Chalmers University of Technology

CFD with OpenFOAM software

Lagrangian particle tracking

Prepared by: Jelena Andric

Gothenburg, 200

1. About multiphase flows

The equations for motion and thermal properties of single-phase flows are well accepted (Navier-Stokes equations) and closed form solutions exist for specific cases, while major difficulty is the modeling and quantification of turbulence and its influence on mass, momentum and energy transfer. Computational Fluid Dynamics has already a long history and different commercially available CFD-tools for this type of the flow.
On the other hand, the correct formulation of the governing equations for multiphase flows is still subject to debate. Interaction between different phases makes these flows complicated and very difficult to describe theoretically.
Multiphase flows are of great importance, since they can occur even more frequently than single phase flows and are present in various forms in industrial practice, as for example transient flows with a transition from pure liquid to a vapor flow as a result of external heating, separated flows and dispersed two-phase flows where one phase is present in the form of particles, droplets, or bubbles in a continuous carrier phase (i.e. gas or liquid).

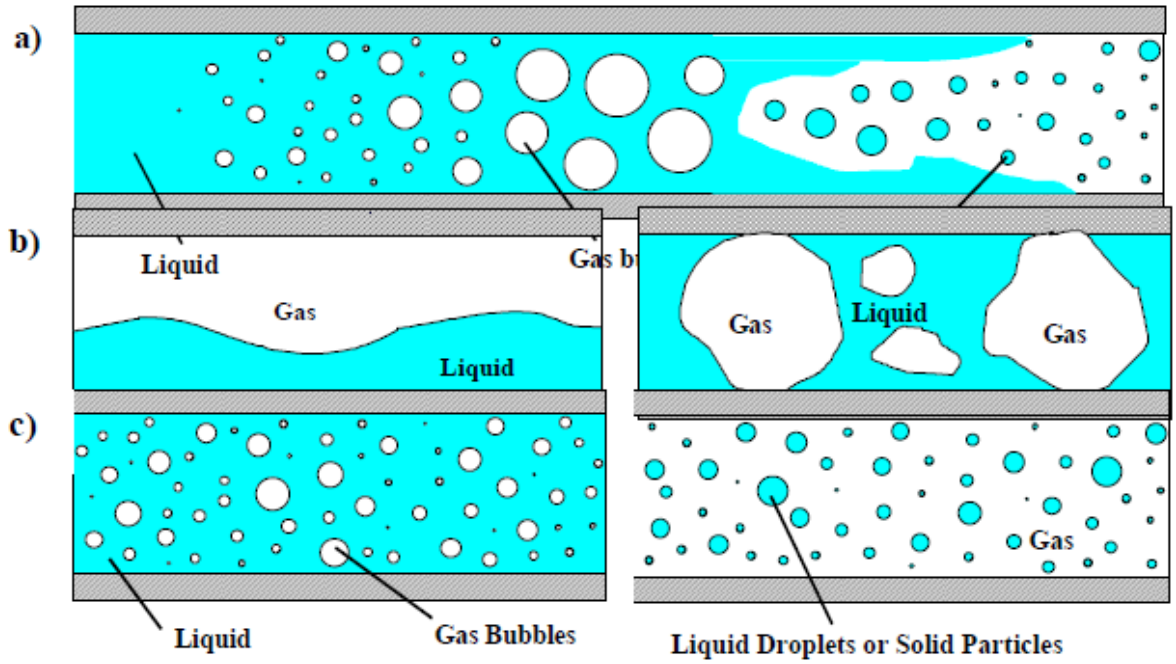

Figure1:Different regimes of two-phase flows, a)transient two-phase flow, b)separated two-phase flow, c)dispersed two-phase flow. [1]

For instance dispersed two-phase flows are encountered in numerous technical and industrial processes and may be classified in terms of the different phases being present:

- Gas-solid flows

- Liquid-solid flows
- Gas-droplet flows
- Liquid-droplet flows
- Liquid-droplet flows

Dispersed two-phase flows are usually separated in two flow regimes:

1. Dilute dispersed systems – the spacing between particles is large, a direct interaction is rare and fluid dynamic forces are governing particle transport.
2. Dense dispersed systems- inter –particle spacing is low (smaller than about 10 particle diameters) and the transport of particles is dominated by collisions between them.

They can be characterized by the volume fraction of the dispersed phase which represents the volume occupied by the particles in a unit volume.

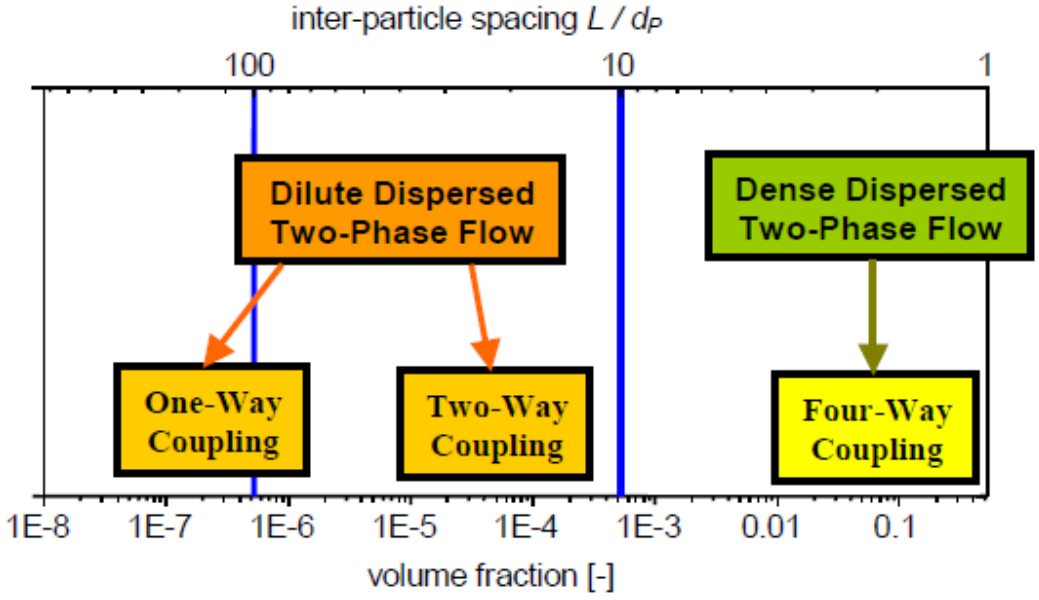

Figure2: Regimes of dispersed two-phase flows as a function of particle volume fraction. [1]

A classification of dispersed two-phase flows with respect to the importance of interaction mechanism was provided by Elghobashi (1994). Generally it is a distinction between dilute and dense two-phase flows as can be seen from the figure above. A two-phase system may be regarded as dilute for volume fractions up to $\alpha_P > 10^{-3}$ (i.e. $L/d_P \approx 8$.In this flow regime the influence of the particle on the fluid flow may be neglected for $\alpha_P < 10^{-6}$ .This is referred to as one-way coupling. For higher volume fractions it is necessary to account for the influence of the particles on the fluid flow – two-way coupling .In the dense regime inter-particle interactions such as collisions and

fluid dynamic interactions between particles become important. This is so-called four way coupling.

