CFD WITH OPENSOURCE SOFTWARE, ASSIGNMENT 3

OpenFOAM Tutorial

Implementation of Gasoline Properties in OpenFOAM Library

Author
CHEN HUANG

Reviewers
ALEXEY VDOVIN
HÅKAN NILSSON

Division of Combustion
Department of Applied Mechanics
Chalmers University of Technology
Jan. 2010

Contents

1. Introduction

The real gasoline properties, such as surface tension, dynamic viscosity, heat of vaporization, are important for the spray formation in the application of gasoline direct injection engines. This tutorial focuses on how to implement gasoline properties in the OpenFOAM library. Firstly, a description of OpenFOAM thermophysical functions and liquids libraries is given. Secondly, the implementation of the gasoline properties is described. Finally, a case of gasoline hollow cone spray in a constant volume is applied to test the implementation.

2. A description of thermophysical functions and liquids libraries

In order to explain how the liquid properties are calculated and the connections among a specific liquid class and the thermophysical function classes, a simplified collaboration diagram for iso-octane (IC8H18) is shown in Fig. 1. Note only NSRDSfunc0 (a thermophysical function that returns the density, heat capacity, enthalpy, and thermal conductivity of a liquid) and IC8H18 are shown in Fig. 1 in order to make the figure easy to read.

The class thermophysicalFunctions is an abstract class, which has a series of sub-classes like, NSRDSfunc0, NSRDSfunc1, NSRDSfunc2, NSRDSfunc4, NSRDSfunc5, NSRDSfunc6, NSRDSfunc7, and APIfunctions. This abstract class and one of its sub-classes NSRDSfunc0 are linked with a blue solid line with an arrow in Fig. 1. The scalar class is used by the NSRDSfunc0 class because it returns a scalar through a member function with a series of scalar variables such as, a_, b_, c_, d_, e_, and f_. In this way, NSRDSfunc0 and scalar are connected with a pink dashed line with an arrow with a_, b_, c_, d_, e_, and f_ above the arrow. Similarly, NSRDSfunc1, NSRDSfunc2, NSRDSfunc4, NSRDSfunc5, NSRDSfunc6, NSRDSfunc7, and APIfunctions are connected with the scalar class respectively.

The class liquid is an abstract class, which has a series of sub-classes like, IC8H18, C7H8, C7H16, and so on. The liquid class is linked with the scalar class because the liquid class returns a series of scalar values through its member functions, such as W(), Tc(), Pc(), Vc(), and so on. IC8H18 is a sub-class which is inherited from the liquid class. The IC8H18 class uses the NSRDSfunc0 class, and K_, h_ and cp_ are feedbacks to the IC8H18 class through a member function in the NSRDSfunc0 class. The values of K_, h_, and cp_ are different because the member function in the NSRDSfunc0 class is called differently. The relationship between the IC8H18 class and the rest of the thermophysical functions, such as NSRDSfunc1, NSRDSfunc2, NSRDSfunc4, NSRDSfunc5, NSRDSfunc6, NSRDSfunc7, and APIfunctions, can be explained in a similar way.

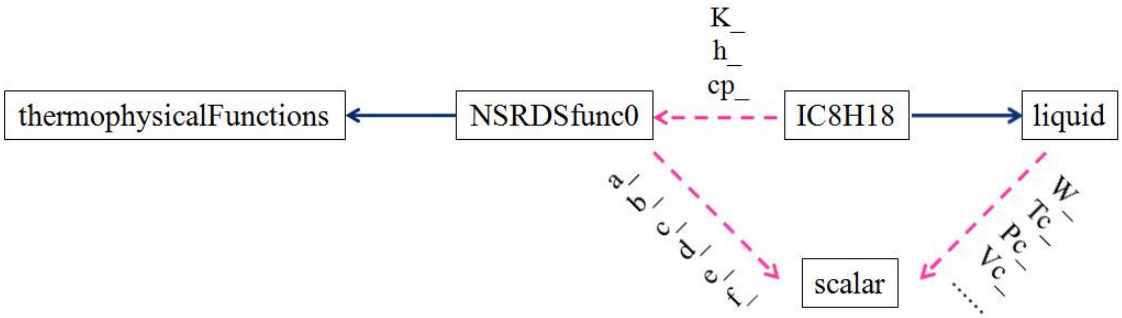

Figure 1 Simplified collaboration diagram for IC8H18

3. Implementation of gasoline properties

Note: This tutorial is based on OpenFOAM -1.6-x. The linux commands are in italic form, and the modifications are in underlined form. The ellipsis symbol means the contents which are not changed in the source code or the case files.

