAM

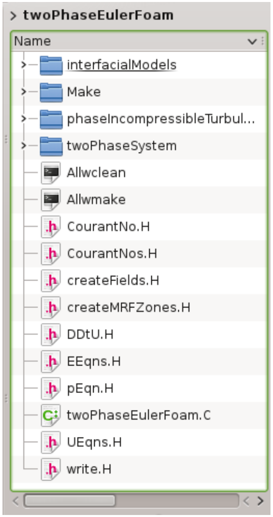


twoPhaseEulerFoam

Name

- interfacialModels
- Make
- phaseIncompressibleTurbul...
- twoPhaseSystem
- Allwclean
- Allwmake
- CourantNo.H
- CourantNos.H
- createFields.H
- createMRFZones.H
- DDtU.H
- EEqns.H
- pEqn.H
- twoPhaseEulerFoam.C
- UEqns.H
- write.H

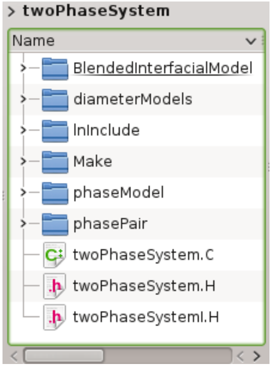
Calculation of Interfacial Forces in OpenFOAM



twoPhaseEulerFoam

Name

- interfacialModels
- Make
- phaseIncompressibleTurbul...
- twoPhaseSystem
- Allwclean
- Allwmake
- CourantNo.H
- CourantNos.H
- createFields.H
- createMRFZones.H
- DDtU.H
- EEqns.H
- pEqn.H
- twoPhaseEulerFoam.C
- UEqns.H
- write.H



twoPhaseSystem

Name

- BlendedInterfacialModel
- diameterModels
- lnInclude
- Make
- phaseModel
- phasePair
- twoPhaseSystem.C
- twoPhaseSystem.H
- twoPhaseSystemI.H

?

Solution Procedure

executable `twoPhaseEulerFoam` calls `twoPhaseEulerFoam.C`:

```
Info<< "\ nStarting time loop \ n" << endl;
while (runTime.run())
{
    runTime++;
    Info<< "Time=" << runTime.timeName() << nl << endl;
     // --- Pressure-velocity PIMPLE corrector loop
    while (pimple.loop())
    {
        fluid.solve();
        rho = fluid.rho();
        fluid.correct();

        #include "EEqns.H"
        #include "UEqns.H"

        // --- Pressure corrector loop
        while (pimple.correct())
        {
            #include "pEqn.H"
        }
        if (pimple.turbCorr())
        {
            fluid.correctTurbulence();
        }
    }
    #include "write.H"
}
Info<< "End\ n" << endl;
return 0;
```

Solution Procedure

executable `twoPhaseEulerFoam` calls `twoPhaseEulerFoam.C`:

```
Info<< "\ nStarting time loop \ n" << endl;
while (runTime.run())
{
    runTime++;
    Info<< "Time=" << runTime.timeName() << nl << endl;
     // --- Pressure-velocity PIMPLE corrector loop
    while (pimple.loop())
    {
        fluid.solve();
        rho = fluid.rho();
        fluid.correct();

        #include "EEqns.H"
        #include "UEqns.H"

        // --- Pressure corrector loop
        while (pimple.correct())
        {
            #include "pEqn.H"
        }
        if (pimple.turbCorr())
        {
            fluid.correctTurbulence();
        }
    }
    #include "write.H"
}
Info<< "End\ n" << endl;
return 0;
```

Solution Procedure

the "main" function `solve()` takes place in `twoPhaseSystem.C`:

```
void Foam::coupledTwoPhaseSystem::solve(){
// 1. DEFINE FLUXES:

    surfaceScalarField phic("phic", phi_);
    surfaceScalarField phir("phir", phi1 - phi2);

// 2. ADDED INTERFACE COMPRESSION TO FLUX:

    volScalarField GammaFs = findInterface();
    phir += min(GammaFsf*mag(phic/mesh_.magSf()), max(phic/mesh_.magSf()))*-nHatf_;

// 3. CALCULATE FLUXES:

            surfaceScalarField alphaPhic1
            (
                fvc::flux
                (
                    phic,
                    alpha1,
                    alphaScheme
                )
              + fvc::flux
                (
                   -fvc::flux(-phir, scalar(1) - alpha1, alpharScheme),
                    alpha1,
                    alpharScheme
                )
             );
...
...

// 4. LIMIT PHASE FRACTIONS:

            MULES::explicitSolve

// 5. CALCULATE INTERFACE CURVATURE:

            calculateCurv();
}
```