1.1Forces acting on particles

The motion of particles in fluids is described in a Lagrangian way by solving a set of ordinary differential equations along the trajectory in order to calculate the change of particle location and the linear and angular components of particle velocity. The relevant forces acting on the particle need to be taken into account. Hence, considering spherical particles the differential equations for calculating the particle location and velocity are given by Newtonian second law:

$$\frac{dx_P}{dt} = u_P \ ,$$
$$m_P \frac{d\,u_P}{dt} = \sum F_i \ ,$$
$$I_P \frac{d\,\omega_P}{dt} = T \ .$$

where $m_P = \rho_P \, d_P^3 \pi / 6$ is a particle mass, $I_P = 0.1 m_P \, d_P^2$ is a moment of inertia for a sphere, F_i represents the relevant forces acting on the particle, ω_P is the angular velocity of a particle and T is the torque acting on a rotating particle due to the viscous interaction with the fluid.
Analytical solutions for the different forces are available for small Reynolds numbers (Stokes flow).An extension to higher Reynolds numbers is usually obtained by including a coefficient C in front of the force, where C is based on empirical correlations derived from experiments or direct numerical simulations. In most fluid-particle systems the drag force is dominating the particle motion. Its extension to higher particle Reynolds number is based on the introduction of a drag coefficient C_D which is defined as:

$$C_D = \frac{F_D}{\frac{\rho_F}{2}\left(u_F - u_P\right)^2 A_P} \ ,$$

where $A_P = \frac{d_P^2 \pi}{4}$ is the cross-section of a spherical particle. The drag force is expressed by:

$$F_D = \frac{3}{4} \frac{\rho_F \, m_P}{\rho_P \, d_P} \, C_D \left(u_F - u_P\right) \left| u_F - u_P \right| .$$

The drag coefficient is given as a function of particle Reynolds number which is defined as the ratio of inertial force to friction force:

$$\mathrm{Re}_P = \frac{\rho_F \, d_P \left| u_F - u_P \right|}{\mu_F} .$$

The dependence of the drag coefficient of a spherical particle on the Reynolds number is shown in figure bellow and it is based on numerous experimental investigations (Schlichting 1965). From this dependence several regimes associated with flow characteristics can be identified as can presented in figure bellow:

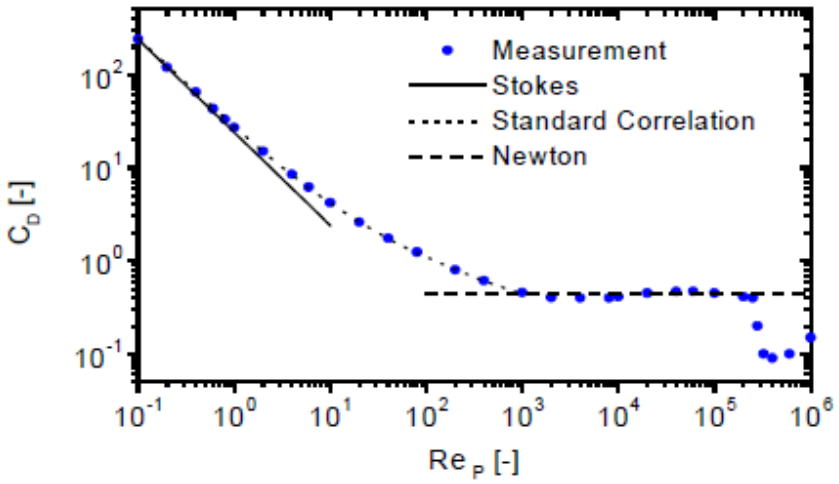

Figure3: Drag coefficient as a function of particle Reynolds number. Comparison of experimental data with the correlation for the different regimes. [1]

For small Reynolds number (i.e. $\mathrm{Re}_P < 0.5$) the viscous effect dominate and no separation occurs. The analytic solution for drag is possible as proposed by Stokes (1851):

$$C_D = \frac{24}{\mathrm{Re}_P} .$$

This regime is often referred to as Stokes flow.
For transition region (i.e. $0.5 < \mathrm{Re}_P < 1000$) numerous correlations have been proposed. The frequently used is the one proposed by Schiller and Neumann which fits well the data up to $\mathrm{Re}_P = 1000$.

$$C_D = \frac{24}{\mathrm{Re}_P}\left(1 + 0.15\ \mathrm{Re}_P^{0.687}\right) = \frac{24}{\mathrm{Re}_P} f_D .$$

Above $\mathrm{Re}_P \approx 1000$ the flow is fully turbulent and the drag coefficient remains almost constant up to the critical Reynolds number. This regime is often referred to as Newton-regime with:

$$C_D \approx 0.44 .$$

At critical Reynolds number ($\mathrm{Re}_{crit} \approx 2.5 \cdot 10^5$) there is a drastic decrease in drag coefficient due to transition from a laminar to turbulent boundary layer around the particle.

A generally more accurate sub-critical expression is given by Clift and Gauvin as:

$$C_D = \left[\frac{24}{\mathrm{Re}_P} \left(1 + 0.15\,\mathrm{Re}_P^{0.687} \right) \right] + \frac{0.42}{1 + \frac{42{,}500}{\left(\mathrm{Re}_P \right)^{1.16}}} \qquad for\ \mathrm{Re}_P < 2 \cdot 10^5$$

where the term in square brackets is Schiller-Naumman expression mentioned above. This can be written in terms of the Stokes correction as:

$$f = f_{shape} \left[1 + 0.15 \left(\mathrm{Re}_P \right)^{0.687} \right] \quad for\ \mathrm{Re}_P < 800 \ .$$

This is perhaps, the most commonly used drag correction expression in multiphase flows since many particles are constrained to Re_P values in this range.

1.2 Drag of non-spherical solid particles

Non-spherical solid bodies can be classified as either regularly shaped particles (ellipsoids, cones, disks) or irregularly-shaped particles (non-symmetric rough surfaces).Circular cylinders belong to the class of regularly-shaped particles. For cylindrical particles it is straightforward to define an aspect ratio E_{cyl} in the form $E_{cyl} = L_{cyl} / D_{cyl}$ (L-length, D-diameter).Depending on the relationship between these the limiting cases, i.e. disks ($E_{cyl} << 1$) and needles $E_{cyl} >> 1$can be identified.