The fuel properties, the acronyms and the units (for OpenFOAM and KIVA) are shown in the appendix. In the rest of the tutorial the acronyms of the properties are used. The gasoline properties, such as pv, hl, h, mu, K, and sigma, are obtained from the KIVA fuel library in the form of a table. The tables of gasoline cp and cpg are deduced from the enthalpy table. The values of the gasoline W and Tc are also taken from KIVA fuel library. The gasoline rho is taken from reference [1] in the form of a table. The gasoline B is the same as for n-octane (C8H18). The gasoline D is the same as for n-heptane (C7H16). The gasoline surrogate is composed of iso-octane (IC8H18), toluene (C7H8) and n-heptane (C7H16) in a volume proportion of 55%:35%:10%. Thus, an averaged chemical formula C8H15 is used to represent gasoline when it is in gas phase. The rest of the gasoline properties, including mug, Kg, Pc, Vc, Zc, Tt, Pt, Tb, dipm, omega, and delta, are approximated using mixing rule [2] of the above mentioned ternary mixture, see equation 1.

$$Q_m = \sum_i y_i Q_i \qquad (1)$$

where, Q_m is a mixture parameter.

y_i is a liquid or vapor mole fraction.

Q_i is a property of a pure liquid or vapor.

3.1 Create new thermophysical functions for the gasoline properties

a. Make a directory for the thermophysical functions in the user working directory.
mkdir –p \
$WM_PROJECT_USER_DIR/src/thermophysicalModels/thermophysicalFunctions/NSRDSfunctions/

b. Go to the NSRDSfunctions/ directory within OpenFOAM installation, and copy the NSRDfunc.
cd $FOAM_SRC/thermophysicalModels/thermophysicalFunctions/NSRDSfunctions/
cp –r NSRDSfunc0 NSRDSfunc1 NSRDSfunc5 NSRDSfunc6 NSRDSfunc7 \
$WM_PROJECT_USER_DIR/
src/thermophysicalModels/thermophysicalFunctions/NSRDSfunctions/.

c. Go to the user working directory, and rename those thermophysical functions to the corresponding gasoline property functions. We only change NSRDSfunc5 to NSRDSfuncgRho as an example.
cd $WM_PROJECT_USER_DIR/ \
src/thermophysicalModels/thermophysicalFunctions/NSRDSfunctions/
mv NSRDSfunc5 NSRDSfuncgRho
cd NSRDSfuncgRho
*rename NSRDSfunc5 NSRDSfuncgRho **
sed –i s/"" NSRDSfunc5"/"" NSRDSfuncgRho"/g NSRDSfuncgRho.C
sed –i s/"" NSRDSfunc5"/"" NSRDSfuncgRho"/g NSRDSfuncgRho.H
rm NSRDSfuncgRho.dep
Similarly, we can change NSRDSfunc1 to NSRDSfuncgPv, NSRDSfunc6 to NSRDSfuncgHl, NSRDSfunc0 to NSRDSfuncgCp, NSRDSfunc0 to NSRDSfuncgH, NSRDSfunc7 to NSRDSfuncgCpg, NSRDSfunc1 to NSRDSfuncgMu, NSRDSfunc0 to NSRDSfuncgK, and NSRDSfunc6 to NSRDSfuncgSigma, respectively.

