CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Project work:

Descriptions and comparisons of sprayFoam, reactingParcelFoam, and basicSprayCloud, basicReactingCloud

Developed for OpenFOAM-2.1.x

Author:
Salman Arshad

Peer reviewed by:
Olivier Petit
Ali Al Sam

February 6, 2014

Chapter 1

1.1 Introduction

This tutorial provides description and comparison of the two compressible flow solvers i.e. `reactingParcelFoam` and `sprayFoam`. It covers description and comparison of the clouds `basicReactingCloud` and `basicSprayCloud`. In the end description is provided on how to use and modify properties of both the clouds in case setup.

1.2 Description and comparison of reactingParcelFoam and sprayFoam

This section covers the description and comparsion of solvers `reactingParcelFoam` and `sprayFoam`.

1.2.1 Comparsion of solvers

To see the difference of files included in both solvers, go to `reactingParcelFoam` solver directory.

```
OF21x
sol
cd lagrangian/reactingParcelFoam
ls
```

This directory contains files `createClouds.H`, `createFields.H, hsEqn.H, pEqn.H`, `reactingParcelFoam.C`, `reactingParcelFoam.dep, rhoEqn.H, UEqn.H and YEqn.H`.

Now go to `sprayFoam` solver directory to see the files.

```
sol
cd lagrangian/sprayFoam
ls
```

This directory contains files `createClouds.H`, `sprayFoam.C and sprayFoam.dep` only. The `sprayFoam` directory does not have any of the files `createFields.H, hsEqn.H, pEqn.H, rhoEqn.H`, `UEqn.H` and `YEqn.H`. In order to locate these missing files in `sprayFoam` solver directory, go to directory `Make` and open file `options`.

```
vi Make/options
```

The first two lines of the file are:

```
EXE_INC = \
    -I$(FOAM_SOLVERS)/lagrangian/reactingParcelFoam \
```

which shows the compiler that these missing files are actually included from the `reactingParcelFoam` directory. The `sprayFoam` solver thus only differs from `ReactingParcelFoam` solver in the files `createClouds.H` and obviously the top level solver file `sprayFoam.C`.

To investigate the differences on top level solver, go to `reactingParcelFoam` solver directory and open file `reactingParcelFoam.C`.

```
sol
cd lagrangian/reactingParcelFoam
vi reactingParcelFoam.C
```

Also go to `sprayFoam` solver directory and open file `sprayFoam.C`.

```
sol
cd lagrangian/sprayFoam
vi sprayFoam.C
```

By opening the both top level solver files i.e. `reactingParcelFoam.C` and `sprayFoam.C` it is seen that both are compressible flow solvers using pimple loop. It can be seen that the only major difference is that they include two different types of clouds. The `reactingParcelFoam.C` includes

```
#include "basicReactingCloud.H"
```

and the `sprayFoam.C` includes

```
#include "basicSprayCloud.H"
```

This can be further investigated by opening the `createClouds.H` file. Go to directory of `reactingParcelFoam` solver and open the file `createClouds.H`.

```
sol
cd lagrangian/reactingParcelFoam/
vi createClouds.H
```

This file reads as:

```
Info<< "\nConstructing reacting cloud" << endl;
basicReactingCloud parcels
(
    "reactingCloud1",
    rho,
    U,
    g,
    slgThermo
);
```

Now go to `sprayFoam` solver directory and open the file `createClouds.H`.

```
sol
cd lagrangian/sprayFoam/
vi createClouds.H
```

This file reads as:

```
Info<< "\nConstructing reacting cloud" << endl;
basicSprayCloud parcels
(
    "sprayCloud",
    rho,
    U,
    g,
    slgThermo
);
```

It can easily be concluded after reading the `createClouds.H` files that that the only major difference between the two solvers is that one solver (`reactingParcelFoam`) uses the cloud class `basicReactingCloud` and other (`sprayFoam`) uses the cloud class `basicSprayCloud`. Also it can be seen that both types of clouds i.e. `basicReactingCloud` and `basicSprayCloud` are initialized using similar constructors in the files `createClouds.H`.

1.2.2 Description of solvers

As both of the solvers `reactingParcelFoam` and `sprayFoam` are similar so a short description of any solver is enough. In the `reactingParcelFoam` solver, reacting parcels are evolved first in the time loop and then density equation is solved. In the pressure velocity PIMPLE corrector loop, the momentum, species and enthaply equations are solved followed by the pressure corrector loop.

1.3 Description and comparison of basicReactingCloud and basicSprayCloud

This section will describe and compare the cloud classes `basicReactingCloud` and `basicSprayCloud`.