The interfacial forces are also calculated within `twoPhaseSystem.C`:

```
Foam::tmp<Foam::volScalarField> Foam::coupledTwoPhaseSystem::dragCoeff() const {...}

Foam::tmp<Foam::volScalarField> Foam::coupledTwoPhaseSystem::virtualMassCoeff() const {...}

Foam::tmp<Foam::volVectorField> Foam::coupledTwoPhaseSystem::liftForce() const {...}

Foam::coupledTwoPhaseSystem::wallLubricationForce() const {...}

Foam::coupledTwoPhaseSystem::turbulentDispersionForce() const {...}

Foam::tmp<Foam::surfaceScalarField> Foam::coupledTwoPhaseSystem::surfaceTension(){...}
```

Interfacial Forces in Multi-Scale Two-Phase Flow

$$\frac{\partial \alpha_k \rho_k \overline{\mathbf{U}}_k}{\partial t} + \nabla \bullet (\alpha_k \rho_k \overline{\mathbf{U}}_k \overline{\mathbf{U}}_k) - \nabla \bullet (\alpha_k \overline{\mathbf{R}}_k^{\mathrm{eff}}) =$$

$-\alpha_k \nabla \overline{p} + \alpha_k \rho_k \mathrm{g}$	pressure and gravitation
$+\Gamma_{\mathrm{fs}} \cdot \sigma \kappa \nabla \alpha$	surface Tension
$+\Gamma_{\mathrm{fs}} \cdot \alpha_l \alpha_g K^{\mathrm{s}} (\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l)$	drag - segregated flow
$+(1-\Gamma_{\mathrm{fs}}) \cdot \alpha_l \alpha_g K^{\mathrm{d}} (\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l)$	drag - dispersed flow
$+(1-\Gamma_{\mathrm{fs}}) \cdot \alpha_g \rho_l C_L (\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l) \times (\nabla \times \overline{\mathbf{U}}_l)$	lateral Lift
$+(1-\Gamma_{\mathrm{fs}}) \cdot \rho_l k_l C_{td} \nabla \alpha$	turbulent dirpersion
$+(1-\Gamma_{\mathrm{fs}}) \cdot \alpha_g \rho_l \left(\frac{D\ \overline{\mathbf{U}}_g}{Dt} - \frac{D\ \overline{\mathbf{U}}_l}{Dt} \right)$	virtual mass
$+(1-\Gamma_{\mathrm{fs}}) \cdot \rho_l \alpha_g C_W \lvert \overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l \rvert^2 \vec{n}_W$	wall lubrication

Interfacial Forces in Multi-Scale Two-Phase Flow

$$\frac{\partial \alpha_k \rho_k \overline{\mathbf{U}}_k}{\partial t} + \nabla \bullet (\alpha_k \rho_k \overline{\mathbf{U}}_k \overline{\mathbf{U}}_k) - \nabla \bullet (\alpha_k \overline{\mathbf{R}}_k^{\mathrm{eff}}) =$$

$-\alpha_k \nabla \overline{p} + \alpha_k \rho_k \mathrm{g}$	pressure and gravitation
$+\Gamma_{\mathrm{fs}} \cdot \sigma \kappa \nabla \alpha$	surface Tension
$+\Gamma_{\mathrm{fs}} \cdot \alpha_l \alpha_g K^{\mathrm{s}} (\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l)$	drag - segregated flow
$+(1-\Gamma_{\mathrm{fs}}) \cdot \alpha_l \alpha_g K^{\mathrm{d}} (\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l)$	drag - dispersed flow
$+(1-\Gamma_{\mathrm{fs}}) \cdot \alpha_g \rho_l C_L (\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l) \times (\nabla \times \overline{\mathbf{U}}_l)$	lateral Lift
$+(1-\Gamma_{\mathrm{fs}}) \cdot \rho_l k_l C_{td} \nabla \alpha$	turbulent dirpersion
$+(1-\Gamma_{\mathrm{fs}}) \cdot \alpha_g \rho_l \left(\frac{D\ \overline{\mathbf{U}}_g}{Dt} - \frac{D\ \overline{\mathbf{U}}_l}{Dt} \right)$	virtual mass
$+(1-\Gamma_{\mathrm{fs}}) \cdot \rho_l \alpha_g C_W \lvert \overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l \rvert^2 \vec{n}_W$	wall lubrication