d. Modify the new NSRDSfunc. We only modify the NSRDSfuncgRho function as an example.
In NSRDSfuncgRho.H,

```
……
    // NSRDS function 105 coefficients
    scalar a_, b_, c_, d_;
    // create an scalar array with 56 members
    scalar rho[56];

public:

  //- Runtime type information
  TypeName("NSRDSfuncgRho");
……
```

```
    //- Construct from Istream
    NSRDSfuncgRho(Istream& is)
    :
      a_(readScalar(is)),
      b_(readScalar(is)),
      c_(readScalar(is)),
      d_(readScalar(is))
    {}

    /*- Construct from null & initialize the density (kg/m^3) from a table.
   Density, rho, ranges from 0 K to 550 K (critical temperature for gasoline) with an interval of 10 K.
Only the first and the final element of rho are shown here. The full list can be found at http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2009/ChenHuang/chenProjectFiles.tgz.  */
    NSRDSfuncgRho() {
        rho[0] = 9.53673e+02;
        rho[55] = 4.94729e+02;
        }

  // Member Functions

    //- Evaluate the function and return the result
    scalar f(scalar, scalar T) const
    {
    /* instead of returning NSRDSfunc5, we make interpolation in the density table rho[56]. */
          scalar rho_ = 0.0;
          for(int i=0; i<55; i++){
                if(T>=10*i && T<10*(i+1)){
                      rho_ = rho[i]+(T-10*i)*(rho[i+1]-rho[i])/10;
                      break;
                      }
                }
          return rho_;
    }
……
```

Similarly, we can modify the rest of the functions, including NSRDSfuncgPv, NSRDSfuncgHl, NSRDSfuncgCp, NSRDSfuncgH, NSRDSfuncgCpg, NSRDSfuncgMu, NSRDSfuncgK, and NSRDSfuncgSigma. Those implementations are provided in files at http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2009/ChenHuang/chenProjectFiles.tgz.

e. Create Make/files and Make/options, and compile the user thermophysical functions.
cd \
$WM_PROJECT_USER_DIR/src/thermophysicalModels/thermophysicalFunctions/NSRDSfunctions/
mkdir Make
touch Make/files
the contents of Make/files
NSRDSfuncgRho/NSRDSfuncgRho.C
NSRDSfuncgPv/NSRDSfuncgPv.C
NSRDSfuncgHl/NSRDSfuncgHl.C
NSRDSfuncgCp/NSRDSfuncgCp.C
NSRDSfuncgH/NSRDSfuncgH.C
NSRDSfuncgCpg/NSRDSfuncgCpg.C
NSRDSfuncgMu/NSRDSfuncgMu.C
NSRDSfuncgK/NSRDSfuncgK.C
NSRDSfuncgSigma/NSRDSfuncgSigma.C

LIB = $(FOAM_USER_LIBBIN)/libmyThermophysicalFunctions

touch Make/options
the contents of Make/options
EXE_INC = \
    -I$(LIB_SRC)/thermophysicalModels/thermophysicalFunctions/lnInclude

LIB_LIBS = \
    -lthermophysicalFunctions

Compile the library

wmake libso

A new thermophysical function library, libmyThermophysicalFunctions, will appear in the directory $FOAM_USER_LIBBIN.

3.2 Create a new liquid class gasoline

a. Make a directory for gasoline liquid in the user working directory.
mkdir –p $WM_PROJECT_USER_DIR/src/thermophysicalModels/liquids /
b. Go to that directory, and copy the IC8H18/ directory.

cd $WM_PROJECT_USER_DIR/src/thermophysicalModels/thermophysicalFunctions/liquids

cp –r $FOAM_SRC/ thermophysicalModels/ liquids/IC8H18 .

c. Rename IC8H18 to gasoline.

mv IC8H18 gasoline

cd gasoline

*rename IC8H18 gasoline **

sed –i s/"IC8H18"/"gasoline"/g gasoline.C

sed –i s/"IC8H18"/"gasoline"/g gasoline.H

sed –i s/"IC8H18"/"gasoline"/g gasolineI.H

rm gasoline.dep

d. Modify the gasoline class.