1.3.1 Overview of classes

Go to `src/lagrangian` directory and see all the directories inside it.

```
src
cd lagrangian
ls
```

Three directories are relevant here i.e. `basic`, `intermediate` and `spray`. Go to `basic` directory

```
cd basic
ls
```

The `basic` directory contains two important directories i.e. `particle` and `Cloud`. The `particle` directory contains the base particle class and the `Cloud` directory contains the base cloud class templated on particle type. Now go to `intermediate` directory

```
src
cd lagrangian/intermediate
ls
```

The `intermediate` directory contains many directories out of which three directories are important i.e `clouds`, `parcels` and `submodels`. The directory `parcels` contains different types of particles in which different submodels (available in the `submodels` directory) are added. To see different types of particles, list all the directories inside the directory `parcel`

```
tree -d parcels
```

Two important directories are `Templates` and `derived`. The directory `Templates` has different templated particle types while the directory `derived` contains combination of different particles to acheive added funtionalities (submodels). The submodels asscoiated to each particle type can be seen by listing all the directries in the `submodels`.

```
tree -d submodels
```

Go to the `Clouds` directory inside the `intermediate` directory

```
src
cd lagrangian/intermediate/clouds
ls
```

Three directories can be seen i.e. `baseClasses`, `derived` and `Templates`. The directory baseClasses contain virtual base classes for different templated clouds (avaialable in `Templates` directory). The `derived` directory contains combination of different clouds. The class `basicReactingCloud` is actually located in this directory `derived`. Now go to directory `spray`.

```
src
cd lagrangian/spray
ls
```

Just like `intermediate` directory, this directory contains three important directories i.e. `clouds`, `parcels` and `submodels`. These directories are similar to directories in `intermediate` except that these directories now contain spray clouds, spray parcels and submodels associated with them as shown by the following `tree -d` command.

```
src
cd lagrangian/spray
tree -d
```

The class `basicSprayCloud` is located in dircetory `src/lagrangian/spray/clouds/derived/basicSprayCloud`.

1.3.2 Description of classes

This section will provide description of classes `basicReactingCloud` and `basicSprayCloud`.

basicReactingCloud

Go to directory of class `basicReactingCloud` and open the file `basicReactingCloud.H` to see its definition.

```
src
cd lagrangian/intermediate/clouds/derived/basicReactingCloud
vi basicReactingCloud.H
```

In the file `basicReactingCloud.H`, the definition of `basicReactingCloud` is:

```
typedef ReactingCloud
<
    ThermoCloud
    <
        KinematicCloud
        <
            Cloud
            <
                basicReactingParcel
            >
        >
    >
> basicReactingCloud;
```

In the above definition, it can be seen that `basicReactingCloud` is a short name (`typedef`) for different layers of clouds on top of each other. `ReactingCloud` is layered on `ThermoCloud` which is layered on `KinematicCloud` and all these clouds are layered on the base cloud class `Cloud` (templated on particle type). In every cloud layer, new funtionalities (models) are added to the base cloud layer. The templated `Cloud` class is instantiated with class `basicReactingParcel` as type pararmeter. Now it is important to see the definition of `basicReactingParcel`. To see the definition of `basicReactingParcel` class go to its directory and open file `basicReactingParcel.H`.

```
src
cd lagrangian/intermediate/parcels/derived/basicReactingParcel
vi basicReactingParcel.H
```

The definition of `basicReactingParcel` in the file `basicReactingParcel.H` is given as:

```
typedef ReactingParcel
<
    ThermoParcel
    <
        KinematicParcel
        <
            particle
        >
    >
> basicReactingParcel;
```

This definition shows that `basicReactingParcel` is a short name (`typedef`) for different layers of parcels on top of each other just like `basicReactingCloud` was a short name for different layers of clouds on top of each other. `ReactingParcel` is layered on `ThermoParcel` which is layered on `KinematicParcel` and all these parcel types are layered on the base particle class `particle`.

basicSprayCloud

Go to directory of `basicSprayCloud` class and see it definition in the file `basicSprayCloud.H`.

```
src
cd lagrangian/spray/clouds/derived/basicSprayCloud
vi basicSprayCloud.H
```

The definition in the file `basicSprayCloud.H` is:

```
typedef SprayCloud
<
    ReactingCloud
    <
        ThermoCloud
        <
            KinematicCloud
            <
                Cloud
                <
                    basicSprayParcel
                >
            >
        >
    >
> basicSprayCloud;
```

According to the definition, `basicSprayCloud` is a short name (`typedef`) for an extra layer of `SprayCloud` on top of other layers of clouds which are similar to the cloud layers defined for the `basicReactingCloud` (which has been described earlier in this section 1.3.2). The difference now is that the templated base cloud class `Cloud` has been instantiated with class `basicSprayParcel`.
To see the definition of `basicSprayParcel` class, go to its directory and open file `basicSprayParcel.H`.

```
src
cd lagrangian/spray/parcels/derived/basicSprayParcel
vi basicSprayParcel.H
```

The definition of `basicSprayParcel` is:

```
typedef SprayParcel
<
    ReactingParcel
    <
        ThermoParcel
        <
            KinematicParcel
            <
                particle
            >
        >
    >
> basicSprayParcel;
```

So `basicSprayParcel` has now an extra layer of `SprayParcel` on top of the `basicReactingParcel` (which also has been described earlier in this section 1.3.2).

