Cite as: Markale, I.: Discrete multiphase modeling of electrostatic sprays. In Proceedings of CFD with OpenSource Software, 2016, Edited by Nilsson. H.,
http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2016

CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Project work:

Discrete multiphase modeling of electrostatic sprays

Developed for OpenFOAM-4.0.x

Author:
Ishaan Markale
ishaan@student.chalmers.se

Peer reviewed by:
Sampann Arora
Håkan Nilsson

Disclaimer: This is a student project work, done as part of a course where OpenFOAM and some other OpenSource software are introduced to the students. Any reader should be aware that it might not be free of errors. Still, it might be useful for someone who would like learn some details similar to the ones presented in the report and in the accompanying files. The material has gone through a review process. The role of the reviewer is to go through the tutorial and make sure that it works, that it is possible to follow, and to some extent correct the writing. The reviewer has no responsibility for the contents.

January 23, 2017

Contents

Learning Outcomes

The reader will learn:

How to use it:

- how to use the sprayFoam solver.
- how to post-process Lagrangian particles in paraview.

The theory of it:

- the theory of Lagrangian Particle Tracking.

How it is implemented:

- the implementation of different forces on Lagrangian particles in OpenFOAM.
- the structure of the lagrangian library.

How to modify it:

- how to modify the sprayFoam solver for solving the electrostatic field.
- how to implement the electrostatic force on Lagrangian particles.

Chapter 1

Introduction

1.1 Motivation

Water mist for fire protection is usually created by injecting water under high pressure. The high pressure creates small water droplets which are injected through a nozzle into quiescent air. The nozzle produces a conical spray shape. The spread angle of the spray and the droplet size depends on the injection pressure. If a continuous current is applied to the inlet nozzle with an electric potential in the range of 20kV, there is an effect on the spray properties. The aim of this project is to model this phenomenon in OpenFOAM to see the effects on the spray due to the charge on the injected droplets. A comparison will be made between the sprays resulting with and without the charging of the droplets.

According to the experiments performed [1], it is seen that applying a potential to the nozzle leads to wider cone angles, more homogeneously distributed spray patterns, and reduced droplet sizes than sprays without any charged droplets. These droplets are affected by the electrostatic field in the domain and they also affect each other due a repelling electrostatic force. These effects lead to a wider cone angle of the resulting spray. The electric potential also affects the atomisation of the droplets leading to changes in the resulting droplet size distribution.

1.2 Lagrangian Particle Tracking (LPT)

The motion of particles inside a fluid can be modeled in two ways, using Eulerian-Eulerian modeling or using Lagrangian Particle Tracking. In this particular problem, there is a movement of particles (here in the form of droplets) in the the flow of a fluid (air). Dispersed two phase flows can either be :

1. Dense - The spacing between particles is short and the particle transport is mainly due to collisions. Eulerian-Eulerian modeling is suitable when there is a large particle concentration where two-way coupling between the discrete and continuous phase and particle-particle collisions are significant and need to be taken into account.

2. Dilute - The spacing between the particles is large. Here the fluid dynamic forces are the main responsible for the transport of particles. For this case, the Eulerian-Lagrangian approach is employed in which the continuum equations are solved for the continuous phase whereas Newton's equations of motion are solved for each particle or a group of particles, hence the name Lagrangian Particle Tracking. In this problem, the volume fraction of the droplets is low enough ($< 10^{-3}$) to use the LPT approach [2], [3].

The equations of motion of a particle can be given by

$$m_P \frac{d\mathbf{u}_P}{dt} = \sum \mathbf{F}_i \tag{1.1}$$

and

$$\frac{dx_p}{dt} = \mathbf{u}_P \tag{1.2}$$

The forces, $\mathbf{F}_i$ acting on the particle are modeled in order to solve for the particle velocity, $\mathbf{u_P}$ which in turn gives the trajectory of the particle, x_P. Solving the equations of motion for the particles can be done in many different regimes i.e.,

1. One-way coupled - The flow field affects the particles but the particles do not affect the flow field.
2. Two-way coupled - The flow field affects the particle motion and also the force on the flow field due the particles is taken into account.
3. Four-way coupled - Two way coupling along with particle-particle interaction is taken into account.

1.2.1 Relevant Forces

The equation of motion will be solved when the forces acting on each particle are known. The forces considered are the drag force $\mathbf{F}_D$, the gravity force $\mathbf{F}_g$ and the electrostatic force $\mathbf{F}_E$. In order to simply the problem and reduce the computational time, several modeling assumptions have been made. Forces such as the history force, added mass force, pressure gradient force and buoyancy force have been considered negligible [3]. Now, the equation of motion of a droplet is

$$m_P \frac{d\mathbf{u}_P}{dt} = \mathbf{F}_D + \mathbf{F}_g + \mathbf{F}_E \tag{1.3}$$

where $\mathbf{F}_D$ is the drag force given by ([2])

$$\mathbf{F}_D = m_P \frac{18\mu}{\rho_P d_P^2} \frac{C_D Re_P (\mathbf{u} - \mathbf{u}_P)}{24} \tag{1.4}$$

Re_P is the particle Reynolds number, $Re_P = \rho d_P(\mathbf{u} - \mathbf{u}_P)/\mu$. The drag coefficient C_D is given by ([2])

$$C_D = \begin{cases} \frac{24}{Re_P}, & \text{if } Re_P < 1 \\ \frac{24}{Re_P}(1 + 0.5 Re_P^{0.687}), & \text{if } 1 \leq Re_P \leq 1000 \\ 0.44, & \text{if } Re_P > 1000 \end{cases}$$

$\mathbf{F}_g = m_P \mathbf{g}$ is the gravitational force, and $\mathbf{F}_E = q\mathbf{E}$ is the electrostatic force on the particle. q is the electric charge of the fluid particle and $\mathbf{E}$ is the electrostatic field at the location of the particle.

The electrostatic field depends on the electrostatic potential ϕ. From the experiments performed by Ochoterena, et al. [1], the boundary conditions for the electrostatic potential are known. The electric field is then computed as

$$\mathbf{E} = -\nabla\phi \tag{1.5}$$

1.2.2 Simplifying assumptions

As mentioned earlier, certain particle forces have been neglected. Two-way coupling between the flow field and particles is considered. Particle-particle interactions have been neglected. Finally, all particles are assumed spherical in shape.

The Poisson's equation relates the electrostatic potential to the space charge ρ (for constant electric conductivity σ) as

$$-\nabla^2\phi = \frac{\rho}{\epsilon} \tag{1.6}$$

where ϵ is the permittivity of the surrounding medium. The electrostatic field will apply a force on the charged droplets. Simultaneously, the droplets will change the space charge in the domain leading to a change in the electric potential and hence a change in the electric field. Hence there is a two way coupling between the electric field and the droplets. Additionally, the droplets repel each other due to their charge. Thus, droplet-droplet interactions also exist. However, for the purpose of this report, only one-way coupling between the electric field and the droplets is implemented. This means that the space charge ρ is assumed to be zero for the purpose of this report.

Chapter 2

OpenFOAM Implementation

Since it is computationally expensive to track a large number of particles, it is preferred to track a *parcel* instead. A parcel is essentially a computational particle i.e., it contains a number of *real* particles. Instead of tracking real particles, parcels are tracked. The assumption is that a parcel behaves the same was as a particle i.e., it has the same properties as that of a single particle. A collection of parcels is called a spray and a cloud is a collection of Lagrangian particles.