Regularly shaped non-spheroidal particles do not typically have analytical solution for the drag even in the creeping flow limit. Firstly their shape and corresponding drag corrections may be approximated as ellipsoids by determining an effective aspect ratio. This approach is good for cylinders since their shape is quite similar to that of spheroid. Additional accuracy may be obtained introducing the shape factor defined as:

$$f_{shape} \equiv \frac{C_{D,shape}}{C_{D,sphere}} \quad f_{shape} \equiv \left. \frac{C_{D,shape}}{C_{D,sphere}} \right|_{\mathrm{Re}_P << 1 \ \&\ const.vol.} .$$

To estimate the shape factor for non-spheroidal regularly-shaped particles, it is common to consider two dimensionless area parameters: the surface and the projected area ratios. Each of those can be normalized by the surface area of a sphere which has the same volume:

$$A^*_{surf} \equiv \frac{A_{surf}}{\pi d^2} \ , \ A^*_{proj} \equiv \frac{A_{proj}}{1/4\ \pi d^2}$$

The inverse of the surface are ratio is more commonly defined as the “sphericity ratio” or the “shape factor”. For a cylinder with an aspect ratio E_{cyl}, the surface area ratio and equivalent volume diameter are given as:

$$E_{cyl} \equiv \frac{L_{cyl}}{d_{cyl}}\ ,\ A^*_{surf} = \frac{2E_{cyl}+1}{\left(18E^2_{cyl}\right)^{1/3}}\ ,\ d = d_{cyl}\left(\frac{3E_{cyl}}{2}\right)^{1/3}.$$

The projected area ratio will depend on the orientation of the particle as well as its shape. For example, a long cylinder will have $A^*_{proj} > 1$ if it falls broadside, but $A^*_{proj} < 1$ if it falls vertically along its axis.
Generally it is expected that lager values of A^*_{proj} or A^*_{surf} would correspond to larger drag values, and indeed this is the case.
The following correlation of these two area ratios was suggested by Leith for the Stokes shape correction factor:

$$f_{shape} = \frac{1}{3}\sqrt{A^*_{proj}} + \frac{2}{3}\sqrt{A^*_{surf}} \quad \text{for } \mathrm{Re}_P << 1.$$

The relation is based on the fact that one-third of the drag of the sphere is form drag (related to the projected area) while two thirds is friction drag (related to the surface *****area) and that the form and friction drags are proportional to the particle diameter. It holds for non-spherical particles with small deviations from the sphere, so it doesn’t hold for very high or very low aspect ratios. It gives reasonable results for many well defined shapes with moderate aspect ratios. If particle has surface area ratio close to that of a spheroid the following relationships stand for a given aspect ratio:

$$A^*_{surf} = \frac{E^{-2/3}}{2} + \frac{E^{4/3}}{4\sqrt{1-E^2}}\ln\left(\frac{1+\sqrt{1-E^2}}{1-\sqrt{1-E^2}}\right) \quad \textit{for } E < 1\,,$$

$$A^*_{surf} = \frac{1}{2E^{2/3}} + \frac{E^{1/3}}{2\sqrt{1-E^{-2}}}\sin^{-1}\left(\sqrt{1-E^{-2}}\right) \quad \textit{for } E \geq 1.$$

1.2.1 Non-spherical particles in the Newtonian-drag regime

It is helpful to consider the drag at high Reynolds number before proceeding to intermediate vales. Non-spherical particles tend to have drag coefficients that are approximately independent of Reynolds number ($10^4 < \mathrm{Re}_P < 10^5$), so that an approximately constant critical drag coefficient can be defined in Newton-drag regime.

Similarly to the definition of f_{shape} this drag coefficient can be normalized by that of a sphere with the same volume:

$$C_{shape} = \left. \frac{C_{D,shape,crit}}{C_{D,sphere,crit}} \right|_{cont.vol.}$$

The approximate average for a sphere is:

$$C_{D,sphere,crit} = 0.42 \quad for \quad 10^4 < \mathrm{Re}_P < 10^5 .$$

Drag at high Reynolds number is normally defined by projected area, it is difficult to determine the A^*_{proj} for some particles, since the trajectories will generally include secondary motion, so that they are not always falling in a broadside orientation (vs. area associated with volumetric diameter). For cylinders secondary motion was found to be important at extreme aspect ratios. In general cylinders and prolate ellipsoids can be approximately represented for a wide variety of density ratios by the following expression:

$$\left\langle C_{shape} \right\rangle \approx 1 + 0.7\sqrt{A^*_{surf} - 1} + 2.4\left(A^*_{surf} - 1\right) \quad for \; E > 1 .$$

1.2.2 Non-spherical particles at intermediate Reynolds numbers

There are many forms of correlations trying to predict drag coefficient of non-spherical particles at intermediate Re_P. The most successful approaches are those which use a combination of the Stokes drag correction and the Newton-drag correction. These approaches assume that the dependence from $\mathrm{Re}_P << 1$ to $\mathrm{Re}_{P,crit}$ is similar for all particle shapes and the difference is simply correction at two extremes, given by f_{shape} and $C_{D,shape}$. The dependency comes out from dimensional analysis and can be expressed as $C^*_D = f\left(\mathrm{Re}^*_P\right)$ by normalizing the drag coefficient and the Reynolds number as:

$$C^*_D = \frac{C_D}{C_{shape}}, \quad \mathrm{Re}^*_P = \frac{C_{shape}\,\mathrm{Re}_P}{f_{shape}}$$

Use of dimensionless Clift-Gauvin expression yields:

$$C^*_D = \frac{24}{\mathrm{Re}^*_P}\left[1 + 0.15\left(\mathrm{Re}^*_P\right)^{0.687}\right] + \frac{0.42}{1 + \dfrac{42{,}500}{\left(\mathrm{Re}^*_P\right)^{1.16}}} \quad for \approx circular \; C/S \quad .$$

This gives good correlation for particles for a wide range of Reynolds number whose relative cross-section (C/S) is approximately circular, e.g. spheres and cylinders. For moderate particle Reynolds numbers, a normalized Schiller-Neumann expression may be similarly defined:

$$f = f_{shape}\left[1 + 0.15\left(\mathrm{Re}_P^*\right)^{0.687}\right] \quad for\ \mathrm{Re}_P^* < 800\ \&\approx circular\ C/S$$

2. Implementation in OpenFOAM

Two classes are to be descried. Class *solidParticle* and *solidParticleCloud.*

2.1Class solidParticle

- Complete documentation is given by files *solidParticle.H* and *solidPaticle.C* located in /`src/lagrangian/solidParticle`.
- This is simple solid spherical particle class with one-way coupling with the continuous phase.
- It is inherited from class *particle*. Inheritance is one of the key features of C++ classes. Class (called a subclass or derived type) can inherit the characteristics of another class(es) (super class or base type) plus include its own. In order to derive a class from another, a colon (:) in the declaration of the derived class is used.
- Complex inheritance is one of the main OpenFOAM characteristics.
- Its private members are diameter of the spherical particle and velocity of parcel.
- In public part class *Cloud<solidParticle>* is defined as its friend class.Class *trackData* used to pass data to *trackToFace* function is also defined in this part.
- There are two constructors for this class: constructor form components and the constructor from *Istream.*
- Member functions used to access the private members of class are defined as well.

solidParticle.H

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

   =========                 |
   \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
    \\    /   O peration     |
     \\  /    A nd           | Copyright (C) 1991-2007 OpenCFD Ltd.
      \\/     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 2 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, write to the Free Software Foundation,
 Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Class
    solidParticle

Description
    Simple solid spherical particle class with one-way coupling with the
    continuous phase.