Surface Tension

continuum surface force: surfaceVector → volumeVector

$$\int_S \sigma\kappa\delta(x - x')dS \approx \sigma\kappa\nabla\alpha$$

The momentum equation in `/twoPhaseEulerFoam/UEqns.H`

```
U1Eqn =
 (
   fvm::ddt(alpha1, U1)
 + fvm::div(alphaPhi1, U1)
 + phase1.turbulence().divDevReff(U1)
  ==
   fvc::interpolate(fluid.surfaceTension())
   ...
 )
```

Surface Tension

continuum surface force: surfaceVector → volumeVector

$$\int_S \sigma\kappa\delta(x-x')dS \approx \sigma\kappa\nabla\alpha$$

The momentum equation in /twoPhaseEulerFoam/UEqns.H

```
U1Eqn =
 (
   fvm::ddt(alpha1, U1)
 + fvm::div(alphaPhi1, U1)
 + phase1.turbulence().divDevReff(U1)
  ==
   fvc::interpolate(fluid.surfaceTension())
   ...
 )
```

calls the calculation in twoPhaseSystem.C:

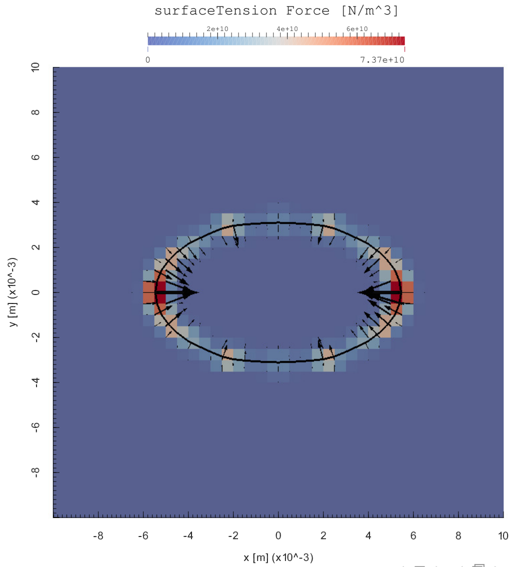
```
Foam::tmp<Foam::surfaceScalarField> Foam::coupledTwoPhaseSystem::surfaceTension()
{
      *fvc::interpolate(GammaFs)
      *sigma()
      *fvc::interpolate(Curv_)
      *(
           fvc::interpolate(phase2_)*fvc::snGrad(phase1_)
         - fvc::interpolate(phase1_)*fvc::snGrad(phase2_)
       )
}
```

Used Geometric Functions

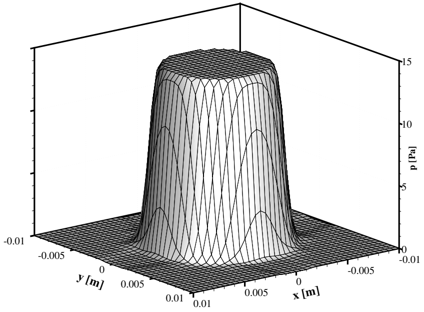
`fvc::interpolate()`	coverts volumeField<Type> to surfaceField<Type> by central differenzing
`fvc::reconstruct()`	coverts surfaceField<Type> to volumeField<Type>
`fvc::snGrad()`	surface normal gradient $\rightarrow$ returns surfaceField<Type>

$$(\nabla\phi)_f = \frac{\phi_N - \phi_P}{|\vec{\mathrm{d}}|}, \qquad \vec{\mathrm{d}} = N \rightarrow P$$