1.3.3 Submodels of classes

As described previously in the description of e.g. `basicReactingCloud` class that each layer adds some funtionality to the base layer. To see this description in OpenFOAM, go to

```
src
cd lagrangian/intermediate/clouds/Templates/ReactingCloud
vi ReactingCloud.H
```

In the header portion of this file `ReactingCloud.H`, it is written as

```
Class
    Foam::ReactingCloud

Description
    Templated base class for reacting cloud

    - Adds to thermodynamic cloud
      - Variable composition (single phase)
      - Phase change
```

The above desciption in header file shows that templated `ReactingCloud` class adds variable composition and phase change funtionalities to the base templated thermodnamic class `ThermoCloud`. Also go to

```
src
cd lagrangian/spray/parcels/Templates/SprayParcel
vi SprayParcel.H
```

In the header portion of `SprayParcel.H`, it is written as

```
Class
    Foam::SprayParcel

Description
    Reacting spray parcel, with added functionality for atomization and breakup
```

This descirption shows that templated `SprayParcel` adds functionalities of atomization and breakup to the templated `ReactingParcel`.

Location of submodels

To see all the available submodels for the templated `ReactingCloud` class, go to

```
src
cd lagrangian/intermediate/submodels/Reacting
tree -d
```

The above commands show following tree on terminal

```
CompositionModel
    CompositionModel
    NoComposition
    SingleMixtureFraction
    SinglePhaseMixture
InjectionModel
    ReactingLookupTableInjection
PhaseChangeModel
    LiquidEvaporation
    LiquidEvaporationBoil
    NoPhaseChange
    PhaseChangeModel
```

As described earlier in this section 1.3.3 (in the header file of `ReactingCloud.H`) that templated `ReactingCloud` class adds functionalities of composition and phase change to the templated `ThermoCloud` class and can be seen from the above tree. The avaialble submodels for e.g. composition models are `NoComposition`, `SingleMixtureFraction` and `SinglePhaseMixture`. In the tree the directory `CompositionModel` inside the directory `CompositionModel` is a virtual base class and other directories `NoComposition`, `SingleMixtureFraction` and `SinglePhaseMixture` are the submodels to the composition model to be selected on run time.
To see all the available submodels for the templated `SprayCloud` class, go to

```
src
cd lagrangian/spray/submodels
tree -d
```

The above commands show following tree on terminal

```
AtomizationModel
    AtomizationModel
    BlobsSheetAtomization
    LISAAtomization
    NoAtomization
BreakupModel
    BreakupModel
    ETAB
    NoBreakup
    PilchErdman
    ReitzDiwakar
    ReitzKHRT
    SHF
    TAB
StochasticCollision
    NoStochasticCollision
```

```
ORourkeCollision
StochasticCollisionModel
TrajectoryCollision
```

As described before in this section 1.3.3 (in the header file of `SprayParcel.H`) that templated `SprayParcel` class adds functionalities of atomization and breakup to the templated `ReactingParcel` class and can be seen from the above tree. The avaialble submodels for e.g atomization models are `BlobsSheetAtomization`, `LISAAtomization` and `NoAtomization` (atomization of spray is not modelled at all).

Adding or removing submodels

Go to

```
src
cd lagrangian/intermediate/parcels/derived/basicReactingParcel
vi makeBasicReactingParcelSubmodels.C
```

This file `makeBasicReactingParcelSubmodels.C`reads as:

```
makeParcelCloudFunctionObjects(basicReactingCloud);

// Kinematic sub-models
makeThermoParcelForces(basicReactingCloud);
makeParcelDispersionModels(basicReactingCloud);
makeReactingParcelInjectionModels(basicReactingCloud);
makeParcelPatchInteractionModels(basicReactingCloud);

// Thermo sub-models
makeParcelHeatTransferModels(basicReactingCloud);

// Reacting sub-models
makeReactingParcelCompositionModels(basicReactingCloud);
makeReactingParcelPhaseChangeModels(basicReactingCloud);
makeReactingParcelSurfaceFilmModels(basicReactingCloud);
```

This file contains all the funtionalities that will be added to the `basicReactingCloud` class. This file shows that `ReactingCloud` submodels are added to the `ThermoCloud` and `KinemticCloud` submodels.
If any other functionality needs to be added then it must be defined in the `submodels` directory and also added to this file `makeBasicReactingParcelSubmodels.C`. For removing any functionality, it must be removed from the submodels directory and the file `makeBasicReactingParcelSubmodels.C`.
Now to see all the functionalities for `basicSprayCloud` class, go to

```
src
cd lagrangian/spray/parcels/derived/basicSprayParcel
vi makeBasicSprayParcelSubmodels.C
```

This file `makeBasicSprayParcelSubmodels.C` reads as:

```
makeParcelCloudFunctionObjects(basicSprayCloud);

// Kinematic sub-models
makeThermoParcelForces(basicSprayCloud);
makeParcelDispersionModels(basicSprayCloud);
makeSprayParcelInjectionModels(basicSprayCloud);
makeParcelPatchInteractionModels(basicSprayCloud);
```

```
// Thermo sub-models
makeParcelHeatTransferModels(basicSprayCloud);

// Reacting sub-models
makeReactingParcelCompositionModels(basicSprayCloud);
makeReactingParcelPhaseChangeModels(basicSprayCloud);
makeReactingParcelSurfaceFilmModels(basicSprayCloud);

// Spray sub-models
makeSprayParcelAtomizationModels(basicSprayCloud);
makeSprayParcelBreakupModels(basicSprayCloud);
makeSprayParcelCollisionModels(basicSprayCloud);
```

This file shows that `SprayCloud` submodels are added to the `ReactingCloud`, `ThermoCloud` and `KinematicCloud` submodels.