2.1 The sprayFoam solver

`sprayFoam` is a transient solver for compressible, turbulent flow with a spray particle cloud. Lagrangian Particle Tracking is used to track the particles and the usual continuity and momentum equations are solved for the carrier fluid. This solver is located in `$FOAM_SOLVERS/lagrangian/sprayFoam`. We will start by looking at the general algorithm and working of the solver.

```
OF4x
cd $FOAM_SOLVERS/lagrangian/sprayFoam
vim sprayFoam.C
```

The following code shows the time loop in `sprayFoam.C`

```
...
     while (runTime.run())
    {
        #include "readTimeControls.H"
        #include "compressibleCourantNo.H"
        #include "setDeltaT.H"

        runTime++;

        Info<< "Time = " << runTime.timeName() << nl << endl;

        parcels.evolve();

        #include "rhoEqn.H"

        // --- Pressure-velocity PIMPLE corrector loop
        while (pimple.loop())
        {
```

```
            #include "UEqn.H"
            #include "YEqn.H"
            #include "EEqn.H"

            // --- Pressure corrector loop
            while (pimple.correct())
            {
                #include "pEqn.H"
            }

            if (pimple.turbCorr())
            {
                turbulence->correct();
            }
        }

        rho = thermo.rho();

        if (runTime.write())
        {
            combustion->dQ()().write();
        }

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
    }

    Info<< "End\n" << endl;

    return 0;
...
```

The working of the solver is as follows

1. Computations for a time-step start. The time is incremented from the previous time step using `runTime++`.
2. Particle cloud is evolved (either injected or moved depending on the previous time step) according to the forces acting on them using the `evolve` function of the `parcel` class.
3. The continuity equation for the density is solved which is specified in the `rhoEqn.H` header file.
4. The PIMPLE loop begins.
5. The momentum equation and transport equation for the mass fractions of the species is solved.
6. The pressure corrector loop begins.
7. The `turbCorr` function then determines whether to correct turbulence at every outer iteration or only at the last iteration.
8. The above three steps are performed till the residual control conditions are met.
9. The density is recalculated from the corrected pressure.
10. The next time step starts

The `sprayFoam` solver includes the header file `basicSprayCloud.H`, which is the basic cloud class to introduce reacting spray parcels.

2.2 The lagrangian library

Since it is computationally expensive to track a large number of particles, it is preferred to track a *parcel* instead. A parcel is essentially a computational particle i.e. it contains a number of *real* particles. Instead of tracking real, particles, parcels are tracked. The assumption is that a parcel behaves the same was as a particle i.e. it has the same properties as that of a single particle. A collection of parcels is called a spray and a cloud is a collection of Lagrangian particles.
Now we will go to the directory where the lagrangian methods are implemented, and investigate the relevant implementations.

```
cd $WM_PROJECT_DIR/src/lagrangian/
ls
```

The following directories are seen.

```
basic
coalCombustion
distributionModels
DSMC
intermediate
molecularDynamics
solidParticle
spray
turbulence
```

It is seen that the `lagrangian` library consists of many sub-directories which consist of various LPT implementations. For example, the `basic` directory consists of the `particle` directory and `Cloud` directory among other directories. The `basic/particle` directory contains the base particle class and the `basic/Cloud` directory contains the base cloud class templated on particle type. This can be seen by

```
vim basic/particle/particle.H
vim basic/Cloud/Cloud.H
```

The `intermediate` library is a sub-directory of the `lagrangian` library and it consists of different models and forces for LPT. It contains some important directories i.e., `parcels` and `submodels`. The sub-directory `parcels` contains many types of particles in which different submodels (which are present in the `submodels` sub-directory) are added. The `parcels` directory has the sub-directories `Templates` and `derived`. The `Templates` sub-directory has different templated particle types and the `derived` has different particles types and this adds funtionalities to the class via the `submodels` directory. These submodels are available in the `lagrangian/intermediate/submodels` directory. There are submodels associated with different particle types and they can be seen by typing

```
tree -d intermediate/submodels/
```

From the above discussion, it is seen that the `lagrangian` library is templated. This is also observed in `basicSprayCloud.H` which is responsible for introducing spray parcels (as mentioned in Section[2.1]). The definition of `basicSprayCloud.H` can be seen by typing

```
vim $WM_PROJECT_DIR/src/lagrangian/spray/clouds/derived/basicSprayCloud/basicSprayCloud.H
```

```
namespace Foam
{
    typedef SprayCloud
    <
        ReactingCloud
        <
            ThermoCloud
            <
                KinematicCloud
                <
                    Cloud
                    <
                        basicSprayParcel
                    >
                >
            >
        >
    > basicSprayCloud;
}
```

This shows that `basicSprayCloud` is a typedef for the templated class `SprayCloud` on top of other templates of different types. The class `cloud` is instantiated with the `basicSprayParcel`. `basicSprayParcel` is contained in `lagrangian/spray/parcels/derived/basicSprayParcel` which is itself a `typedef` of `SprayParcel` and is instantiated with the `particle` class at the lowest level. It's definition is can be seen by typing

```
vim $WM_PROJECT_DIR/src/lagrangian/spray/parcels/derived/basicSprayParcel/basicSprayParcel.H
```

```
namespace Foam
{
    typedef SprayParcel
    <
        ReactingParcel
        <
            ThermoParcel
            <
                KinematicParcel
                <
                    particle
                >
            >
        >
    > basicSprayParcel;

    template<>
    inline bool contiguous<basicSprayParcel>()
```

```
    {
        return false;
    }
}
```

2.3 Creating a new solver - electrostaticSprayFoam

The already existing solver `sprayFoam` will be modified to create the electrostaticSprayFoam solver. This will solve for the electric field and implement one-way coupling to take into account the effect of this electric field on the charged particles. The electric field depends on the electrostatic potential ϕ. Once the electrostatic potential field is known the electric field is calculated according to Equation[1.5].
Now the basic file structure is created in the user directory. The `--parents` flag is used to preserve the same file structure as in the OpenFOAM installation.

```
foam
cp -r --parents applications/solvers/lagrangian/sprayFoam/ $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR/applications/solvers/lagrangian
mv sprayFoam electrostaticSprayFoam
cd electrostaticSprayFoam
wclean
```

We dont need the sub-solvers `sprayDyMFoam` and `sprayEngineFoam`, and we should re-name the file name to be the same as the class name.

```
rm -r sprayDyMFoam sprayEngineFoam
mv sprayFoam.C electrostaticSprayFoam.C
```

The `Make/files` needs to be modified. This can be done by

```
string="electrostaticSprayFoam.C\nEXE=\$(FOAM_USER_APPBIN)/electrostaticSprayFoam"
printf "%b\n" "$string" > Make/files
```

Add the Laplace equation for the electric potential in the top level solver i.e., in `electrostaticSprayFoam.C`. Add the electric potential equation (Equation [1.6]) just before the PIMPLE loop. Although it is not necessary to include this equation inside the time loop when only one-way coupling is implemented, the electric field **E** will change with time when the effect of particles on the field inside the domain is taken into account.

```
        solve
        (
                fvm::laplacian(ElPot)
        );

        E=-fvc::grad(ElPot);
```

Also comment the include statement `//#include "EEqn.H"` since we don't need to solve the energy equation.

The electric potential (`ElPot`) needs to be constructed as a `volScalarField` which is read at runtime from the `0` directory. This construction in done in `createFields.H`. Add the following just before the include statements at the end of `createFields.H`.

```
volScalarField ElPot
(
    IOobject
    (
        "ElPot",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
```

Similarly, the electric field (`E`) needs to be constructed as a `volVectorField` and also needs initialized in the `createFields.H` file. Inside `createFields.H` add the following,

```
volVectorField E
(
    IOobject
    (
        "E",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    -fvc::grad(ElPot)
);
```

Add the following to the `Make/options` file (where ... are the original lines). This is because the `electrostaticSprayFoam` solver reads the `#include "YEqn.H"` from the `reactingParcelFoam` solver.

```
EXE_INC = \
    ...\
    -I$(FOAM_APP)/solvers/lagrangian/reactingParcelFoam
```

Now compile the solver using `wmake`.

Note - The `Make/options` file has been given in the appendix.

2.4 Coupling of the intermediate library with the electrostaticSprayFoam solver

The `$WM_PROJECT_DIR/src/lagrangian/intermediate/submodels` directory contains different templated submodels that can be additionally be applied to the lagrangian cloud classes.The `submodels` directory contains a directory called `Kinematic`, the contents of which are seen by:

```
tree -d $FOAM_SRC/lagrangian/intermediate/submodels/Kinematic/
```

It is seen that the `Kinematic` directory contains various kinematic properties that can be implemented for lagrangian particles. For example, it contains implementations of collision models, injection models, particle forces dispersion models etc. The sub-directory `ParticleForces` contains all forces that can be applied to the lagrangian particles. Type

```
tree -d $FOAM_SRC/lagrangian/intermediate/submodels/Kinematic/ParticleForces
```

It contains the following directories for different particle forces:

```
Drag
├── forceSuSp
├── Gravity
├── Lift
├── Paramagnetic
├── ParticleForce
├── PressureGradient
├── VirtualMass
└── SRF
```

2.4.1 Creating a new particle force - ElectrostaticForce.C

Once the solver has been created, the next step is to implement one way coupled Lagrangian Particle Tracking between the charged particles and the electrostatic field. This is done by creating a new `ParticleForce` class called `ElectrostaticForce`.