Calculation of Surface Tension



Pressure due to Surface Tension



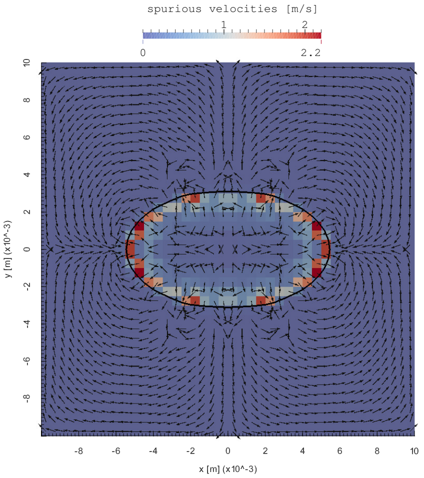
pressure jump over bubble interface

- Laplace's law for infinite cylinder:

$$p_{bub} = \frac{\sigma}{r_{bub}} \;,$$

  gives analytical solution of ? $p_{bub} = 14$ Pa.
- graphic shows the solution for 40x40 grid
- pressure smeared over several cell

Spurious Velocities due to Surface Tension



- inaccuracy in curvature calculations leads to imbalance in forces
- in consequence, spurious velocities appear and disturb the interface
- to overcome this problem:
  - geometrical reconstruction of the interface
  - mesh adaptive methods[1]

?

1 Ganesan, S., 2008, *On spurious velocities in problems with incompressible interfaces flow*, Otto-von-Guericke-Universität Magdeburg,Germany

Lehrstuhl für THERMODYNAMIK

Interfacial Forces in Multi-Scale Two-Phase Flow

$$
\begin{aligned}
\frac{\partial \alpha_k \rho_k \overline{\mathbf{U}}_k}{\partial t} + \nabla \bullet (\alpha_k \rho_k \overline{\mathbf{U}}_k \overline{\mathbf{U}}_k) - \nabla \bullet (\alpha_k \overline{\mathbf{R}}_k^{\mathrm{eff}}) = & \\
- \alpha_k \nabla \overline{p} + \alpha_k \rho_k \mathrm{g} & \qquad \text{pressure and gravitation} \\
+ \Gamma_{\mathrm{fs}} \cdot \sigma \kappa \nabla \alpha & \qquad \text{surface Tension} \\
+ \Gamma_{\mathrm{fs}} \cdot \alpha_l \alpha_g K^{\mathrm{s}} (\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l) & \qquad \text{drag - segregated flow} \\
+ (1 - \Gamma_{\mathrm{fs}}) \cdot \alpha_l \alpha_g K^{\mathrm{d}} (\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l) & \qquad \text{drag - dispersed flow} \\
+ (1 - \Gamma_{\mathrm{fs}}) \cdot \alpha_g \rho_l C_L (\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l) \times (\nabla \times \overline{\mathbf{U}}_l) & \qquad \text{lateral Lift} \\
+ (1 - \Gamma_{\mathrm{fs}}) \cdot \rho_l k_l C_{td} \nabla \alpha & \qquad \text{turbulent dirpersion} \\
+ (1 - \Gamma_{\mathrm{fs}}) \cdot \alpha_g \rho_l \left(\frac{D\ \overline{\mathbf{U}}_g}{Dt} - \frac{D\ \overline{\mathbf{U}}_l}{Dt} \right) & \qquad \text{virtual mass} \\
+ (1 - \Gamma_{\mathrm{fs}}) \cdot \rho_l \alpha_g C_W |\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l|^2 \vec{n}_W & \qquad \text{wall lubrication}
\end{aligned}
$$

Interfacial Forces in Multi-Scale Two-Phase Flow

$$\frac{\partial \alpha_k \rho_k \overline{\mathbf{U}}_k}{\partial t} + \nabla \bullet (\alpha_k \rho_k \overline{\mathbf{U}}_k \overline{\mathbf{U}}_k) - \nabla \bullet (\alpha_k \overline{\mathbf{R}}_k^{\text{eff}}) =$$