Inheritance Diagrams of submodel classes

This portion shows and desrcibes the inheritance diagrams of submodel classes in OpenFOAM. The inheritance diagram for composition submodel classes is shown in figure 1.1.

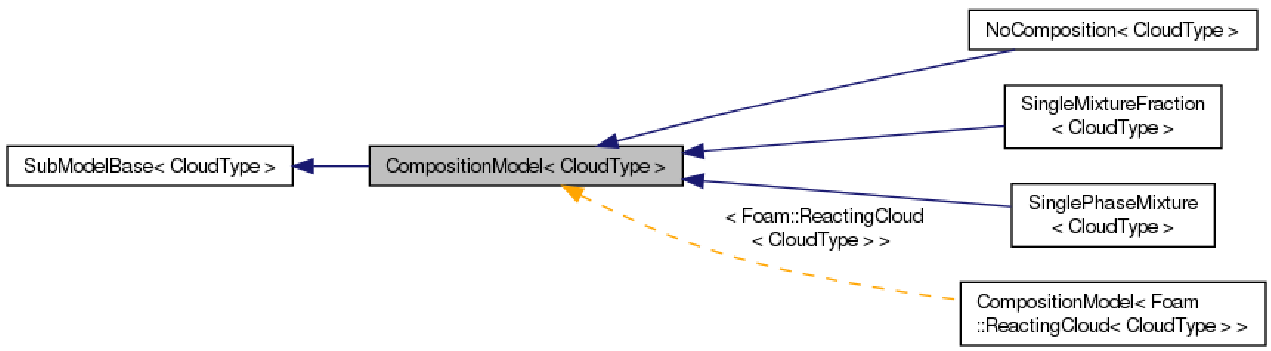

Figure 1.1: Inheritance diagram for composition submodel classes

The legend for all the figures in this section is:

```
A dark blue arrow is used to visualize a public inheritance relation between two classes.
A yellow dashed arrow denotes a relation between a template instance and the template
    class it was instantiated from. The arrow is labeled with the template parameters
    of the instance.
```

The figure 1.1 shows that the templated classes `NoComposition<CloudType>`, `SingleMixtureFormation<CloudType>` and `SinglePhaseMixture<CloudType>` are publically inherited from the templated base class `CompositionModel<CloudType>`. The template instance `CompositionModel<Foam::ReactingCloud<CloudType>>` is instantiated from template class `CompositionModel<CloudType>` with template parameter of instance `<Foam::ReactingCloud<CloudType>>`.

The inheritance diagram for phase change submodel classes is shown in figure 1.2. The figure 1.2

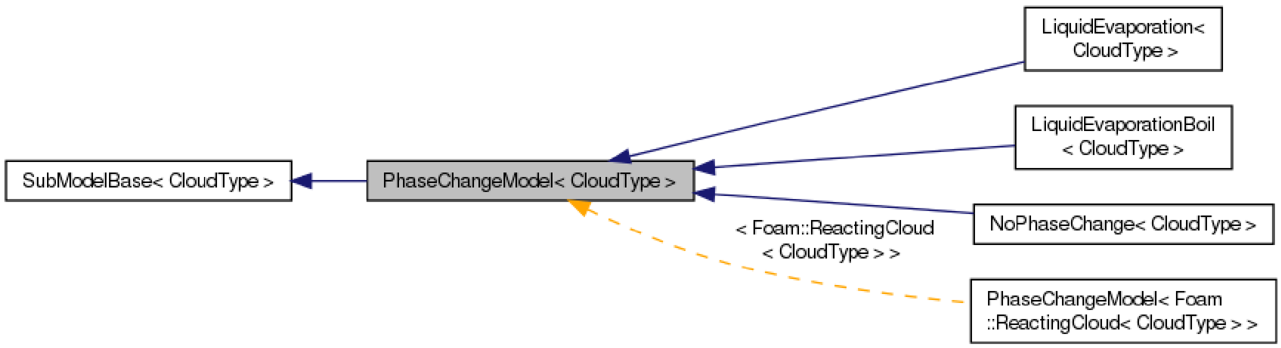

Figure 1.2: Inheritance diagram for phase change submodel classes

shows that the templated classes `NoPhaseChange<CloudType>`, `LiquidEvaporation<CloudType>` and `LiquidEvaporationBoil<CloudType>` are inherited publically from the templated base class `PhaseChaneModel<CloudType>`. The template instance
`PhaseChangeModel<Foam::ReactingCloud<CloudType>>` is instantiated from template class `PhaseChangeModel<CloudType>`.
The inheritance diagram for breakup submodel classes is shown in figure 1.3. The figure 1.3 shows