First, the necessary libraries required to couple the solver are to be recompiled in the user worling directory i.e. in `$WM_PROJECT_USER_DIR`.

```
cd $WM_PROJECT_USER_DIR
cp -r $FOAM_SRC/lagrangian $WM_PROJECT_USER_DIR/src/
cd src/lagrangian/intermediate/submodels/Kinematic/ParticleForces/
```

Note - The entire lagrangian directory is copied to the user working directory. The reason for this will be explained later.
The paramagnetic force class `ParamagneticForce.C` will be modified to implement the one-way coupled electrostatic force. Rename all files according to the new particle force 'ElectrostaticForce'. Next, all instances of the class `Paramagnetic` are replaced with the new class `Electrostatic` in the three files `ElectrostaticForce.C`, `ElectrostaticForce.H` and `ElectrostaticForceI.H`. Next, all

instances of the `magneticSusceptibility` scalar is replaced by the `charge` scalar. `paramagnetic` is the typname which is looked for in the sprayCloudProperties file during runtime, hence this needs to be changed to `electrostatic` and this keyword will be supplied when giving the electrostatic field (see Section[3.2]).

```
cp -r Paramagnetic Electrostatic
cd Electrostatic
mv ParamagneticForce.C ElectrostaticForce.C
mv ParamagneticForce.H ElectrostaticForce.H
mv ParamagneticForceI.H ElectrostaticForceI.H
sed -i s/Paramagnetic/Electrostatic/g ElectrostaticForce*
sed -i s/magneticSusceptibility/charge/g ElectrostaticForce*
sed -i s/paramagnetic/electrostatic/g ElectrostaticForce.H
```

`HdotGradH` is the field strength of the paramagnetic force which is changed to `E`. The charge on the particle is read from the dictionary file "sprayCloudProperties" in the case folder and is specified as a scalar. In ElectrostaticForce.C replace

```
this->coeffs().template lookupOrDefault<word>("HdotGradH", "HdotGradH")
```

with

```
this->coeffs().template lookupOrDefault<word>("E", "E")
```

This way, the solver knows what keyword to look for when the user specifies the electric field as a dictionary during the initialization of the case.
The equation for calculating the electrostatic force needs to be changed. In ElectrostaticForce.C, replace:

```
template<class CloudType>
Foam::forceSuSp Foam::ParamagneticForce<CloudType>::calcNonCoupled
(
    const typename CloudType::parcelType& p,
    const scalar dt,
    const scalar mass,
    const scalar Re,
    const scalar muc
) const
{
    forceSuSp value(Zero, 0.0);

    const interpolation<vector>& HdotGradHInterp = *HdotGradHInterpPtr_;

    value.Su()=
        mass*3.0*constant::electromagnetic::mu0.value()/p.rho()
       *magneticSusceptibility_/(magneticSusceptibility_ + 3)
       *HdotGradHInterp.interpolate(p.position(), p.currentTetIndices());

    return value;
}
```

with

```
template<class CloudType>
Foam::forceSuSp Foam::ElectrostaticForce<CloudType>::calcNonCoupled
(
    const typename CloudType::parcelType& p,
    const scalar dt,
    const scalar mass,
    const scalar Re,
    const scalar muc
) const
{
    forceSuSp value(Zero, 0.0);

    const interpolation<vector>& HdotGradHInterp = *HdotGradHInterpPtr_;

    value.Su()=charge_*HdotGradHInterp.interpolate(p.position(),
    p.currentTetIndices());

    return value;
}
```

Note that the force is calculated on each particle by multiplying its charge with the electric field at the location of the charge. This means that the electric field has to be interpolated. This is because the particle mesh and fluid mesh is the same, however the particle size is much smaller than the mesh size. After calculation, the force is stored in `forceSuSp`, which is a helper container class for both implicit and explicit terms. `Su` is the explicit contribution which is directly calculated as a force whereas `Sp` is the implicit contribution which is calculated as $(force/velocity)$. The total force is hence given by $S_P(\mathbf{u}-\mathbf{u}_P)+S_u$. In the member function it is seen that the force is `calcNonCoupled`. This means that there is no source term added to the governing equation of the electric field. Some other forces like drag and lift forces are of `calcCoupled` instead.

Now, the constructor for this new particle force needs to be added to the hash table held by the base parcel class. This is done by:

```
cd $WM_PROJECT_USER_DIR/src/lagrangian/intermediate
vim parcels/include/makeParcelForces.H
```

The `makeParcelForces.H` file should look like

```
#ifndef makeParcelForces_H
#define makeParcelForces_H

#include "SphereDragForce.H"
#include "NonSphereDragForce.H"
#include "WenYuDragForce.H"
#include "ErgunWenYuDragForce.H"
#include "PlessisMasliyahDragForce.H"
```

```
#include "SaffmanMeiLiftForce.H"
#include "TomiyamaLiftForce.H"

#include "GravityForce.H"
#include "NonInertialFrameForce.H"
#include "ParamagneticForce.H"
#include "PressureGradientForce.H"
#include "SRFForce.H"
#include "VirtualMassForce.H"
#include "ElectrostaticForce.H"

#define makeParcelForces(CloudType)                                           \
                                                                              \
    makeParticleForceModel(CloudType);                                        \
    makeParticleForceModelType(SphereDragForce, CloudType);                   \
    makeParticleForceModelType(NonSphereDragForce, CloudType);                \
    makeParticleForceModelType(WenYuDragForce, CloudType);                    \
    makeParticleForceModelType(ErgunWenYuDragForce, CloudType);               \
    makeParticleForceModelType(PlessisMasliyahDragForce, CloudType);          \
    makeParticleForceModelType(SaffmanMeiLiftForce, CloudType);               \
    makeParticleForceModelType(TomiyamaLiftForce, CloudType);                 \
    makeParticleForceModelType(GravityForce, CloudType);                      \
    makeParticleForceModelType(NonInertialFrameForce, CloudType);             \
    makeParticleForceModelType(ParamagneticForce, CloudType);                 \
    makeParticleForceModelType(PressureGradientForce, CloudType);             \
    makeParticleForceModelType(SRFForce, CloudType);                          \
    makeParticleForceModelType(VirtualMassForce, CloudType);                  \
    makeParticleForceModelType(ElectrostaticForce, CloudType);

#endif
```

Similarly, add the Electrostatic force constructor to `parcels/include/makeThermoParcelForces.H`.

```
vim parcels/include/makeThermoParcelForces.H
```

The `makeThermoParcelForces.H` file should look like

```
#define makeThermoParcelForces_H

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#include "SphereDragForce.H"
#include "NonSphereDragForce.H"

#include "SaffmanMeiLiftForce.H"
#include "TomiyamaLiftForce.H"

#include "GravityForce.H"
#include "NonInertialFrameForce.H"
#include "ParamagneticForce.H"
#include "PressureGradientForce.H"
```

```
#include "SRFForce.H"
#include "VirtualMassForce.H"
#include "ElectrostaticForce.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#define makeThermoParcelForces(CloudType)                                              \
                                                                                       \
    makeParticleForceModel(CloudType);                                                 \
    makeParticleForceModelType(SphereDragForce, CloudType);                            \
    makeParticleForceModelType(NonSphereDragForce, CloudType);                         \
    makeParticleForceModelType(SaffmanMeiLiftForce, CloudType);                        \
    makeParticleForceModelType(TomiyamaLiftForce, CloudType);                          \
    makeParticleForceModelType(GravityForce, CloudType);                               \
    makeParticleForceModelType(NonInertialFrameForce, CloudType);                      \
    makeParticleForceModelType(ParamagneticForce, CloudType);                          \
    makeParticleForceModelType(PressureGradientForce, CloudType);                      \
    makeParticleForceModelType(SRFForce, CloudType);                                   \
    makeParticleForceModelType(VirtualMassForce, CloudType);                           \
    makeParticleForceModelType(ElectrostaticForce, CloudType);


// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#endif
```