$- \alpha_k \nabla \overline{p} + \alpha_k \rho_k g$	pressure and gravitation
$+ \Gamma_{\text{fs}} \cdot \sigma \kappa \nabla \alpha$	surface Tension
$+ \Gamma_{\text{fs}} \cdot \alpha_l \alpha_g K^{\text{s}} (\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l)$	drag - segregated flow
$+ (1 - \Gamma_{\text{fs}}) \cdot \alpha_l \alpha_g K^{\text{d}} (\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l)$	drag - dispersed flow
$+ (1 - \Gamma_{\text{fs}}) \cdot \alpha_g \rho_l C_L (\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l) \times (\nabla \times \overline{\mathbf{U}}_l)$	lateral Lift
$+ (1 - \Gamma_{\text{fs}}) \cdot \rho_l k_l C_{td} \nabla \alpha$	turbulent dirpersion
$+ (1 - \Gamma_{\text{fs}}) \cdot \alpha_g \rho_l \left(\frac{D\ \overline{\mathbf{U}}_g}{Dt} - \frac{D\ \overline{\mathbf{U}}_l}{Dt} \right)$	virtual mass
$+ (1 - \Gamma_{\text{fs}}) \cdot \rho_l \alpha_g C_W \lvert \overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l \rvert^2 \vec{n}_W$	wall lubrication

Drag Calculation

- also referred to as *interface momentum transfer term*
- commonly based on the assumption of spherical particles, drag force for dispersed flow $\vec{F}^{\mathrm{d}}_{D,k}$ reads (OF standart):
- in OpenFOAM is extracted from the equation
- several common models to calculate drag coefficient C_D are available in the official release

Lehrstuhl für THERMODYNAMIK

Drag Calculation

- also referred to as *interface momentum transfer term*
- commonly based on the assumption of spherical particles, drag force for dispersed flow $\vec{F}^{\mathrm{d}}_{D,k}$ reads (OF standart):

$$\vec{F}^{\mathrm{d}}_{D,k} = \frac{3}{4}\alpha(1-\alpha)\rho_l \frac{C_D}{d_P}|\mathbf{U}_g - \mathbf{U}_l|(\mathbf{U}_g - \mathbf{U}_l)$$

- in OpenFOAM is extracted from the equation
- several common models to calculate drag coefficient C_D are available in the official release

Drag Calculation

- also referred to as *interface momentum transfer term*
- commonly based on the assumption of spherical particles, drag force for dispersed flow $\vec{F}^{\mathrm{d}}_{D,k}$ reads (OF standart):

$$\vec{F}^{\mathrm{d}}_{D,k} = \frac{3}{4}\alpha(1-\alpha)\rho_l \frac{C_D}{d_P}|\mathbf{U}_g - \mathbf{U}_l|(\mathbf{U}_g - \mathbf{U}_l)$$

- in OpenFOAM $K = \frac{3}{4}\rho_l \frac{C_D}{d_P}|\mathbf{U}_g - \mathbf{U}_l|$ is extracted from the equation
- several common models to calculate drag coefficient C_D are available in the official release

Drag Calculation

The momentum equation in `/twoPhaseEulerFoam/UEqns.H`

```
U1Eqn =
 (
   fvm::ddt(alpha1, U1)
 + fvm::div(alphaPhi1, U1)
 + phase1.turbulence().divDevReff(U1)
   ==
     fvc::interpolate(fluid.surfaceTension())
   - fvm::Sp(fluid.dragCoeff()/rho1, U1)
   ...
 )
```

Drag Calculation

The momentum equation in /twoPhaseEulerFoam/UEqns.H

```
U1Eqn =
 (
   fvm::ddt(alpha1, U1)
 + fvm::div(alphaPhi1, U1)
 + phase1.turbulence().divDevReff(U1)
   ==
     fvc::interpolate(fluid.surfaceTension())
   - fvm::Sp(fluid.dragCoeff()/rho1, U1)
   ...
 )
```

Drag Calculation

The momentum equation in /twoPhaseEulerFoam/UEqns.H

```
 U1Eqn =
  (
    fvm::ddt(alpha1, U1)
  + fvm::div(alphaPhi1, U1)
  + phase1.turbulence().divDevReff(U1)
    ==
      fvc::interpolate(fluid.surfaceTension())
    - fvm::Sp(fluid.dragCoeff()/rho1, U1)
    ...
  )
```

calls the drag calculation in /twoPhaseEulerFoam/twoPhaseSystem/twoPhaseSystem.C

```
Foam::tmp<Foam::volScalarField> Foam::coupledTwoPhaseSystem::dragCoeff() const
{
        phasePair::dictTable dragTable(lookup("drag"));

        const dragModel& dm = dragTable;

        volScalarField* Kptr_ =
        (
            dm.phase1()*dm.phase2()
           *dm.K(mag(dm.phase1().U() - dm.phase2().U()))
        ).ptr();

        dragCoeff->Kptr_;

    return dragCoeff;
}
```

?