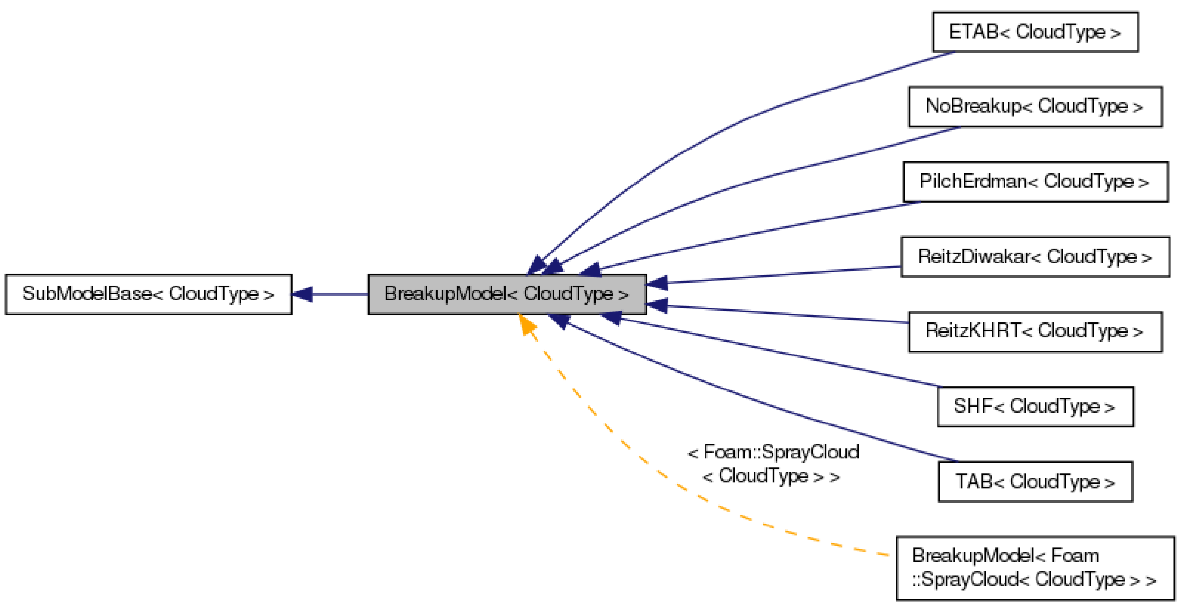

Figure 1.3: Inheritance diagram for breakup submodel classes

public inheritance of the templated classes `NoBreakup<CloudType>`, `ETAB<CloudType>`, `PilchErdman<CloudType>`, `ReitzDiwakar<CloudType>`, `ReitzKHRT<CloudType>`, `SHF<CloudType>` and `TAB<CloudType>` from the templated base class `BreakupModel<CloudType>`. The template instance
`BreakupModel<Foam::SprayCloud<CloudType>>` is instantiated from template class `BreakupModel<CloudType>` with template parameter of instance `<Foam::SprayCloud<CloudType>>`.

The inheritance diagram for atomization submodels is shown in figure 1.4. This figure shows that the templated classes `NoAtomization<CloudType>`, `BlobsSheetAtomization<CloudType>` and `LISAAtomization<CloudType>` are publically inherited from the templated base class `AtomizationModel<CloudType>`. The template instance `AtomizationModel<Foam::SprayCloud<CloudType>>` is instantiated from template class `AtomizationModel<CloudType>` with template parameter of instance `<Foam::SprayCloud<CloudType>>`.

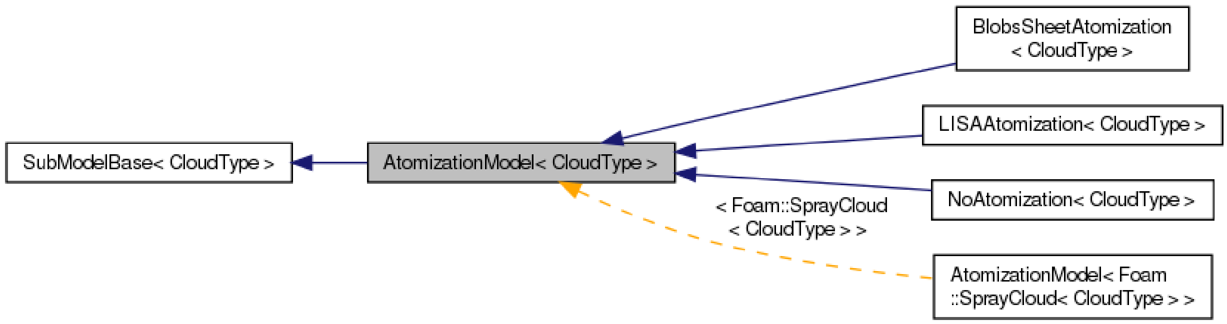

Figure 1.4: Inheritance diagram for atomization submodels

1.3.4 Use of cloud classes

The cloud class `basicSprayCloud` is used instead of `basicReactingCloud` when the effects of atomization, breakup and collision need to be included. The reason is that `basicSprayCloud` has extra submodels of atomization, breakup and collision included other than the submodels of `basicReactingCloud`.