The file gasoline.H is modified as follows,

……

```
#include "NSRDSfunc5.H"
#include "NSRDSfunc6.H"
#include "NSRDSfunc7.H"
#include "NSRDSfunc14.H"
#include "APIdiffCoefFunc.H"
// include the gasoline property functions
#include "NSRDSfuncgRho.H"
#include "NSRDSfuncgPv.H"
#include "NSRDSfuncgHl.H"
#include "NSRDSfuncgCp.H"
#include "NSRDSfuncgH.H"
#include "NSRDSfuncgCpg.H"
#include "NSRDSfuncgMu.H"
#include "NSRDSfuncgK.H"
#include "NSRDSfuncgSigma.H"

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

namespace Foam
{

/*---------------------------------------------------------------------------*\
                    Class gasoline Declaration
\*---------------------------------------------------------------------------*/

class gasoline
:
   public liquid
```

```
{
    // Private data

        NSRDSfuncgRho rho_;
        NSRDSfuncgPv pv_;
        NSRDSfuncgHl hl_;
        NSRDSfuncgCp cp_;
        NSRDSfuncgH h_;
        NSRDSfuncgCpg cpg_;
        NSRDSfunc4 B_;
        NSRDSfuncgMu mu_;
        NSRDSfunc2 mug_;
        NSRDSfuncgK K_;
        NSRDSfunc2 Kg_;
        NSRDSfuncgSigma sigma_;
        APIdiffCoefFunc D_;

public:

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


    // Constructors

        //- Construct null
        gasoline();

        //- Construct from components
        gasoline
        (
            const liquid& l,
            const NSRDSfuncgRho& density,
            const NSRDSfuncgPv& vapourPressure,
            const NSRDSfuncgHl& heatOfVapourisation,
            const NSRDSfuncgCp& heatCapacity,
            const NSRDSfuncgH& enthalpy,
            const NSRDSfuncgCpg& idealGasHeatCapacity,
            const NSRDSfunc4& secondVirialCoeff,
            const NSRDSfuncgMu& dynamicViscosity,
            const NSRDSfunc2& vapourDynamicViscosity,
            const NSRDSfuncgK& thermalConductivity,
```

```
        const NSRDSfunc2& vapourThermalConductivity,
        const NSRDSfuncgSigma& surfaceTension,
        const APIdiffCoefFunc& vapourDiffussivity
    );

    //- Construct from Istream
    gasoline(Istream& is);
……
```

The file gasoline.C is modified as follows,

```
// * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * //

Foam::gasoline::gasoline()
:
/* the gasoline liquid properties are approximated based on the gasoline constituent components in mole fraction except for mole weight and critical temperature */
  liquid(113.228, 548.00, 3.12419e+6, 0.411, 0.265, 171.80, 4.05773e-2, 376.30, 0.0, 0.2941, 1.5669e+4),
  rho_(),
  pv_(),
  hl_(),
  cp_(),
  // NN: enthalpy, h_, is not used in the sprayModel.
  // For consistency, the enthalpy is derived from hlat and hl.
  // It is, however, convenient to have it available.
  h_(),
  cpg_(),
  // B is the same as n-octane C8H18;
  B_(0.00239777293379205, -2.81394717721109, -585042.589139551, -1.11265768486663e+18, 1.40968738783693e+20),
  mu_(),
  //Mug is approximated using mixing rule in equation 1.
  mug_(7.77735e-08, 8.30817e-01, 4.83952e+01, 0.0),
  K_(),
  //Kg is approximated using mixing rule in equation 1.
  Kg_(-6.98476e-03, 1.20363e+00, -3.00945e+02, -2.40050e+05),
  sigma_(),
  D_(147.18, 20.1, 114.231, 28) // NN: Same as nHeptane
{}


Foam::gasoline::gasoline
```

```
(
    const liquid& l,
    const NSRDSfuncgRho& density,
    const NSRDSfuncgPv& vapourPressure,
    const NSRDSfuncgHl& heatOfVapourisation,
    const NSRDSfuncgCp& heatCapacity,
    const NSRDSfuncgH& enthalpy,
    const NSRDSfuncgCpg& idealGasHeatCapacity,
    const NSRDSfunc4& secondVirialCoeff,
    const NSRDSfuncgMu& dynamicViscosity,
    const NSRDSfunc2& vapourDynamicViscosity,
    const NSRDSfuncgK& thermalConductivity,
    const NSRDSfunc2& vapourThermalConductivity,
    const NSRDSfuncgSigma& surfaceTension,
    const APIdiffCoefFunc& vapourDiffussivity
)
:
    liquid(l),
    rho_(density),
    pv_(vapourPressure),
    hl_(heatOfVapourisation),
    cp_(heatCapacity),
    h_(enthalpy),
    cpg_(idealGasHeatCapacity),
    B_(secondVirialCoeff),
    mu_(dynamicViscosity),
    mug_(vapourDynamicViscosity),
    K_(thermalConductivity),
    Kg_(vapourThermalConductivity),
    sigma_(surfaceTension),
    D_(vapourDiffussivity)
{}
……
```

e. Create Make/files and Make/options, and compile the new liquid class gasoline.
   cd $WM_PROJECT_USER_DIR/src/thermophysicalModels/liquids/gasoline/
   mkdir Make
   touch Make/files
   the contents of Make/files
   gasoline.C