Drag Calculation

The momentum equation in /twoPhaseEulerFoam/UEqns.H

```
U1Eqn =
 (
   fvm::ddt(alpha1, U1)
 + fvm::div(alphaPhi1, U1)
 + phase1.turbulence().divDevReff(U1)
   ==
     fvc::interpolate(fluid.surfaceTension())
   - fvm::Sp(fluid.dragCoeff()/rho1, U1)
   ...
 )
```

calls the drag calculation in /twoPhaseEulerFoam/twoPhaseSystem/twoPhaseSystem.C

```
Foam::tmp<Foam::volScalarField> Foam::coupledTwoPhaseSystem::dragCoeff() const
{
        phasePair::dictTable dragTable(lookup("drag"));

        const dragModel& dm = dragTable;

        volScalarField* Kptr_ =
        (
            dm.phase1()*dm.phase2()
           *dm.K(mag(dm.phase1().U() - dm.phase2().U()))
        ).ptr();

        dragCoeff->Kptr_;

    return dragCoeff;
}
```

?

Drag Calculation

The momentum equation in `/twoPhaseEulerFoam/UEqns.H`

```
U1Eqn =
 (
   fvm::ddt(alpha1, U1)
 + fvm::div(alphaPhi1, U1)
 + phase1.turbulence().divDevReff(U1)
   ==
     fvc::interpolate(fluid.surfaceTension())
   - fvm::Sp(fluid.dragCoeff()/rho1, U1)
   ...
 )
```

calls the drag calculation in
`/twoPhaseEulerFoam/twoPhaseSystem/twoPhaseSystem.C`

```
Foam::tmp<Foam::volScalarField> Foam::coupledTwoPhaseSystem::dragCoeff() const
{
        phasePair::dictTable dragTable(lookup("drag"));

        const dragModel& dm = dragTable;

        volScalarField* Kptr_ =
        (
            dm.phase1()*dm.phase2()
          *dm.K(mag(dm.phase1().U() - dm.phase2().U()))
        ).ptr();

        dragCoeff->Kptr_;

    return dragCoeff;
}
```

$\Rightarrow \alpha_l \alpha_g K(|\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l|)$?

Drag Calculation

The momentum equation in /twoPhaseEulerFoam/UEqns.H

```
U1Eqn =
 (
   fvm::ddt(alpha1, U1)
 + fvm::div(alphaPhi1, U1)
 + phase1.turbulence().divDevReff(U1)
   ==
     fvc::interpolate(fluid.surfaceTension())
   - fvm::Sp(fluid.dragCoeff()/rho1, U1)
   ...
 )
```

calls the drag calculation in /twoPhaseEulerFoam/twoPhaseSystem/twoPhaseSystem.C

```
Foam::tmp<Foam::volScalarField> Foam::coupledTwoPhaseSystem::dragCoeff() const
{
        phasePair::dictTable dragTable(lookup("drag"));

        const dragModel& dm = dragTable;

        volScalarField* Kptr_ =
        (
            dm.phase1()*dm.phase2()
           *dm.K(mag(dm.phase1().U() - dm.phase2().U()))
        ).ptr();

        dragCoeff->Kptr_;

    return dragCoeff;
}
```

$\Rightarrow \alpha_l \alpha_g K(|\overline{\mathbf{U}}_g - \overline{\mathbf{U}}_l|)$

?

modells can be added modular!

Summary

- several common sub-models in two-phase flow are available in the official release
- extending them or add new models can be achieved by acceptable effort
- modular composition of OpenFOAM makes it possible to modify submodels without changing the whole code
- physical weaknesses are sometimes accepted in favor of stability and numerical effort (e.g. interface compression)
- OpenFOAM shows up to be a *TOOLBOX* and should be used as that

Thanks for your attantion!