1.3.5 Usage and modification of cloud properties in cases

This section shows how to use and modify the cloud properties in case setup.

basicReactingCloud

Go to OpenFOAM tutorials directory and copy the original case `evaporationTest` (related to solver `reactingParcelFoam`) to the `run` directory.

```
tut
cp -r lagrangian/reactingParcelFoam/evaporationTest/  $FOAM_RUN
```

The original case `evaporationTest` will not be modified and only studied here. Open the file `reactingCloud1Positions`

```
run
cd evaporationTest/constant
vi reactingCloud1Positions
```

This file reads as:

```
(
(0.002 0.002 0.00005)
(0.004 0.002 0.00005)
(0.006 0.002 0.00005)
(0.008 0.002 0.00005)
```

```
(0.010 0.002 0.00005)
(0.002 0.004 0.00005)
(0.004 0.004 0.00005)
(0.006 0.004 0.00005)
(0.008 0.004 0.00005)
(0.010 0.004 0.00005)
(0.002 0.006 0.00005)
(0.004 0.006 0.00005)
(0.006 0.006 0.00005)
(0.008 0.006 0.00005)
(0.010 0.006 0.00005)
(0.002 0.008 0.00005)
(0.004 0.008 0.00005)
(0.006 0.008 0.00005)
(0.008 0.008 0.00005)
(0.010 0.008 0.00005)
(0.002 0.010 0.00005)
(0.004 0.010 0.00005)
(0.006 0.010 0.00005)
(0.008 0.010 0.00005)
(0.010 0.010 0.00005)
)
```

This file shows the position of twenty five reacting particles in three dimensional coordinate system. Now open the dictionary file `reactingCloud1Properties`.

```
run
cd evaporationTest/constant
vi reactingCloud1Properties
```

This file `reactingCloud1Properties` contains information about submodels. The portion of this file about submodels reads as:

```
subModels
{
     particleForces
     {

     }

     injectionModel  manualInjection;

     dispersionModel none;

     patchInteractionModel standardWallInteraction;

     heatTransferModel none; // RanzMarshall;

     compositionModel singlePhaseMixture;

     phaseChangeModel none ; // liquidEvaporation;

     surfaceFilmModel none;

     radiation       off;
```

```
        manualInjectionCoeffs
        {
            massTotal       1e-3;
            parcelBasisType mass;
            SOI             0;
            positionsFile   "reactingCloud1Positions";
            U0              ( 0 -0.1 0 );
            sizeDistribution
            {
                type        uniform;
                uniformDistribution
                {
                    minValue        1e-04;
                    maxValue        1e-04;
                }
           }
        }

        standardWallInteractionCoeffs
        {
            type            rebound;
        }

        RanzMarshallCoeffs
        {
            BirdCorrection  true;
        }

        singlePhaseMixtureCoeffs
        {
            phases
            (
                liquid
                {
                    H2O          1;
                }
            );
        }

        liquidEvaporationCoeffs
        {
            enthalpyTransfer enthalpyDifference;

            activeLiquids    ( H2O );
        }
    }
```

The above file shows that the injection model being used in this case is `manualInjection` which is specified by `manualInjectionCoeffs`. The `manualInjectionCoeffs` contains information about mass and position of reacting particles. It also shows that velocity is `0.1 m/s` in negative Y-direction and particles are uniformly distributed. There is no dispersion and heat transfer model being used in the case. Also phase change, surface film and radiation effects are not being modelled. For interaction of particles with patches, `standardWallInteraction` model is used. The properties of `standardWallInteraction` model are specified by `standardWallInteractionCoeffs` which show

that particles will rebound on hitting the wall. The particles are composed of water droplets as defined by `singlePhaseMixtureCoeffs`. The `RanzMarshallCoeffs` and `liquidEvaporationCoeffs` corresponding to the models `RanzMarshall` and `liquidEvaporation` are also available in case but they are not being used for now (they are commented out). So submodels for reacting cloud can easily be selected and modified for any case by changing this file `reactingCloud1Properties`.
Now make the mesh, run the solver and open the post processor.

```
blockMesh
reactingParcelFoam
paraFoam
```

After post processing of results, figure 1.5 is obtained which shows that 25 water droplets are moving in Y-direction and bouncing up and down after striking the wall.