Now the files inside the intermediate library are ready to be compiled. However, when the library is compiled and used for a case, the solver gives the following error:

```
Constructing particle forces
    Selecting particle force sphereDrag
    Selecting particle force gravity
    Selecting particle force electrostatic


--> FOAM FATAL ERROR:
Unknown particle force type electrostatic, constructor not in hash table

    Valid particle force types are:

12
(
BrownianMotion
SRF
SaffmanMeiLiftForce
TomiyamaLift
distortedSphereDrag
gravity
nonInertialFrame
nonSphereDrag
paramagnetic
```

```
pressureGradient
sphereDrag
virtualMass
)
```

This problem might be because of some missing link in the templating of the `intermediate` library. To solve this problem, the whole `lagrangian` library can be compiled in the user directory.

```
cd $WM_PROJECT_USER_DIR/src/lagrangian/
./Allclean
```

Note - The Allclean file is supplied along with this report. Its is also given in the appendix. It is important to change the location directory to the user's directory in `Make/files` of all directories inside the `lagrangian` library. The correct paths also need to be given in the `Make/options`. This can be done by

```
sed -i s/FOAM_LIBBIN/FOAM_USER_LIBBIN/g */Make/files
sed -i s/FOAM_LIBBIN/FOAM_USER_LIBBIN/g molecularDynamics/*/Make/files
sed -i 's@I$(LIB_SRC)/lagrangian@I..@g' */Make/options
sed -i 's@I$(LIB_SRC)/lagrangian@I../..@g' molecularDynamics/*/Make/options
```

The `lagrangian` library can now be compiled using the `Allwmake` file existing in the `lagrangian` library. Compile the library using

```
./Allwmake
```

Note - It is important to note that OpenFOAM will use this user compiled library instead of the one located in `$FOAM_SRC/lagrangian/` whenever the lagrangian library is used. The order in which OpenFOAM reads these libraries can be found by typing `echo $LD_LIBRARY_PATH` in the terminal.

2.4.2 Changes to electrostaticSprayFoam

Now that the library is created, the next step is to link it to the electrostaticSprayFoam solver. Go to the solver directory

```
cd $WM_PROJECT_USER_DIR/applications/solvers/lagrangian/electrostaticSprayFoam/
```

The Make/options file should contain the following so that the compiler knows where to look for the files.

```
EXE_INC = \
    -I. \
    -I../reactingParcelFoam \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I${LIB_SRC}/meshTools/lnInclude \
    -I${LIB_SRC}/sampling/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/compressible/lnInclude \
    -I$(WM_PROJECT_USER_DIR)/src/lagrangian/basic/lnInclude \
```

```
    -I$(WM_PROJECT_USER_DIR)/src/lagrangian/intermediate/lnInclude \
    -I$(WM_PROJECT_USER_DIR)/src/lagrangian/spray/lnInclude \
    -I$(WM_PROJECT_USER_DIR)/src/lagrangian/distributionModels/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/specie/lnInclude \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/basic/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/properties/liquidProperties/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/properties/liquidMixtureProperties/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/properties/solidProperties/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/properties/solidMixtureProperties/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/thermophysicalFunctions/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/reactionThermo/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/SLGThermo/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/chemistryModel/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/radiation/lnInclude \
    -I$(LIB_SRC)/ODE/lnInclude \
    -I$(LIB_SRC)/regionModels/regionModel/lnInclude \
    -I$(LIB_SRC)/regionModels/surfaceFilmModels/lnInclude \
    -I$(LIB_SRC)/combustionModels/lnInclude \
    -I$(FOAM_APP)/solvers/lagrangian/reactingParcelFoam

EXE_LIBS = \
    -lturbulenceModels \
    -lcompressibleTurbulenceModels \
    -llagrangian \
    -llagrangianTurbulence \
    -llagrangianSpray \
    -lspecie \
    -lcompressibleTransportModels \
    -lfluidThermophysicalModels \
    -lliquidProperties \
    -lliquidMixtureProperties \
    -lsolidProperties \
    -lsolidMixtureProperties \
    -lthermophysicalFunctions \
    -lreactionThermophysicalModels \
    -lSLGThermo \
    -lchemistryModel \
    -lradiationModels \
    -lODE \
    -lregionModels \
    -lsurfaceFilmModels \
    -lcombustionModels \
    -lfiniteVolume \
    -lfvOptions \
    -lmeshTools \
    -lsampling \
    -L$(FOAM_USER_LIBBIN) \
    -llagrangianIntermediate
```

The solver is now ready to be recompiled. Type in the terminal

```
wclean
wmake
```

Chapter 3

Case Setup

3.1 Geometry

The geometry for this case is shown in Figure[3.1]. It consists of a cube of width $8cm$ with a spray inlet at the center of the top boundary.

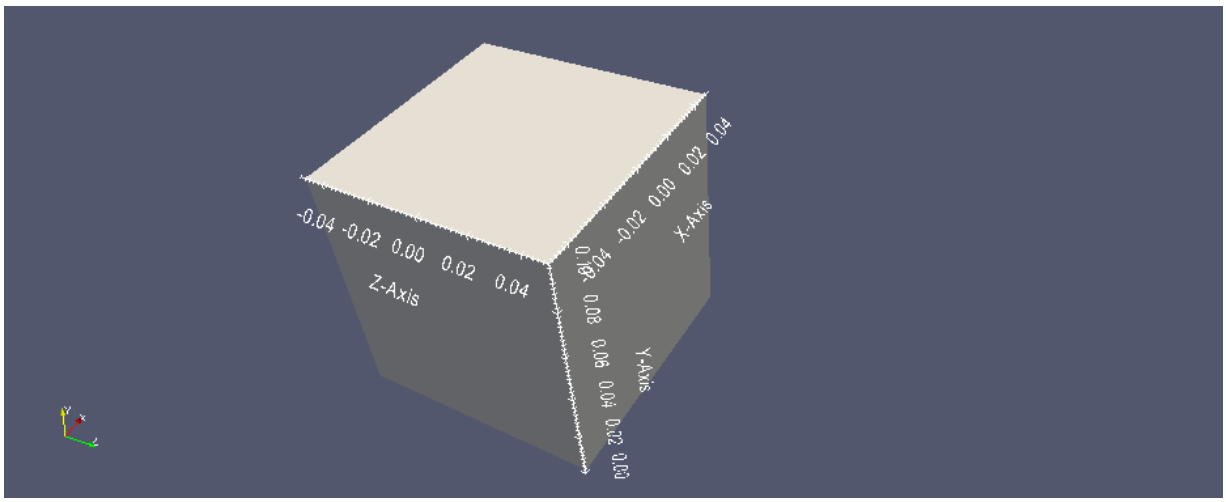


Figure 3.1: Geometry

3.2 Pre-processing

The `aachemBomb` tutorial for the `sprayFoam` solver will be modified for this problem. Copy the sprayFoam tutorial to the run directory.

```
OF4x
cp -r $FOAM_TUTORIALS/lagrangian/sprayFoam/aachenBomb $FOAM_RUN
cd $FOAM_RUN/aachenBomb
vim system/blockMeshDict
```

It is seen that all the walls are contained in a single patch type called `walls`. Create new patches called `topWall` and `otherWalls`. This is done so that the boundary condition of the electric potential can be applied at the top wall. The shape of the geometry is changed to form a cube. Therefore,

the vertices of the domain need to be changed in the `blockMeshDict`. The number of blocks have also been reduced to reduce the computational time. The `blockMeshDict` dictionary is given in the Appendix.

The 0/ directory

The existing files need to modified in the 0 directory in order to account for the change in the `blockMeshDict`. Two new dictionary files need to be created, for `ElPot` and `E`. These files are needed in order to construct the cloud. The `topWall` patch is assigned an electric potential of $20kV$ and since the rest of the walls are supposed to be ground, the electric potential is zero.

`ElPot`

```
/*--------------------------------------------------------------------------*\
| =========                 |                                                |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  1.4                                  |
|   \\  /    A nd           | Web:      http://www.openfoam.org              |
|    \\/     M anipulation  |                                                |
\*--------------------------------------------------------------------------*/

// Field Dictionary

FoamFile
{
    version         2.0;
    format          ascii;
    class           volScalarField;
    object          ElPot;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [1 2 -3 0 0 -1 0]; // [kg * m^2 /( s^3 A)]

internalField   uniform 0;

boundaryField
{
    topWall
    {

        type            fixedValue;
        value           uniform 20000;
      }

    otherWalls
    {
      type            fixedValue;
      value           uniform  0;
    }
}
```

The electric field is initialised to 0 everywhere initially and is solved for in the Laplace equation.