SourceFiles
   solidParticleI.H
   solidParticle.C
   solidParticleIO.C
\*---------------------------------------------------------------------------*/

#ifndef solidParticle_H
#define solidParticle_H

#include "particle.H"
#include "IOstream.H"
#include "autoPtr.H"
#include "interpolationCellPoint.H"
#include "contiguous.H"

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

namespace Foam
{

class solidParticleCloud;

/*---------------------------------------------------------------------------*\
                           Class solidParticle Declaration
\*---------------------------------------------------------------------------*/

class solidParticle
:
    public particle<solidParticle>
{
    // Private member data

        //- Diameter
       scalar d_;

        //- Velocity of parcel
        vector U_;


public:

   friend class Cloud<solidParticle>;

    //- Class used to pass tracking data to the trackToFace function
    class trackData
```

```
{
    //- Reference to the cloud containing this particle
    solidParticleCloud& spc_;

    // Interpolators for continuous phase fields

        const interpolationCellPoint<scalar>& rhoInterp_;
        const interpolationCellPoint<vector>& UInterp_;
        const interpolationCellPoint<scalar>& nuInterp_;

    //- Local gravitational or other body-force acceleration
    const vector& g_;


public:

    bool switchProcessor;
    bool keepParticle;


    // Constructors

        inline trackData
        (
            solidParticleCloud& spc,
            const interpolationCellPoint<scalar>& rhoInterp,
            const interpolationCellPoint<vector>& UInterp,
            const interpolationCellPoint<scalar>& nuInterp,
            const vector& g
        );


    // Member functions

        inline solidParticleCloud& spc();

        inline const interpolationCellPoint<scalar>& rhoInterp() const;

        inline const interpolationCellPoint<vector>& UInterp() const;

        inline const interpolationCellPoint<scalar>& nuInterp() const;

        inline const vector& g() const;
};


// Constructors

    //- Construct from components
    inline solidParticle
    (
        const Cloud<solidParticle>& c,
        const vector& position,
        const label celli,
        const scalar m,
        const vector& U
    );
```

```
        //- Construct from Istream
        solidParticle
        (
            const Cloud<solidParticle>& c,
            Istream& is,
            bool readFields = true
       );

        //- Construct and return a clone
        autoPtr<solidParticle> clone() const
        {
            return autoPtr<solidParticle>(new solidParticle(*this));
        }


    // Member Functions

        // Access

            //- Return diameter
           inline scalar d() const;

            //- Return velocity
            inline const vector& U() const;

            //- The nearest distance to a wall that
            //  the particle can be in the n direction
            inline scalar wallImpactDistance(const vector& n) const;

       //- Tracking
        bool move(trackData&);

        //- Overridable function to handle the particle hitting a
        //-    processorPatch
        void hitProcessorPatch
        (
            const processorPolyPatch&,
            solidParticle::trackData& td
        );

      //- Overridable function to handle the particle hitting a
      //-  processorPatch without trackData
        void hitProcessorPatch
        (
            const processorPolyPatch&,
            int&
        );

    //- Overridable function to handle the particle hitting a wallPatch
        void hitWallPatch
        (
            const wallPolyPatch&,
            solidParticle::trackData& td
        );
```

```
        //- Overridable function to handle the particle hitting a wallPatch
        //- without trackData
        void hitWallPatch
        (
            const wallPolyPatch&,
            int&
        );

        //- Overridable function to handle the particle hitting a polyPatch
        void hitPatch
        (
            const polyPatch&,
            solidParticle::trackData& td
        );

        //- Overridable function to handle the particle hitting a polyPatch
        //- without trackData
        void hitPatch
        (
            const polyPatch&,
            int&
        );


    // Ostream Operator

        friend Ostream& operator<<(Ostream&, const solidParticle&);
};

template<>
inline bool contiguous<solidParticle>()
{
    return true;
}

template<>
void Cloud<solidParticle>::readFields();

template<>
void Cloud<solidParticle>::writeFields() const;


// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
} // End namespace Foam

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include "solidParticleI.H"