   LIB = $(FOAM_USER_LIBBIN)/libmyLiquids

touch Make/options
the contents of Make/options
EXE_INC = \
-I$(LIB_SRC)/thermophysicalModels/liquids/lnInclude \
-I$(LIB_SRC)/thermophysicalModels/thermophysicalFunctions/lnInclude \
-I$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/thermophysicalFunctions/NSRDSfunctions/lnInclude \

LIB_LIBS = \
-lliquids \
-lthermophysicalFunctions \
-L$(WM_PROJECT_USER_DIR)/lib/$(WM_OPTIONS) \
-lmyThermophysicalFunctions

Compile the library,

wmake libso

A new liquid library, libmyLiquids, will appear in the directory $FOAM_USER_LIBBIN.

4. A test of the implementation through a case study

A gasoline liquid spray into a constant volume will be used to test the implementation of the gasoline properties. Before setting up the case, the source code of the reitzDiwakar breakup model which is related to surface tension is changed to help us verify the implementation. The value of the newly implemented surface tension is added to the log file.

4.1 Modify the reitzDiwakar breakup model

a. Make a directory for a new reitzDiwakar breakup model.
mkdir –p \
$WM_PROJECT_USER_DIR/src/lagrangian/dieselSpray/spraySubModels/breakupModel/
b. Go to the user breakupModel/ directory, and copy the original reitzDiwakar/ directory.
cd \
$WM_PROJECT_USER_DIR/src/lagrangian/dieselSpray/spraySubModels/breakupModel/
cp –r \
$FOAM_SRC/lagrangian/dieselSpray/spraySubModels/breakupModel/reitzDiwakar/ .
c. Rename reitzDiwakar as myReitzDiwakar.
mv reitzDiwakar myReitzDiwakar

cd myReitzDiwakar
*rename reitzDiwakar myReitzDiwakar **
sed –i s/"reitzDiwakar"/"myReitzDiwakar"/g myReitzDiwakar.C
sed –i s/"reitzDiwakar"/"myReitzDiwakar"/g myReitzDiwakar.H
rm myReitzDiwakar.dep

d. Modify myReitzDiwakar.C as follows,

……

```
 // ideal gas law to evaluate density
   scalar rhoAverage = pressure/R/Taverage;
   scalar nuAverage = muAverage/rhoAverage;
   scalar sigma = fuels.sigma(pressure, p.T(), p.X());
 // output the temperature and corresponding surface tension
   Info<< "T = " << p.T() << endl;
   Info<< "sigma = " << sigma << endl;
   // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
   //    The We and Re numbers are to be evaluated using the 1/3 rule.
   // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

……

e. Create Make/files and Make/options, and compile myReitzDiwakar breakup model.

cd \
$WM_PROJECT_USER_DIR/src/lagrangian/dieselSpray/spraySubModels/breakupModel/reitzDiwakar/
mkdir Make
touch Make/files
the contents of Make/files
myReitzDiwakar.C

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

touch Make/options
the contents of Make/options

```
EXE_INC = \
   -I$(LIB_SRC)/finiteVolume/lnInclude \
   -I$(LIB_SRC)/lagrangian/basic/lnInclude \
   -I$(LIB_SRC)/lagrangian/dieselSpray/lnInclude \
   -I$(LIB_SRC)/turbulenceModels \
   -I$(LIB_SRC)/turbulenceModels/compressible/turbulenceModel \
   -I$(LIB_SRC)/turbulenceModels/compressible/RAS/lnInclude \

 -I$(LIB_SRC)/turbulenceModels/LES/LESdeltas/lnInclude \
   -I$(LIB_SRC)/turbulenceModels/compressible/LES/lnInclude \
   -I$(LIB_SRC)/thermophysicalModels/basic/lnInclude \
```

```
    -I$(LIB_SRC)/thermophysicalModels/liquids/lnInclude \
    -I$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/liquids/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/liquidMixture/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/thermophysicalFunctions/lnInclude \
    -
I$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/thermophysicalFunctions/NSR
DSfunctions/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/specie/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/reactionThermo/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/pdfs/lnInclude