E

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      E;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [1 1 -3 0 0 -1 0]; // [kg * m /( s^3 A)]

internalField       uniform (0 0 0);

boundaryField
{
    topWall
    {
        type               fixedValue;
        value              uniform (0 0 0);
    }
    otherWalls
    {
        type               fixedValue;
        value              uniform (0 0 0);
    }
}

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

The constant directory

Water droplets are being injected into quiescent air. Hence, there is no need to solve for the chemistry. Inside `chemistryProperties`, turn chemistry `off`. Inside `combustionProperties`, make combustion inactive by setting `active=no`. The $k-\epsilon$ turbulence model is specified in `turbulenceProperties`. The injected liquid needs to be changed to water. Type

```
sed -i s/C7H16/H2O/g constant/thermoPhysicalProperties
```

The `sprayCloudProperties` gives details of the injected particle cloud. This file is available in the appendix. Inportant things to note are:

1. The maximum allowed Courant number is set to 1 in order for the time step to not become very small.

2. The interpolation scheme used for the electric field `E` is `cellPoint`.

3. The electric field and charge are specified inside `particleForces`. A charge of $-6e-9C$ is given to the particles in order to match the current passing through the spray (from experiments performed by Ochoterena, et al.[1]) for a given mass flow rate. For running the simulation without charge, the charge can just be specified as zero.

4. The injection is of type `coneNozzleInjection` with an initial spread angle of 15^o and a size distribution of the particles given by the Rosin-Rammler probability distribution function.

The system directory

In `fvSchemes`, the `gradSchemes` for `ElPot` is set as `Gauss Linear`. The rest of the schemes are the same as that of the tutorial case. In the `fvSolution` file, add the following for the `ElPot` solver.

```
ElPot
    {
        solver           PCG;
        preconditioner   DIC;
        tolerance        1e-08;
        relTol           0;
    };
```

In `fvSolution` a relaxation factor of 0.2 is specified to enhance convergence. An initial time step of $2.5e-7$ is set in the `controlDict` and it is runtime modifiable.
Note - The `fvSolution`, `fvSchemes` and `controlDict` files have been provided in the appendix and also hae been attached with this report. After making all the changes mentioned above, the case is run

```
blockMesh
checkMesh
electrostaticSprayFoam >& log&
```

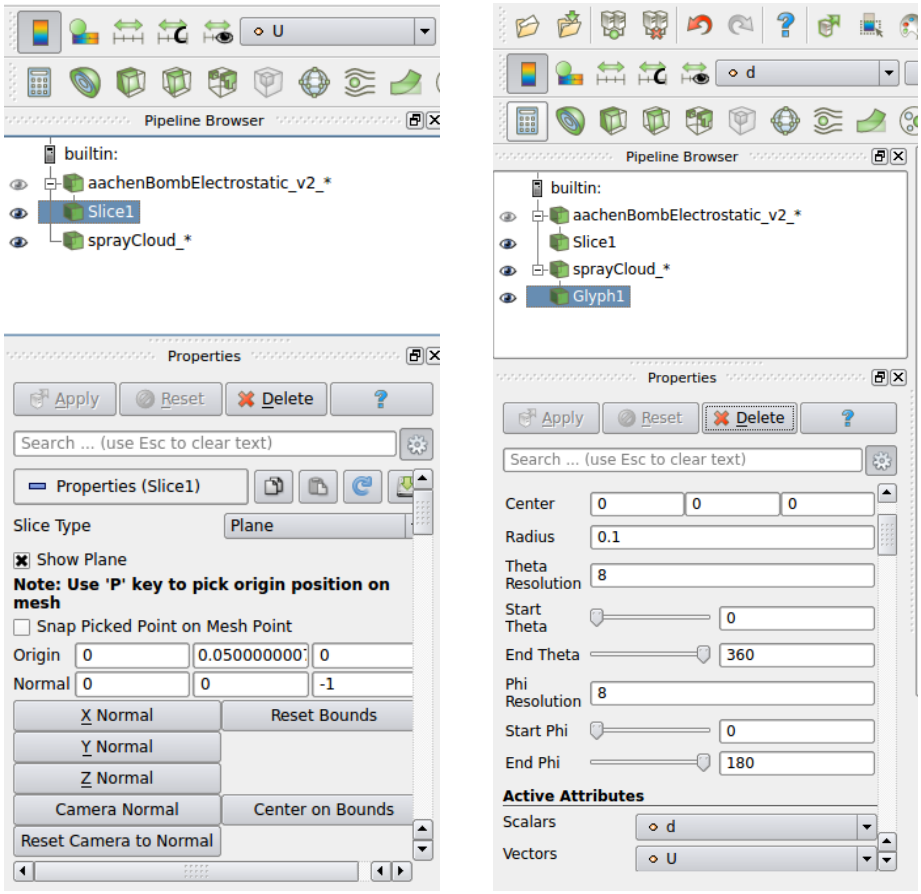
3.3 Post-Processing and results

The case is run two times, first without any charge on the droplets and next with giving charge to the droplets. The resulting spray can be compared to see the effects of the charge on the droplet behavior. An easy way to post-process Lagrangian particles is to transform the case data into VTK format and visualise the particles.

```
foamToVTK
```

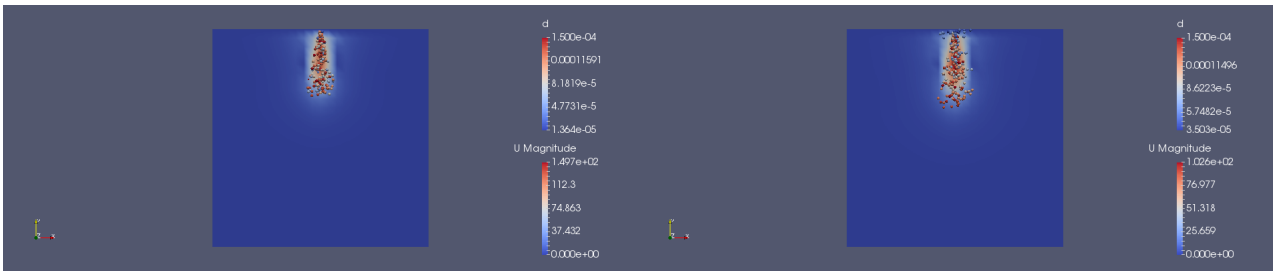
Once, the VTK files have been generated, open `paraview` and open all the `.VTK` files inside the `VTK` directory. The directory `sprayCloud` contains the files for the particles. Create a slice for the fluid region and spherical glyphs for the sprayCloud with the properties as shown in Figure 3.2.

Figures [3.3a]-[3.3h] depict the evolution of the spray cloud and water droplets as a function of time. The left column represents the droplets without any charge and the right column represents the droplets with charge. The time is increasing from the top row to the bottom row.