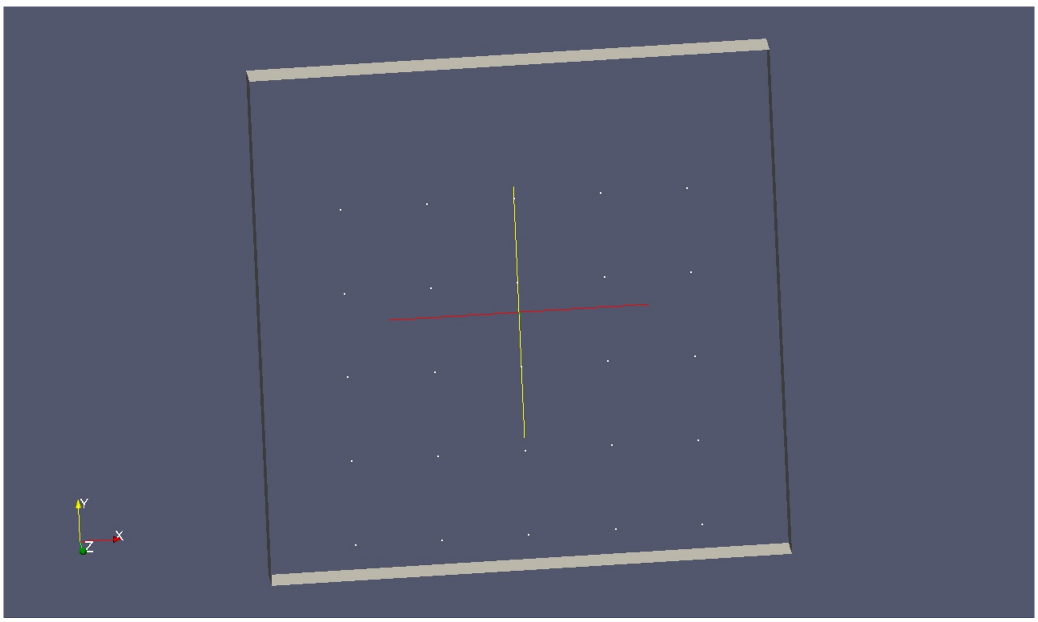

Figure 1.5: 25 water droplets inside the domain

basicSprayCloud

Go to tutorials directory of OpenFOAM and copy the case `aachenBomb` (corresponding to solver `sprayFoam`) to the OpenFOAM `run` directory.

```
tut
cp -r lagrangian/sprayFoam/aachenBomb/ $FOAM_RUN
```

The original case `aachenBomb` will not be modified and only studied here. Go to diectory `aachenBomb/constant` and open the dictionary file `sprayCloudProperties`.

```
run
cd aachenBomb/constant
vi sprayCloudProperties
```

The part of this file related to submodels reads as:

```
subModels
{
    particleForces
    {
        sphereDrag;
    }

    injectionModel  coneNozzleInjection;

    dispersionModel none;

    patchInteractionModel standardWallInteraction;

    heatTransferModel RanzMarshall;

    compositionModel singlePhaseMixture;

    phaseChangeModel liquidEvaporationBoil;

    surfaceFilmModel none;

    atomizationModel none;

    breakupModel    ReitzDiwakar; // ReitzKHRT;

    stochasticCollisionModel none;

    radiation       off;

    coneNozzleInjectionCoeffs
    {
        SOI             0;
        massTotal       6.0e-6;
        parcelBasisType mass;
        injectionMethod disc;
        flowType        flowRateAndDischarge;
        outerDiameter   1.9e-4;
        innerDiameter   0;
        duration        1.25e-3;
        position        ( 0 0.0995 0 );
        direction       ( 0 -1 0 );
        parcelsPerSecond 20000000;
        flowRateProfile table
        (
            (0 0.1272)
            (4.16667e-05 6.1634)
            (8.33333e-05 9.4778)
            (0.000125 9.5806)
            (0.000166667 9.4184)
            (0.000208333 9.0926)
            (0.00025 8.7011)
            (0.000291667 8.2239)
            (0.000333333 8.0401)
            (0.000375 8.845)
```

```
            (0.000416667 8.9174)
            (0.000458333 8.8688)
            (0.0005 8.8882)
            (0.000541667 8.6923)
            (0.000583333 8.0014)
            (0.000625 7.2582)
            (0.000666667 7.2757)
            (0.000708333 6.968)
            (0.00075 6.7608)
            (0.000791667 6.6502)
            (0.000833333 6.7695)
            (0.000875 5.5774)
            (0.000916667 4.8649)
            (0.000958333 5.0805)
            (0.001 4.9547)
            (0.00104167 4.5613)
            (0.00108333 4.4536)
            (0.001125 5.2651)
            (0.00116667 5.256)
            (0.00120833 5.1737)
            (0.00125 3.9213)
        );

        Cd              constant 0.9;

        thetaInner      constant 0.0;
        thetaOuter      constant 10.0;

        sizeDistribution
        {
            type        RosinRammler;

            RosinRammlerDistribution
            {
                minValue        1e-06;
                maxValue        0.00015;
                d               0.00015;
                n               3;
            }
       }
    }

    standardWallInteractionCoeffs
    {
        type            rebound;
    }

    RanzMarshallCoeffs
    {
        BirdCorrection  true;
    }

    singlePhaseMixtureCoeffs
    {
```

```
            phases
            (
                liquid
                {
                    C7H16               1;
                }
            );
        }
        liquidEvaporationBoilCoeffs
        {
            enthalpyTransfer enthalpyDifference;

            activeLiquids    ( C7H16 );
        }

        ReitzDiwakarCoeffs
        {
            solveOscillationEq yes;
            Cbag            6;
            Cb              0.785;
            Cstrip          0.5;
            Cs              10;
        }

    /*
        ReitzKHRTCoeffs
        {
            solveOscillationEq yes;
            B0              0.61;
            B1              40;
            Ctau            1;
            CRT             0.1;
            msLimit         0.2;
            WeberLimit      6;
        }
    */
        TABCoeffs
        {
            y0              0;
            yDot0           0;
            Cmu             10;
            Comega          8;
            WeCrit          12;
        }
    }
```