LIB_LIBS = \
    -lfiniteVolume \
    -llagrangian \
    -ldieselSpray \
    -lcompressibleRASModels \
    -lcompressibleLESModels \
    -lLESdeltas \
    -lliquids \
    -lliquidMixture \
    -lthermophysicalFunctions \
    -lspecie \
    -lpdf \
    -L$(WM_PROJECT_USER_DIR)/lib/$(WM_OPTIONS) \
    -lmyLiquids
```

Compile the library,

wmake libso

A new breakup library, libmyReitzDiwakar, will appear in the directory $FOAM_USER_LIBBIN.

4.2 Setup a case for gasoline hollow cone spray in a constant volume

a. Go to the user run directory, and copy the aachenBomb case to the user run directory
   run
   cp –r $FOAM_TUTORIALS/combustion/dieselFoam/aachenBomb .
b. Modify the chemkin/ directory. C8H15 represents gaseous gasoline due to the composition of the constituent components mentioned in section three. Since the combustion is not considered in this case, the balance of atoms is ensured whereas the

reaction rate constants are kept the same when we switch the fuel from C7H16 to C8H15. The chem.inp is modified as follows,

```
ELEMENTS
 H  O   C  N  AR
END
SPECIE
C8H15 O2 N2 CO2 H2O
END
REACTIONS
 C8H15 + 11.75O2          => 8CO2 + 7.5H2O       5.00E+8  0.0   15780.0! 1
          FORD    / C8H15          0.25 /
          FORD    / O2 1.5 /
END
```

Any chemical species that appears in a problem must have thermodynamic data associated with it. Thus, the coefficients for thermodynamic properties (heat capacity, enthalpy and entropy) of C8H15 are added in the end of therm.dat file as follows.

```
C8H15          P 4/85C  8.H 15.   0.   0.G   200.000  5000.000 1396.0      1
 2.15002114e+01   3.36729730e-02-1.16006708e-05   1.82223584e-09-1.06962828e-13
2
-2.59374406e+04-9.22017472e+01-6.34105767e-01   6.77277031e-02   5.91125179e-06
3
-5.53067539e-08 2.69930010e-11-1.77199967e+04 3.02314987e+01              4
```

c. Modify the constant/ directory.

First, modify the geometry in polyMesh/blockMeshDict file since we are using our own geometry and mesh,

```
convertToMeters 0.001;

vertices
(
   (20 -20 0)
   (20 20 0)
   (-20 20 0)
   (-20 -20 0)
   (60.10 -60.10 0)
   (60.10 60.10 0)
   (-60.10 60.10 0)
   (-60.10 -60.10 0)
   (20 -20 -205)
   (20 20 -205)
   (-20 20 -205)
```

```
    (-20 -20 -205)
    (60.10 -60.10 -205)
    (60.10 60.10 -205)
    (-60.10 60.10 -205)
    (-60.10 -60.10 -205)
);

blocks
(
    hex (0 3 2 1 8 11 10 9) (25 25 50) simpleGrading (1 1 2)
    hex (0 1 5 4 8 9 13 12) (25 20 50) simpleGrading (1 2 2)
    hex (1 2 6 5 9 10 14 13) (25 20 50) simpleGrading (1 2 2)
    hex (2 3 7 6 10 11 15 14) (25 20 50) simpleGrading (1 2 2)
    hex (3 0 4 7 11 8 12 15) (25 20 50) simpleGrading (1 2 2)
);

edges
(
    arc 4 5 (85 0 0)
    arc 5 6 (0 85 0)
    arc 6 7 (-85 0 0)
    arc 7 4 (0 -85 0)
    arc 12 13 (85 0 -205)
    arc 13 14 (0 85 -205)
    arc 14 15 (-85 0 -205)
    arc 15 12 (0 -85 -205)
);

patches
(
    wall cylinder
    (
        (4 5 13 12)
        (5 6 14 13)
        (6 7 15 14)
        (7 4 12 15)
    )
    wall cellhead
    (
        (0 3 2 1)
        (0 1 5 4)
        (1 2 6 5)
```