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

#endif

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

solidParticle.C

```
/*---------------------------------------------------------------------------------------------------------------*\
   =========                 |
   \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
    \\    /   O peration     |
     \\  /    A nd           | Copyright (C) 1991-2007 OpenCFD Ltd.
      \\/     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 2 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, write to the Free Software Foundation,
  Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA


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

#include "solidParticleCloud.H"

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

bool Foam::solidParticle::move(solidParticle::trackData& td)
{
    td.switchProcessor = false;
    td.keepParticle = true;

    const polyMesh& mesh = cloud().pMesh();
    const polyBoundaryMesh& pbMesh = mesh.boundaryMesh();

    scalar deltaT = mesh.time().deltaT().value();
    scalar tEnd = (1.0 - stepFraction())*deltaT;
    scalar dtMax = tEnd;

    while (td.keepParticle && !td.switchProcessor && tEnd > SMALL)
    {
        if (debug)
        {
            Info<< "Time = " << mesh.time().timeName()
                << " deltaT = " << deltaT
                << " tEnd = " << tEnd
                << " steptFraction() = " << stepFraction() << endl;
        }

        // set the lagrangian time-step
```

```
        scalar dt = min(dtMax, tEnd);

        // remember which cell the parcel is in
        // since this will change if a face is hit
        label celli = cell();

        dt *= trackToFace(position() + dt*U_, td);

        tEnd -= dt;
        stepFraction() = 1.0 - tEnd/deltaT;

        cellPointWeight cpw(mesh, position(), celli, face());
        scalar rhoc = td.rhoInterp().interpolate(cpw);
        vector Uc = td.UInterp().interpolate(cpw);
        scalar nuc = td.nuInterp().interpolate(cpw);

        scalar rhop = td.spc().rhop();
        scalar magUr = mag(Uc - U_);

        scalar ReFunc = 1.0;
        scalar Re = magUr*d_/nuc;

        if (Re > 0.01)
        {
            ReFunc += 0.15*pow(Re, 0.687);
        }

        scalar Dc = (24.0*nuc/d_)*ReFunc*(3.0/4.0)*(rhoc/(d_*rhop));

        U_ = (U_ + dt*(Dc*Uc + (1.0 - rhoc/rhop)*td.g()))/(1.0 + dt*Dc);

        if (onBoundary() && td.keepParticle)
        {
            if (isType<processorPolyPatch>(pbMesh[patch(face())]))
            {
                td.switchProcessor = true;
            }
        }
    }

    return td.keepParticle;
}


bool Foam::solidParticle::hitPatch
(
    const polyPatch&,
    solidParticle::trackData&,
    const label
)
{
    return false;
}


bool Foam::solidParticle::hitPatch
(
```

```
    const polyPatch&,
    int&,
    const label
)
{
    return false;
}


void Foam::solidParticle::hitProcessorPatch
(
    const processorPolyPatch&,
    solidParticle::trackData& td
)
{
    td.switchProcessor = true;
}


void Foam::solidParticle::hitProcessorPatch
(
    const processorPolyPatch&,
    int&
)
{}


void Foam::solidParticle::hitWallPatch
(
    const wallPolyPatch& wpp,
    solidParticle::trackData& td
)
{
    vector nw = wpp.faceAreas()[wpp.whichFace(face())];
    nw /= mag(nw);

    scalar Un = U_ & nw;
    vector Ut = U_ - Un*nw;

    if (Un > 0)
    {
        U_ -= (1.0 + td.spc().e())*Un*nw;
    }

    U_ -= td.spc().mu()*Ut;
}


void Foam::solidParticle::hitWallPatch
(
    const wallPolyPatch&,
    int&
)
{}


void Foam::solidParticle::hitPatch
```

```
(
    const polyPatch&,
    solidParticle::trackData& td
)
{
    td.keepParticle = false;
}


void Foam::solidParticle::hitPatch
(
    const polyPatch&,
    int&
)
{}

void Foam::solidParticle::transformProperties (const tensor& T)
{
    Particle<solidParticle>::transformProperties(T);
    U_ = transform(T, U_);
}

void Foam::solidParticle::transformProperties(const vector& separation)
{
    Particle<solidParticle>::transformProperties(separation);
}
// ************************************************************************* //
```

2.2 Class solidParticleCloud

- Complete documentation is given by files *solidParticleCloud.H* and *solidPaticleCloud.C* located in `/src/lagrangian/solidParticle`.
- It is inherited from class *cloud.*
- Its private members are fvMesh & mesh and particle properties such as particle density, restitution ratio and friction coefficient. Moreover there are private member functions that disallow default bitwise copy constructor and assignment operator.
- Constructor is defined in public part as well as member functions to access the class private members

solidParticleCloud.H

```
/*---------------------------------------------------------------------------*\
   =========                 |
   \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
    \\    /   O peration     |
     \\  /    A nd           | Copyright (C) 1991-2007 OpenCFD Ltd.
      \\/     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 2 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, write to the Free Software Foundation,
 Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA


Class
    solidParticleCloud

Description

SourceFiles
    solidParticleCloudI.H
    solidParticleCloud.C
    solidParticleCloudIO.C
\*---------------------------------------------------------------------------*/

#ifndef solidParticleCloud_H
#define solidParticleCloud_H
#include "Cloud.H"
#include "solidParticle.H"
#include "IOdictionary.H"

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

namespace Foam
{

// Class forward declarations
class fvMesh;
/*---------------------------------------------------------------------------*\
                           Class solidParticleCloud Declaration
\*---------------------------------------------------------------------------*/


class solidParticleCloud
:
    public Cloud<solidParticle>
{
    // Private data

        const fvMesh& mesh_;

        IOdictionary particleProperties_;
```

```
        scalar rhop_;
        scalar e_;
        scalar mu_;


    // Private Member Functions

        //- Disallow default bitwise copy construct
        solidParticleCloud(const solidParticleCloud&);

        //- Disallow default bitwise assignment
        void operator=(const solidParticleCloud&);


public:

    // Constructors

        //- Construct given mesh
        solidParticleCloud(const fvMesh&);


    // Member Functions

        // Access

            inline const fvMesh& mesh() const;

            inline scalar rhop() const;
            inline scalar e() const;
            inline scalar mu() const;


        // Edit

        //- Move the particles under the influence of the given
        //  gravitational acceleration
        void move(const dimensionedVector& g);

        // Write
        // -Write fields
        virtual void writeFields() const;

};

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

} // End namespace Foam

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

#include "solidParticleCloudI.H"

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

#endif
//*************************************************************************//
```

solidParticleCloud.C

```
/*---------------------------------------------------------------------------------------------------------------*\
   =========                 |
   \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
    \\    /   O peration     |
     \\  /    A nd           | Copyright (C) 1991-2007 OpenCFD Ltd.
      \\/     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 2 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, write to the Free Software Foundation,
  Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

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

#include "solidParticleCloud.H"
#include "fvMesh.H"
#include "volFields.H"
#include "interpolationCellPoint.H"

// * * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

namespace Foam
{
    defineParticleTypeNameAndDebug(solidParticle, 0);
    defineTemplateTypeNameAndDebug(Cloud<solidParticle>, 0);
};

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

Foam::solidParticleCloud::solidParticleCloud
(
    const fvMesh& mesh,
    const word& cloudName
)
:
    Cloud<solidParticle>(mesh, cloudName, false),
    mesh_(mesh),
    particleProperties_
    (
        IOobject
        (
```

```
            "particleProperties",
            mesh_.time().constant(),
            mesh_,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    ),
    rhop_(dimensionedScalar(particleProperties_.lookup("rhop")).value()),
    e_(dimensionedScalar(particleProperties_.lookup("e")).value()),
    mu_(dimensionedScalar(particleProperties_.lookup("mu")).value())
{
    solidParticle::readFields(*this);
}


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

void Foam::solidParticleCloud::move(const dimensionedVector& g)
{
 const volScalarField& rho = mesh_.lookupObject<const volScalarField>("rho");
 const volVectorField& U = mesh_.lookupObject<const volVectorField>("U");
 const volScalarField& nu = mesh_.lookupObject<const volScalarField>("nu");

    interpolationCellPoint<scalar> rhoInterp(rho);
    interpolationCellPoint<vector> UInterp(U);
    interpolationCellPoint<scalar> nuInterp(nu);

 solidParticle::trackData td(*this, rhoInterp, UInterp, nuInterp, g.value());

    Cloud<solidParticle>::move(td);
}


void Foam::solidParticleCloud::writeFields() const
{
    solidParticle::writeFields(*this);
}
// ************************************************************************//
```

2.3 SolidCylinder and solidCylinderCloud classes

Two new classes called *solidCylinder* and *solidCylinder Cloud* which stand for cylindrical particles are built from *solidParticle* and *solidParticleCloud* class respectively. In `$WM_PROJECT_USER_DIR` new directory called *solidCyliner* is created as a copy of solidParicle directory located in `src/lagrangian`.

```
cd solidCylinder
```

Make/files and Make/options are necessary as well. `Make` directory should be created.