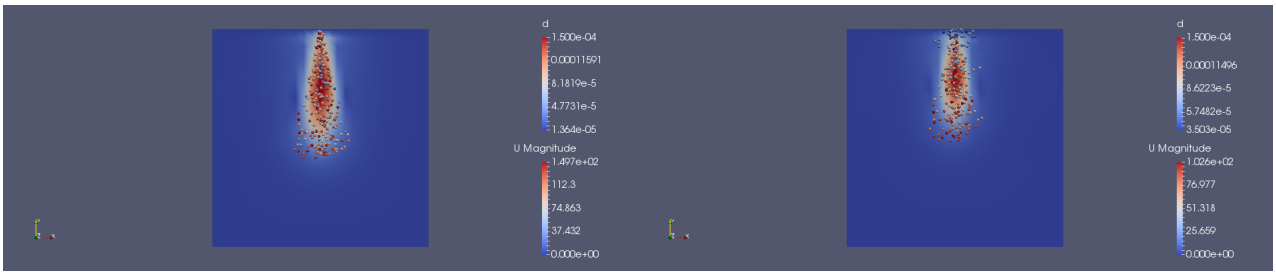


(a) Slice (b) Glyph

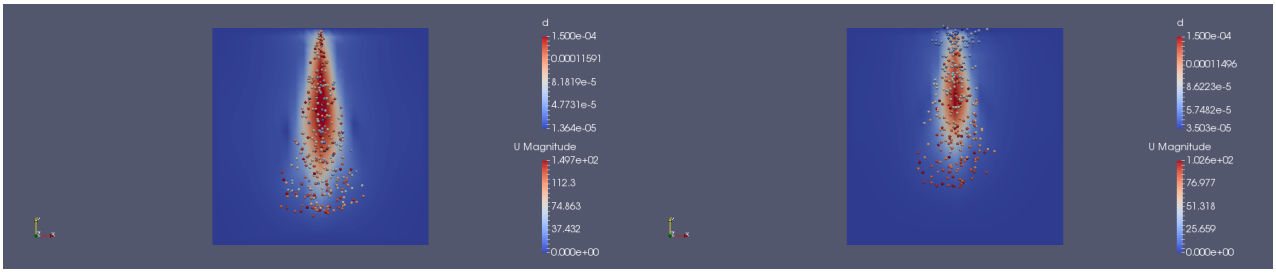
Figure 3.2: Visualisation settings



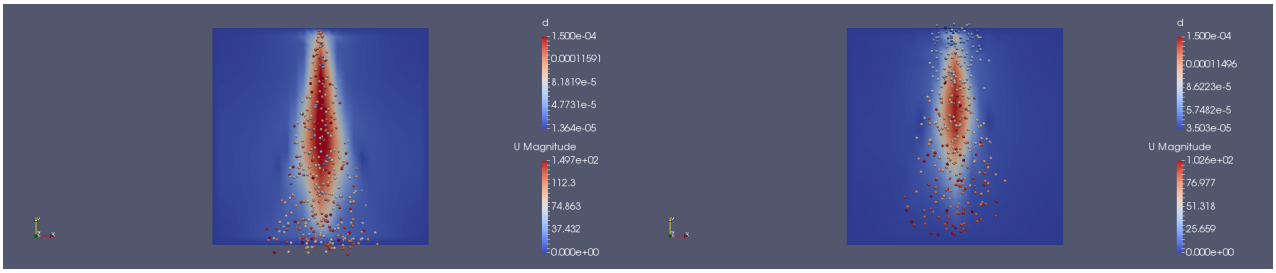
(a) Without Charge (b) With Charge



(c) Without Charge (d) With Charge



(e) Without Charge (f) With Charge



(g) Without Charge (h) With Charge

Figure 3.3: Velocity contours and particle tracks as a function of time

Chapter 4

Further Work

The next step in the problem is to add the effect of the charged droplets on the electric field. This will be done by solving the Poisson's equation for the space charge which will in turn affect the charged droplets. Hence two-way coupling between the electric field and droplets can be implemented. Particle-particle interaction for collisions has already been implemented in OpenFOAM. Similarly, the interaction between charged droplets needs to be implemented. This will essentially take into account 4-way coupling between the continuous phase and the droplets.

Droplet atomisation and droplet size distribution may be affected due to this electrostatic potential. These phenomena can be tried to be modeled as well.

Chapter 5

Study Questions

1. State the Poisson's equation for the electric potential.
2. What is the purpose of the interpolate function while calculating the particle forces?
3. How does momentum transfer take place between the discrete and continuous phase in OpenFOAM?

Bibliography

[1] Raul Ochoterena, Ola Willstrand and Michael Försth. *Electrosprays for fire suppression.* SP Technical Research Institute of Sweden.

[2] Crowe, T.C. et al., *Multiphase flows with droplets and particles.* CRC Press, (2011).

[3] Sommerfeld, M., van Wachem, B., Oliemans, R., 2008. Best Practice Guidelines for Computational Fluid Dynamics of Dispersed Multiphase Flows. ERCOFTAC.

Appendix A

Source Files

Make/files

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

Make/options

```
EXE_INC = \
    -I. \
    -I../reactingParcelFoam \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I${LIB_SRC}/meshTools/lnInclude \
    -I${LIB_SRC}/sampling/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/compressible/lnInclude \
    -I$(WM_PROJECT_USER_DIR)/src/test/lagrangian/basic/lnInclude \
    -I$(WM_PROJECT_USER_DIR)/src/test/lagrangian/intermediate/lnInclude \
    -I$(WM_PROJECT_USER_DIR)/src/test/lagrangian/spray/lnInclude \
    -I$(WM_PROJECT_USER_DIR)/src/test/lagrangian/distributionModels/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/specie/lnInclude \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/basic/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/properties/liquidProperties/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/properties/liquidMixtureProperties/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/properties/solidProperties/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/properties/solidMixtureProperties/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/thermophysicalFunctions/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/reactionThermo/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/SLGThermo/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/chemistryModel/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/radiation/lnInclude \
    -I$(LIB_SRC)/ODE/lnInclude \
    -I$(LIB_SRC)/regionModels/regionModel/lnInclude \
    -I$(LIB_SRC)/regionModels/surfaceFilmModels/lnInclude \
    -I$(LIB_SRC)/combustionModels/lnInclude
```

```
EXE_LIBS = \
    -lturbulenceModels \
    -lcompressibleTurbulenceModels \
    -llagrangian \
    -llagrangianTurbulence \
    -llagrangianSpray \
    -lspecie \
    -lcompressibleTransportModels \
    -lfluidThermophysicalModels \
    -lliquidProperties \
    -lliquidMixtureProperties \
    -lsolidProperties \
    -lsolidMixtureProperties \
    -lthermophysicalFunctions \
    -lreactionThermophysicalModels \
    -lSLGThermo \
    -lchemistryModel \
    -lradiationModels \
    -lODE \
    -lregionModels \
    -lsurfaceFilmModels \
    -lcombustionModels \
    -lfiniteVolume \
    -lfvOptions \
    -lmeshTools \
    -lsampling \
    -L$(FOAM_USER_LIBBIN) \
    -llagrangianIntermediate
```

createFields.H

```
#include "readGravitationalAcceleration.H"

Info<< "Creating combustion model\n" << endl;

autoPtr<combustionModels::psiCombustionModel> combustion
(
    combustionModels::psiCombustionModel::New(mesh)
);

psiReactionThermo& thermo = combustion->thermo();
thermo.validate(args.executable(), "h", "e");

SLGThermo slgThermo(mesh, thermo);

basicSpecieMixture& composition = thermo.composition();
PtrList<volScalarField>& Y = composition.Y();

const word inertSpecie(thermo.lookup("inertSpecie"));

if (!composition.contains(inertSpecie))
```

```
{
    FatalErrorInFunction
        << "Specified inert specie '" << inertSpecie << "' not found in "
        << "species list. Available species:" << composition.species()
        << exit(FatalError);
}

volScalarField& p = thermo.p();

volScalarField rho
(
    IOobject
    (
        "rho",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    thermo.rho()
);

Info<< "\nReading field U\n" << endl;
volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

#include "compressibleCreatePhi.H"

mesh.setFluxRequired(p.name());

dimensionedScalar rhoMax
(
    dimensionedScalar::lookupOrDefault
    (
        "rhoMax",
        pimple.dict(),
        dimDensity,
        GREAT
    )
);

dimensionedScalar rhoMin
```

```
(
    dimensionedScalar::lookupOrDefault
    (
        "rhoMin",
        pimple.dict(),
        dimDensity,
        0
    )
);

Info<< "Creating turbulence model\n" << endl;
autoPtr<compressible::turbulenceModel> turbulence
(
    compressible::turbulenceModel::New
    (
        rho,
        U,
        phi,
        thermo
    )
);

// Set the turbulence into the combustion model
combustion->setTurbulence(turbulence());

Info<< "Creating field dpdt\n" << endl;
volScalarField dpdt
(
    IOobject
    (
        "dpdt",
        runTime.timeName(),
        mesh
    ),
    mesh,
    dimensionedScalar("dpdt", p.dimensions()/dimTime, 0)
);

Info<< "Creating field kinetic energy K\n" << endl;
volScalarField K("K", 0.5*magSqr(U));
multivariateSurfaceInterpolationScheme<scalar>::fieldTable fields;

forAll(Y, i)
{
    fields.add(Y[i]);
}
fields.add(thermo.he());

volScalarField dQ
(
    IOobject
    (
```

```
        "dQ",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    mesh,
    dimensionedScalar("dQ", dimEnergy/dimTime, 0.0)
);

volScalarField ElPot
(
    IOobject
    (
"ElPot",
runTime.timeName(),
mesh,
IOobject::MUST_READ,
IOobject::AUTO_WRITE
    ),
    mesh
);

volVectorField E
(
    IOobject
    (
"E",
runTime.timeName(),
mesh,
IOobject::NO_READ,
IOobject::AUTO_WRITE
    ),
    -fvc::grad(ElPot)
);

#include "createMRF.H"
#include "createClouds.H"
#include "createRadiationModel.H"
```