```
    (2 3 7 6)
    (0 4 7 3)
  )

  wall cellbottom
  (
    (8 11 10 9)
    (8 9 13 12)
    (9 10 14 13)
    (10 11 15 14)
    (8 12 15 11)
  )
);

mergePatchPairs
(
);
```

Modify injectorProperties as follows,

```
    injectorType      unitInjector;

     unitInjectorProps
     {
        position       (0 0 -0.002);   // the position of injector, m
        direction      (0 0 -1);        //the direction of injection
        diameter       0.00046;     //the diameter of injector, m
        Cd   0.9;   //the discharge coefficient of the injector, usually ranges from 0.6 – 0.9
        mass           1.369e-05;    // the total injected fuel mass, kg
        nParcels       5000;         //the number of parcels

        X
        (
          1.0
        );

        massFlowRateProfile
        (
/*the actual mass flow rate will be normalized based on the total injected mass and this
profile. Here a trapezoid shaped mass flow rate profile is given. */
          (0 0.1)
```

```
        (0.0002 1)
        (0.0004 1)
        (0.0006 0.1)
    );

    temperatureProfile
    (
/* the temperature of fuel, K. Here the fuel temperature is assumed to be 243 K during the injection process.  */
        (0.0    243.0)
        (0.0006  243.0)
    );
……
```

Note the probability density function of the OpenFOAM Rosin Rammler distribution (see equation 2) is different from that of the standard Rosin Rammler distribution [3] (see equation 3).

$$pdf = \left(\frac{x}{d}\right)^n e^{-\left(\frac{x}{d}\right)^n} \qquad (2)$$

$$pdf = \frac{n}{d}\left(\frac{x}{d}\right)^{n-1} e^{-\left(\frac{x}{d}\right)^n} \qquad (3)$$

Here, we use the OpenFOASM Rosin Rammler distribution to initialize the droplet size near the nozzle. For the hollow cone spray, the outer cone angle and inner cone angle is 90 and 80 degree, respectively. Modify sprayProperties as follows,

```
……
breakupModel    myReitzDiwakar;
……
myReitzDiwakarCoeffs
{
  Cbag          6;
……
hollowConeInjectorCoeffs
{
  dropletPDF
  {
    pdfType       RosinRammler;
    RosinRammlerPDF
    {
      minValue      1e-06;    // the minimum droplet diameter, m
      maxValue      8.00e-5;  // the maximum droplet diameter , m
      d          ( 3.00e-5 );   // the droplet diameter, m, which has the largest probability
      n          ( 3 );         // the exponential factor
```

```
        }

        exponentialPDF
        {
            minValue       0.0001;
            maxValue       0.001;
            lambda         ( 10000 );
        }
    }

    innerConeAngle  ( 80 );
    outerConeAngle  ( 90 );
}
……
```

Modify thermophysicalProperties as follows,

```
thermoType      hPsiMixtureThermo<reactingMixture<gasThermoPhysics>>;

CHEMKINFile     "$FOAM_CASE/chemkin/chem.inp";

CHEMKINThermoFile "$FOAM_CASE/chemkin/therm.dat";

inertSpecie     N2;

liquidComponents ( C8H15 );