Make/files:

```
solidCylinder.C
solidCylinderIO.C
solidCylinderCloud.C

LIB = $(FOAM_USER_LIBBIN)/libsolidCylinder
```

Make/options:

```
EXE_INC = \
   -I$(LIB_SRC)/finiteVolume/lnInclude \
   -I$(LIB_SRC)/lagrangian/basic/lnInclude \
   -I$(LIB_SRC)/lagrangian/solidParticle/lnInclude

LiB_LIBS = \
    -llagrangian \
    -lfiniteVolume
```

The compilation can be done using `wmake libso` which will build a dynamic library. The file names must be modified:

```
rename solidParticle solidCylinder *
```

In files `solidCylinder.H`, `solidCylinde.C`, `solidCylinderCloud.H`, `solidCylinderCloud.C`, `solidCylinderI.H`, `solidCylinderCloudI.H`, `solidCylinderIO.C` all occurrences of `solidParticle` should be changed to `solidCylinder`:

```
sed –i s/solidParticle/ solidCylinder/g solidCylinder.H
sed –i s/solidParticle/ solidCylinder/g solidCylinder.C
sed –i s/solidParticle/ solidCylinder/g solidCylinderCloud.H
sed –i s/solidParticle/ solidCylinder/g solidCylinderCloud.C
sed –i s/solidParticle/ solidCylinder/g solidCylinderI.H
sed –i s/solidParticle/ solidCylinder/g solidCylinderCloudI.H
sed –i s/solidParticle/ solidCylinder/g solidCylinderIO.C
```

Change in drag force due to particle shape change must be taken into account. This is realized in `solidCylinder.C` file in the following way:

```
scalar E=l_/d_;
scalar Asurfs=(2*E+1)/pow(18*pow(E,2),1/3);
scalar fshape=(1/3)*pow(1.59,1/2)+(2/3)*pow(Asurfs,1/2);
```

```
scalar Cshape=1+0.7*pow(Asurfs-1,1/2)+2.4*(Asurfs-1);
scalar Rec=Cshape*Re/fshape;
scalar ReFuncc=1+0.15*pow(Rec,0.687);
scalar Dcc=(24.0*nuc/d_)*ReFuncc*(3.0/4.0)*(rhoc/(d_*rhop));
U_=(U_+dt*(Dcc*Uc+(1.0-rhoc/rhop)*td.g()))/(1.0+dt*Dcc);
```

The compilation is done using `wmake libso.`

2.4. SolidParticleFoam solver as an application of solidParticleCloud class

It is obtained through svn:

```
svn checkout
http://openfoamextend.svn.sourceforge.net/svnroot/openfoam-
extend/trunk/Breeder_1.5/solvers/other/solidParticleFoam/
cd solidParticleFoam/solidParticleFoam
```

In order to do the compilation file `readEnviromentalproperies.H` has to be copied to the solver from OpenFoam-1.5.x version:

`cp/chalmers/sw/unsup/OpenFOAM_1.5.x/src/finiteVolume/cfdTools/general/include/readEnvironmentalProperties.`

readEnvironmentalProperties.H

```
  Info << "\nReading environmentalProperties" << endl;

    IOdictionary environmentalProperties
    (
        IOobject
        (
            "environmentalProperties",
            runTime.constant(),
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    );

    dimensionedVector g(environmentalProperties.lookup("g"));
```

It can be compiled using `wmake`.

This solver solves for the particle position and velocity. Particles are considered as spherical rigid bodies. Particle properties are density, restitution ratio and friction coefficient.
Box is used as a test case. Two particles at different initial velocities are inserted into the fluid at rest and their motion is tracked.

```
cd ../box
blockMesh
```

In order to apply this solver to cylinder particles the following steps should be done:

```
cd solidParticleFoam
mv solidParticleFoam.C soliCylinderFoam.C
sed –i s/solidParticle/ solidCylinder/g solidCylinderFoam.C
```

The files and options in Make directory of the solver must be changed as well:

Make/files:

```
solidCylinderFoam.C

EXE = $(FOAM_USER_APPBIN)/solidCylinderFoam
```

Make/options:

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/lagrangian/basic/lnInclude \
    -I$(LIB_SRC)/lagrangian/solidParticle/lnInclude \
    -I../solidCylinder/lnInclude

EXE_LIBS = \
    -lfiniteVolume \
    -llagrangian \
    -lsolidParticle \
    -L$(FOAM_USER_LIBBIN) \
    -lsolidCylinder
```

Finally,it is compiled using `wmake libso.`

solidCylinderFoam.C

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

   =========                 |
   \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
    \\    /   O peration     |
     \\  /    A nd           | Copyright Tutorial Author: Hakan Nilsson
      \\/     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 2 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, write to the Free Software Foundation,
    Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Application
    solidParticleFoam

Description
    Example of the use of the solidParticleCloud class
    You must use OpenFOAM-1.5.x

\*---------------------------------------------------------------------------*/
//Some features of the fvCFD class are needed:
#include "fvCFD.H"
//The class definitions of the solidCylinderCloud class are needed:
#include "solidCylinderCloud.H"

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

int main(int argc, char *argv[])
{

#   include "setRootCase.H"
#   include "createTime.H"
#   include "createMesh.H"
#   include "createFields.H"

//The gravitational acceleration is needed:
#   include "readEnvironmentalProperties.H"
```

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

    Info<< "\nCreating particle cloud" << endl;
    solidCylinderCloud particles(mesh);
    //The solidParticleCloud class constructor allows the specification
    //of the cloudName of the Cloud directory (default is defaultCloud):
    //solidParticleCloud particles(mesh, "myCloud");

    //The solidParticleCloud class has access member functions:
    Info<<"\nFrom constant/particleProperties:" << endl;
    Info<<"rhop = " << particles.rhop() << endl;
    Info<<"e = " << particles.e() << endl;
    Info<<"mu = " << particles.mu() << endl;

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

    for (runTime++; !runTime.end(); runTime++)
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

        Info<< "Moving particles." << nl << endl;
        particles.move(g);

        runTime.write();

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

    //It is possible to explicitly write the lagrangian fields.
    //Here they will be placed in the endTime+deltaT time directory:
    //Info<<"\nWriting fields." << endl;
    //particles.writeFields();

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

    return(0);
}
```

In `$WM_PROJECT_USER_DIR/solidParticleFoam` there should be three subdirectories: `box, solidParticleFoam,solidCylinder.` This is where `solidCylinder` directory should be placed, otherwise there will be some errors while compiling.
`Box/0` includes :`lagrangian nu rho U;`
`lagrangian /defaultCloud` includes: `d l positions U;`
More details are provided bellow.

In `0/U` the velocity of the continuous phase is defined. The corresponding OpenFOAM file is shown bellow. The velocity is an object of `volVectorField` class. The units are specified in m/s. The internal field is set to `uniform (0 0 0)` (fluid at rest) and the boundary field is set to zero gradient.