ElectrostaticForce.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2016 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.
```

```
    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Class
    Foam::ElectrostaticForce

Description
    Calculates particle electrostatic (magnetic field) force

SourceFiles
    ElectrostaticForceI.H
    ElectrostaticForce.C

\*---------------------------------------------------------------------------*/

#ifndef ElectrostaticForce_H
#define ElectrostaticForce_H

#include "ParticleForce.H"
#include "interpolation.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{

class fvMesh;

/*---------------------------------------------------------------------------*\
                      Class ElectrostaticForce Declaration
\*---------------------------------------------------------------------------*/

template<class CloudType>
class ElectrostaticForce
:
    public ParticleForce<CloudType>
{
    // Private data

        //- Name of electrostatic field strength field - default = "HdotGradH"
        const word HdotGradHName_;
```

```
        //- HdotGradH interpolator
        const interpolation<vector>* HdotGradHInterpPtr_;

        //- Magnetic susceptibility of particle
        const scalar charge_;


public:

    //- Runtime type information
    TypeName("electrostatic");


    // Constructors

        //- Construct from mesh
        ElectrostaticForce
        (
            CloudType& owner,
            const fvMesh& mesh,
            const dictionary& dict
        );

        //- Construct copy
        ElectrostaticForce(const ElectrostaticForce& gf);

        //- Construct and return a clone
        virtual autoPtr<ParticleForce<CloudType>> clone() const
        {
            return autoPtr<ParticleForce<CloudType>>
            (
                new ElectrostaticForce<CloudType>(*this)
            );
        }


    //- Destructor
    virtual ~ElectrostaticForce();


    // Member Functions

        // Access

            //- Return the name of electrostatic field strength field
            const word& HdotGradHName() const;

            //- Return the magnetic susceptibility of particle
            scalar charge() const;


        // Evaluation
```

```
            //- Cache fields
            virtual void cacheFields(const bool store);

            //- Calculate the non-coupled force
            virtual forceSuSp calcNonCoupled
            (
                const typename CloudType::parcelType& p,
                const scalar dt,
                const scalar mass,
                const scalar Re,
                const scalar muc
            ) const;
};


// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

} // End namespace Foam

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#include "ElectrostaticForceI.H"

#ifdef NoRepository
    #include "ElectrostaticForce.C"
#endif

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#endif

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

ElectrostaticForce.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2016 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.
```

```
    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

\*---------------------------------------------------------------------------*/

#include "ElectrostaticForce.H"
#include "demandDrivenData.H"
#include "electromagneticConstants.H"

// * * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * * //

template<class CloudType>
Foam::ElectrostaticForce<CloudType>::ElectrostaticForce
(
    CloudType& owner,
    const fvMesh& mesh,
    const dictionary& dict
)
:
    ParticleForce<CloudType>(owner, mesh, dict, typeName, true),
    HdotGradHName_
    (
        this->coeffs().template lookupOrDefault<word>("E", "E")
    ),
    HdotGradHInterpPtr_(NULL),
    charge_
    (
        readScalar(this->coeffs().lookup("charge"))
    )
{}


template<class CloudType>
Foam::ElectrostaticForce<CloudType>::ElectrostaticForce
(
    const ElectrostaticForce& pf
)
:
    ParticleForce<CloudType>(pf),
    HdotGradHName_(pf.HdotGradHName_),
    HdotGradHInterpPtr_(pf.HdotGradHInterpPtr_),
    charge_(pf.charge_)
{}


// * * * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * * * //
```

```
template<class CloudType>
Foam::ElectrostaticForce<CloudType>::~ElectrostaticForce()
{}


// * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * //

template<class CloudType>
void Foam::ElectrostaticForce<CloudType>::cacheFields(const bool store)
{
    if (store)
    {
        const volVectorField& HdotGradH =
            this->mesh().template lookupObject<volVectorField>(HdotGradHName_);

        HdotGradHInterpPtr_ = interpolation<vector>::New
        (
            this->owner().solution().interpolationSchemes(),
            HdotGradH
        ).ptr();
    }
    else
    {
        deleteDemandDrivenData(HdotGradHInterpPtr_);
    }
}


template<class CloudType>
Foam::forceSuSp Foam::ElectrostaticForce<CloudType>::calcNonCoupled
(
    const typename CloudType::parcelType& p,
    const scalar dt,
    const scalar mass,
    const scalar Re,
    const scalar muc
) const
{
    forceSuSp value(Zero, 0.0);

    const interpolation<vector>& HdotGradHInterp = *HdotGradHInterpPtr_;

    value.Su()=charge_*HdotGradHInterp.interpolate(p.position(), p.currentTetIndices());

    return value;
}


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

ElectrostaticForceI.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

\*---------------------------------------------------------------------------*/

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

template<class CloudType>
inline const Foam::word&
Foam::ElectrostaticForce<CloudType>::HdotGradHName() const
{
    return HdotGradHName_;
}


template<class CloudType>
inline Foam::scalar
Foam::ElectrostaticForce<CloudType>::charge() const
{
    return charge_;
}


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

electrostaticSprayFoam.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2016 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Application
    sprayFoam

Description
    Transient solver for compressible, turbulent flow with a spray particle
    cloud.

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "turbulentFluidThermoModel.H"
#include "basicSprayCloud.H"
#include "psiCombustionModel.H"
#include "radiationModel.H"
#include "SLGThermo.H"
#include "pimpleControl.H"
#include "fvOptions.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
    #include "postProcess.H"

    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createControl.H"
    #include "createTimeControls.H"
```

```
    #include "createFields.H"
    #include "createFieldRefs.H"
    #include "createFvOptions.H"
    #include "compressibleCourantNo.H"
    #include "setInitialDeltaT.H"
    #include "initContinuityErrs.H"

    turbulence->validate();

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n" << endl;

    while (runTime.run())
    {
        #include "readTimeControls.H"
        #include "compressibleCourantNo.H"
        #include "setDeltaT.H"

        runTime++;

        Info<< "Time = " << runTime.timeName() << nl << endl;

        parcels.evolve();

        #include "rhoEqn.H"

solve
(
fvm::laplacian(ElPot)
);

E = -fvc::grad(ElPot);


        // --- Pressure-velocity PIMPLE corrector loop
        while (pimple.loop())
        {
            #include "UEqn.H"
            #include "YEqn.H"
 //           #include "EEqn.H"

            // --- Pressure corrector loop
            while (pimple.correct())
            {
                #include "pEqn.H"
            }

            if (pimple.turbCorr())
            {
                turbulence->correct();
            }
```

```
        }

        rho = thermo.rho();

        if (runTime.write())
        {
            combustion->dQ()().write();
        }

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }

    Info<< "End\n" << endl;