liquidProperties
{
    C8H15         gasoline defaultCoeffs;
}
```

d. Modify system/ directory
   Since the new breakup model and the new liquid libraries are linked dynamically, we only need to add the following at the end of controlDict,
   libs ("libmyReitzDiwakar.so");
e. Modify 0/ directory according to the case files located at the following address in order to make sure of the correct initial and boundary setup.
   http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2009/ChenHuang/chenProjectFiles.tgz

4.3 Run the gasoline hollow cone spray case

run

blockMesh –case aachenBomb

dieselFoam –case aachenBomb >& aachenBomb/log &

Check the log file for the information about surface tension. For example, we got,

T = 320.326

sigma = 0.0163615

which is exactly the same as we have implemented. Figure 2 shows the droplet distribution of gasoline hollow cone spray in a constant volume at different time steps.

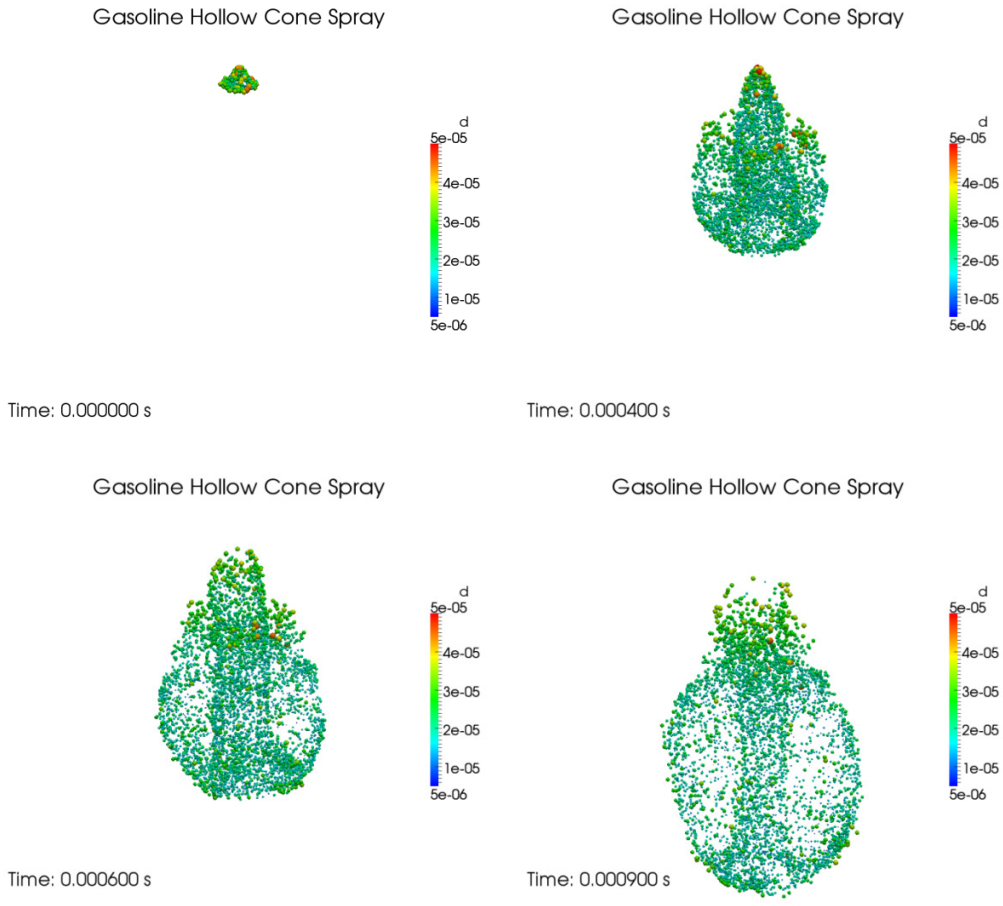

Figure 2 Droplet distribution of gasoline hollow cone spray in a constant volume

Reference

[1] Joseph A. Schetz, Allen E. Fuhs, Handbook of Fluid Dynamics and Fluid Machinery, volume one Fundamentals of Fluid Dynamics. p 156-157.

[2] Robert C. Reid, John M. Prausnitz, Bruce E. Poling, The Properties of Gases & Liquids Fourth Edition. p 75.

[3] Crowe C., Sommerfeld M., Tsuji Y., Multiphase Flows with Droplets and Particles. CRC Press, Boca Raton, 1998, p45.

Appendix

A summary of the fuel properties, the acronyms and the units

Property	Acronym	Unit (OpenFOAM)	Unit (KIVA)
Molecular weight	W	kg/kmol	kg/kmol
Critical temperature	Tc	K	K
Critical pressure	Pc	Pa	
Critical volume	Vc	m^3/mol	
Critical compressibility factor	Zc	-	
Triple point temperature	Tt	K	
Triple point pressure	Pt	Pa	
Normal boiling temperature	Tb	K	
Dipole moment	dipm	-	
Pitzer's ascentric factor	omega	-	
Solubility parameter	delta	(J/m^3)^0.5	
Liquid density	rho	kg/m^3	
Vapour pressure	pv	Pa	dynes/sq. cm.
Heat of vapourisation	hl	J/kg	ergs/gm
Liquid heat capacity	cp	J/(kg K)	
Liquid enthalpy	h	J/kg	kcal/mole
Ideal gas heat capacity	cpg	J/(kg K)	
Second virial coefficient	B	m^3/kg	
Liquid viscosity	mu	Pa s	gm/(cm sec)
Vapour viscosity	mug	Pa s	
Liquid thermo conductivity	K	W/ (m K)	ergs/(s cm degree K)
Vapour thermo conductivity	Kg	W/ (m K)	
Surface tension	sigma	N/m	dynes/cm
Vapour diffusivity	D	m^2/s	