```
/*--------------------------------------------------------------------------------------------------------------------*\
|   =========                 |                                                                                      |
|   \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox                                                |
|    \\    /   O peration     |                                                                                      |
|     \\  /    A nd           | Version 1.5                                                                          |
|      \\/     M anipulation  | Web: http://www.OpenFOAM.org                                                         |
|                                                                                                                    |
\*--------------------------------------------------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);
boundaryField
{

    Walls
    {
        type            fixedValue;
        value          uniform(0 0 0)
    }
}

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

In `0/rho` the density of the continuous phase is defined. From the corresponding OpenFOAM file It can be seen that the density is defined as an object of `volScalarField` class. The units are kg/m^3. The internal field is set to 1 (air density at ambient temperature) while the boundary field is specified as zero gradients.

```
/*--------------------------------------------------------------------------------------------------------------*\
|   =========                 |                                                                                |
|   \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox                                          |
|    \\    /   O peration     |                                                                                |
|     \\  /    A nd           | Version 1.5                                                                    |
|      \\/     M anipulation  | Web: http://www.OpenFOAM.org                                                   |
|                                                                                                              |
\*--------------------------------------------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      rho;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -3 0 0 0 0 0];

internalField   uniform 1;


boundaryField
{

    Walls
    {
        type            zeroGradient;
    }

}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

In `0/nu` the kinematic viscosity of the continuous phase is defined. The corresponding OpenFOAM file is given bellow. It can be seen that viscosity is defined as an object of `volScalarField` class. The units are m^2/s and the boundary field is specified as zero gradient.

```
/*--------------------------------------------------------------------------------------------------------*\
|   =========                 |                                                                            |
|   \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox                                      |
|    \\    /   O peration     |                                                                            |
|     \\  /    A nd           | Version 1.5                                                                |
|      \\/     M anipulation  | Web: http://www.OpenFOAM.org                                               |
|                                                                                                          |
\*--------------------------------------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      nu;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -1 0 0 0 0];

internalField   uniform 1e-6;

boundaryField
{

    Walls
    {
        type            zeroGradient;
    }

}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

In `lagrangian/d` foam file particle (cylinder) diameters are specified:

```
/*--------------------------------------------------------------------------------------------------------*\
|   =========                 |                                                                            |
|   \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox                                      |
|    \\    /   O peration     |                                                                            |
|     \\  /    A nd           | Version 1.5                                                                |
|      \\/     M anipulation  | Web: http://www.OpenFOAM.org                                               |
|                                                                                                          |
\*--------------------------------------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       vectorField;
    location    "0";
    object      d;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
2
(
2.0e-3
2.0e-3
)
// ************************************************************************* //
```

In `lagrangian/l` foam file cylinder lengths are specified:

```
/*--------------------------------------------------------------------------------------------------------*\
|   =========                 |                                                                            |
|   \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox                                      |
|    \\    /   O peration     |                                                                            |
|     \\  /    A nd           | Version 1.5                                                                |
|      \\/     M anipulation  | Web: http://www.OpenFOAM.org                                               |
|                                                                                                          |
\*--------------------------------------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       vectorField;
    location    "0";
    object      d;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
2
(
2.0e-3
2.0e-3
)
// ************************************************************************* //
```

In `lagrangian/U` foam file the initial velocities of two particles (cylinders) are specified:

```
/*--------------------------------------------------------------------------------------------------------*\
|   =========                 |                                                                            |
|   \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox                                      |
|    \\    /   O peration     |                                                                            |
|     \\  /    A nd           | Version 1.5                                                                |
|      \\/     M anipulation  | Web: http://www.OpenFOAM.org                                               |
|                                                                                                          |
\*--------------------------------------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       vectorField;
    location    "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
//
2
(
(1.7e-1 0 0)
(1.7 0 0)
)
// ************************************************************************* //
```

3. Results and discussion

3.1 Reynolds numbers

Reynolds number as functions of time are presented for both, sphere and cylinder pairs. Analyzing the graphs bellow it can be noticed that cylinders experience higher Re-number then spheres, which is physically correct. Moreover, sphere and cylinder with higher initial velocity firstly have rather high Re- numbers, but for a quite short time are approaching very low values. Regarding sphere and cylinder with lower initial velocity the decreasing trend for Re-numbers can be notices, but except for the beginning of the simulation (first 0.2 s) the values remain higher comparing to the second sphere-cylinder pair.