    return 0;
}


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

createFields.H

```
#include "readGravitationalAcceleration.H"

Info<< "Creating combustion model\n" << endl;

autoPtr<combustionModels::psiCombustionModel> combustion
(
    combustionModels::psiCombustionModel::New(mesh)
);

psiReactionThermo& thermo = combustion->thermo();
thermo.validate(args.executable(), "h", "e");

SLGThermo slgThermo(mesh, thermo);

basicSpecieMixture& composition = thermo.composition();
PtrList<volScalarField>& Y = composition.Y();

const word inertSpecie(thermo.lookup("inertSpecie"));

if (!composition.contains(inertSpecie))
{
    FatalErrorInFunction
        << "Specified inert specie '" << inertSpecie << "' not found in "
        << "species list. Available species:" << composition.species()
        << exit(FatalError);
}
```

```
volScalarField& p = thermo.p();

volScalarField rho
(
    IOobject
    (
        "rho",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    thermo.rho()
);

Info<< "\nReading field U\n" << endl;
volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

#include "compressibleCreatePhi.H"

mesh.setFluxRequired(p.name());

dimensionedScalar rhoMax
(
    dimensionedScalar::lookupOrDefault
    (
        "rhoMax",
        pimple.dict(),
        dimDensity,
        GREAT
    )
);

dimensionedScalar rhoMin
(
    dimensionedScalar::lookupOrDefault
    (
        "rhoMin",
        pimple.dict(),
        dimDensity,
        0
```

```
    )
);

Info<< "Creating turbulence model\n" << endl;
autoPtr<compressible::turbulenceModel> turbulence
(
    compressible::turbulenceModel::New
    (
        rho,
        U,
        phi,
        thermo
    )
);

// Set the turbulence into the combustion model
combustion->setTurbulence(turbulence());

Info<< "Creating field dpdt\n" << endl;
volScalarField dpdt
(
    IOobject
    (
        "dpdt",
        runTime.timeName(),
        mesh
    ),
    mesh,
    dimensionedScalar("dpdt", p.dimensions()/dimTime, 0)
);

Info<< "Creating field kinetic energy K\n" << endl;
volScalarField K("K", 0.5*magSqr(U));
multivariateSurfaceInterpolationScheme<scalar>::fieldTable fields;

forAll(Y, i)
{
    fields.add(Y[i]);
}
fields.add(thermo.he());

volScalarField dQ
(
    IOobject
    (
        "dQ",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    mesh,
```

```
    dimensionedScalar("dQ", dimEnergy/dimTime, 0.0)
);

volScalarField ElPot
(
    IOobject
    (
"ElPot",
runTime.timeName(),
mesh,
IOobject::MUST_READ,
IOobject::AUTO_WRITE
    ),
    mesh
);

volVectorField E
(
    IOobject
    (
"E",
runTime.timeName(),
mesh,
IOobject::NO_READ,
IOobject::AUTO_WRITE
    ),
    -fvc::grad(ElPot)
);

#include "createMRF.H"
#include "createClouds.H"
#include "createRadiationModel.H"
```

Appendix B

Allclean

Allclean

```
#!/bin/sh
cd ${0%/*} || exit 1    # Run from this directory

# Parse arguments for library compilation
targetType=${1-libso}
set -x

wclean $targetType distributionModels
wclean $targetType basic
wclean $targetType solidParticle
wclean $targetType intermediate
wclean $targetType turbulence
wclean $targetType spray
wclean $targetType DSMC
wclean $targetType coalCombustion
wclean $targetType$ molecularDynamics
```

Appendix C

Setup Files

blockMeshDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.001;

vertices
(
    (-50 0 -50)
    (-50 0 50)
    (50 0 50)
    (50 0 -50)
    (-50 100 -50)
    (-50 100 50)
    (50 100 50)
    (50 100 -50)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (16 16 15) simpleGrading (1 1 1)
);

edges
```

```
(
);

patches
(
    wall otherWalls
    (
        (2 6 5 1)
        (0 4 7 3)
        (0 1 5 4)
        (7 6 2 3)
        (3 2 1 0)
    )
    wall topWall
    (
        (4 5 6 7)
    )
);

mergePatchPairs
(
);

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

thermophysicalProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      thermophysicalProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

thermoType
{
    type            hePsiThermo;
    mixture         reactingMixture;
    transport       sutherland;
    thermo          janaf;
    energy          sensibleEnthalpy;
```

```
    equationOfState perfectGas;
    specie          specie;
}

CHEMKINFile     "$FOAM_CASE/chemkin/chem.inp";
CHEMKINThermoFile "$FOAM_CASE/chemkin/therm.dat";
CHEMKINTransportFile "$FOAM_CASE/chemkin/transportProperties";

newFormat       yes;

inertSpecie     N2;

liquids
{
    H2O
    {
        defaultCoeffs   yes;
    }
}

solids
{
    // none
}

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

sprayCloudProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      binary;
    class       dictionary;
    location    "constant";
    object      SprayCloudProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solution
{
    active          true;
    coupled         true;
```

```
    transient       yes;
    cellValueSourceCorrection on;
    maxCo           1;

    sourceTerms
    {
        schemes
        {
            rho             explicit 1;
            U               explicit 1;
            Yi              explicit 1;
            h               explicit 1;
            radiation       explicit 1;
        }
    }

    interpolationSchemes
    {
        rho             cell;
        U               cellPoint;
        thermo:mu       cell;
        T               cell;
        Cp              cell;
        kappa           cell;
        p               cell;
E   cellPoint;
    }

    integrationSchemes
    {
        U               Euler;
        T               analytical;
    }
}


constantProperties
{
    T0              298;

    // place holders for rho0 and Cp0
    // - reset from liquid properties using T0
    rho0            1000;
    Cp0             4187;

    constantVolume  true;
}


subModels
{
    particleForces
```

```
    {
        sphereDrag;
gravity;
//paramagnetic
        //    {
        //        magneticSusceptibility             1e-6;
        //        HdotgradH              U;
        //    }
electrostatic
            {
               charge              -6.0e-10;
                Efield              E;
            }

    }

    injectionModels
    {
        model1
        {
            type            coneNozzleInjection;
            SOI             0;
            massTotal       6.0e-4;
            parcelBasisType mass;
            injectionMethod disc;
            flowType        flowRateAndDischarge;
            outerDiameter   1.5e-3;
            innerDiameter   0;
            duration        1.8e-3;
            position        (0 0.0995 0);
            direction       (0 -1 0);
            parcelsPerSecond 2000000;
            flowRateProfile table
            (
                (0              100)
                (0.0018         100)
(0.05 0)
            );

            Cd              constant 0.9;

            thetaInner      constant 0.0;
            thetaOuter      constant 15.0;

            sizeDistribution
            {
                type        RosinRammler;

                RosinRammlerDistribution
                {
                    minValue        1e-06;
                    maxValue        0.00015;
```

```
                d               0.00015;
                n               3;
            }
        }
    }
}

dispersionModel none;

patchInteractionModel standardWallInteraction;

heatTransferModel none;

compositionModel singlePhaseMixture;

phaseChangeModel none;

surfaceFilmModel none;

atomizationModel none;

// breakupModel    ReitzDiwakar; // ReitzKHRT;

breakupModel none;

stochasticCollisionModel none;

radiation       off;

standardWallInteractionCoeffs
{
    type            rebound;
}

RanzMarshallCoeffs
{
    BirdCorrection  true;
}

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

liquidEvaporationBoilCoeffs
{
```

```
        enthalpyTransfer enthalpyDifference;

        activeLiquids    ( H2O );
    }

    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;
    }
}


cloudFunctions
{}


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

fvSchemes

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
```

```
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
    grad(ElPot)     Gauss linear;
}

divSchemes
{
    default         Gauss linear;

    div(phi,U)      Gauss upwind;
    div(phid,p)     Gauss upwind;
    div(phi,K)      Gauss linear;
    div(phi,k)      Gauss upwind;
    div(phi,epsilon) Gauss upwind;
    div(U)          Gauss linear;
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
    div(phi,Yi_h)   Gauss upwind;
}

laplacianSchemes
{
    default         Gauss linear orthogonal;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         orthogonal;
}
```

fvSolution

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{

   ElPot
    {
        solver           PCG;
        preconditioner   DIC;
        tolerance        1e-08;
        relTol           0;
    };

    rho
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-05;
        relTol          0.1;
    }

    rhoFinal
    {
        $rho;
        tolerance       1e-05;
        relTol          0;
    }

    "(U|k|epsilon)"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-06;
        relTol          0.1;
        minIter 1;
    }
```

```
    p
    {
        solver          GAMG;
        tolerance       0;
        relTol          0.1;
        smoother        GaussSeidel;
    }

    pFinal
    {
        $p;
        tolerance       1e-06;
        relTol          0;
    }

    "(U|k|epsilon)Final"
    {
        $U;
        tolerance       1e-06;
        relTol          0;
    }

    "(Yi|O2|N2|H2O)"
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-6;
        relTol          0;
    }

    h
    {
        $Yi;
        relTol          0.1;
    }

    hFinal
    {
        $Yi;
    }
}

PIMPLE
{
    transonic       no;
    nCorrectors     1;
    nOuterCorrectors 2;
    nNonOrthogonalCorrectors 0;
    momentumPredictor yes;
}

relaxationFactors
{
```

```
    equations
    {
        ".*Final"       0.2;
    }
}

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

controlDict

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     electrostaticSprayFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         0.0012;

deltaT          2.5e-07;

writeControl    adjustableRunTime;

writeInterval   0.5e-04;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

adjustTimeStep  yes;

maxCo           0.1;
```

```
runTimeModifiable yes;
```