The above file shows that the injection model being used in this case is `coneNozzleInjection` which is specified by `coneNozzleInjectionCoeffs`. The `coneNozzleInjectionCoeffs` describes that the spray is injected as a cone nozzle spray with `RosinRammler` size distribution. It also gives information about total mass, start of injection, outer and inner diameter of cone nozzle, duration, position and direction of spray injection, nozzle discharge coefficients, parcels injected per second and flow rate profile of spray injection. There is no dispersion, stochastic collision and atomization model being used in the case. Also surface film and radiation effects are neglected. The interaction of spray with wall patches is modelled such that the spray parcels will be rebounded on hitting the wall as defined by `patchInteractionModel` and `standardWallInteractionCoeffs`. The particles are composed of liquid n-heptane as defined by `singlePhaseMixtureCoeffs`. The breakup model is `ReitzDiwakar` which is defined by `ReitzDiwakarCoeffs`. Two other breakup models `ReitzKHRT` and `TAB` have their properties defined by `ReitzKHRTCoeffs` and `TABCoeffs` but are not being used here. So submodels for spray cloud can be selected and modified by doing changes in this file `sprayCloudProperties`. Now run this case after making mesh.

```
blockMesh
sprayFoam
paraFoam
```

Post processing of results gives figure 1.6 which shows the cone shaped spray injection of liquid n-heptane.

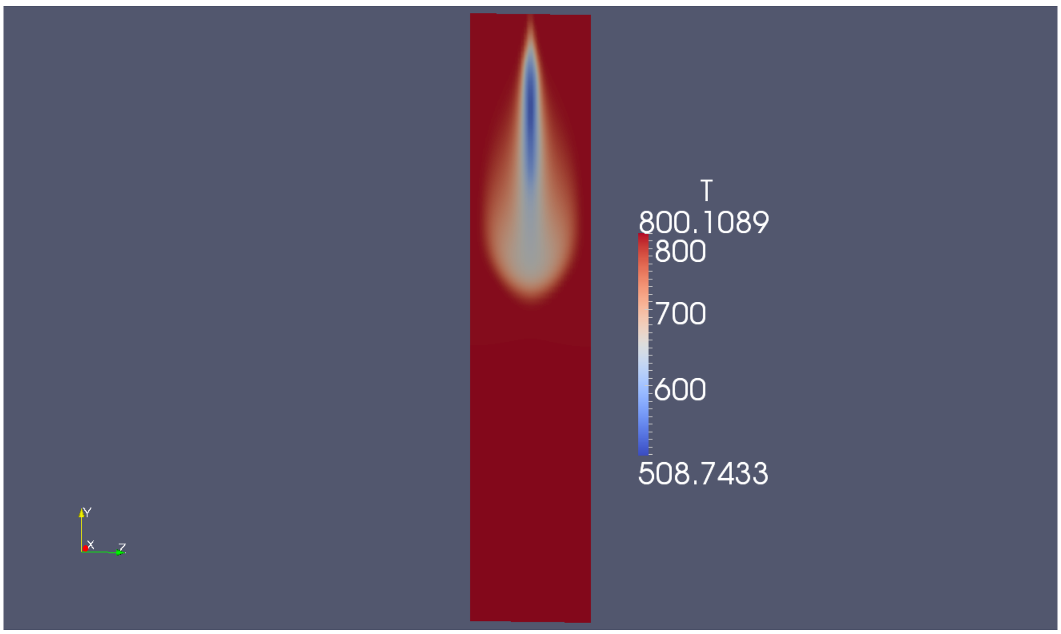

Figure 1.6: Cone shaped spray injection

1.4 Creating and adding new submodel

This section shows how to create and add a new submodel. Go to `run` directory and copy the submodels there:

```
run
cp -r $FOAM_SRC/lagrangian/intermediate/ .
cd intermediate/submodels/Reacting/PhaseChangeModel/
```

Create a new submodel from an old one:

```
cp -r NoPhaseChange MyPhaseChange
cd MyPhaseChange
mv NoPhaseChange.C MyPhaseChange.C
mv NoPhaseChange.H MyPhaseChange.H
sed -i s/NoPhaseChange/MyPhaseChange/g MyPhaseChange.H
sed -i s/NoPhaseChange/MyPhaseChange/g MyPhaseChange.C
sed -i s/none/MyPhaseChange/g MyPhaseChange.H
sed -i s/false/true/g MyPhaseChange.C
```

Add the newly created submodel as:

```
run
cd intermediate/parcels/include/
vi makeReactingParcelPhaseChangeModels.H
```

Add the following two lines to the file `makeReactingParcelPhaseChangeModels.H`:

```
#include "MyPhaseChange.H"
```

and

```
makePhaseChangeModelType(MyPhaseChange, CloudType);
```

Now go to directory `intermediate/Make`.

```
run
cd intermediate/Make/
vi files
```

Replace the last line in the `files` as:

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

Now compile the library.

```
cd ..
wclean
wmake libso
```