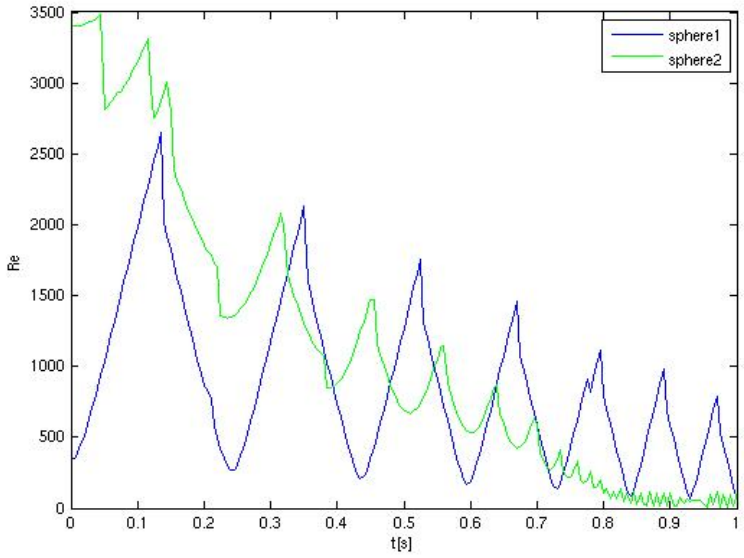

Figure4: Reynolds number as a function of time for spherical particles

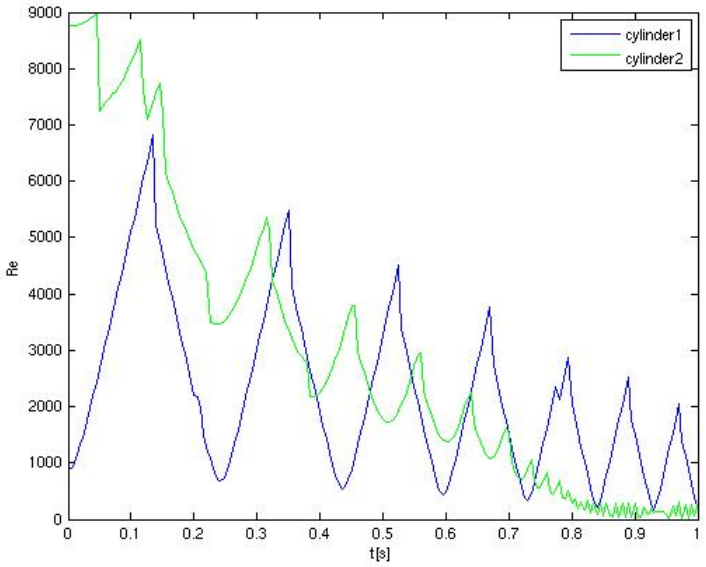

Figure5: Reynolds number as a function of time for cylindrical particles

3.2 Drag forces

Drag forces are plotted for both spheres and cylinders. From the figures below it can be seen that drag forces are higher for cylinders which is physically correct.

Another issue that should be discussed is the manner in which the steady-state condition is approached. *Figure 6* stands for the sphere and cylinder with lower velocity. The slight trend of moving toward stationary can be noticed, but nothing certain can be said for the specified simulation time. On the other hand, *Figure7* corresponds to the sphere and cylinder with higher velocity and it can clearly be seen that steady state condition is reached quite fast (approximately after 0.8 s).

Regarding the force magnitudes, higher values are noticed for the sphere and cylinder with lower initial velocities, which is in agreement with previous discussion concerning Re-numbers.

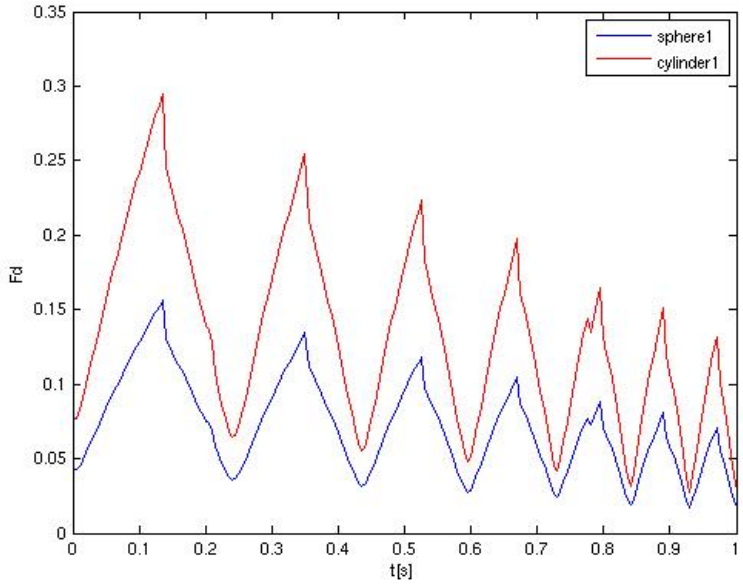

Figure6: Drag force as a function of time for sphere1 and cylinder1

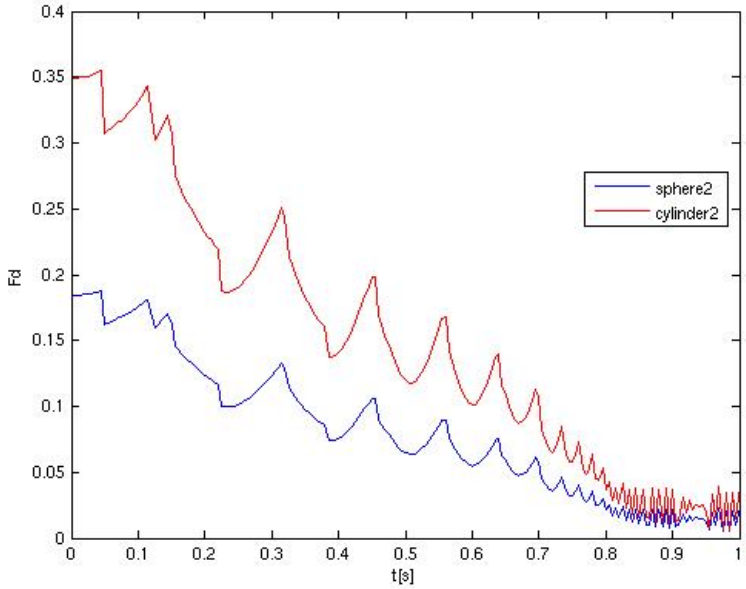

Figure7: Drag force as a function of time for sphere2 and cylinder2

3.3 Drag coefficient

In figures bellow the dependence of drag coefficients for spherical particles on Reynolds numbers is shown. The results are in good agreement with theory. For the first sphere and cylinder the Stokes (creeping flow), transition and Newton region can be identified.

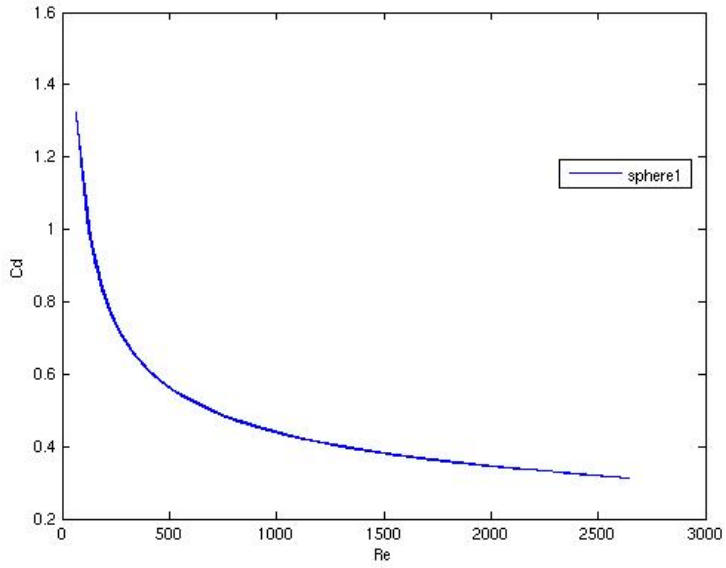

Figure8: Drag coefficient as a function of particle Reynolds number for sphere1

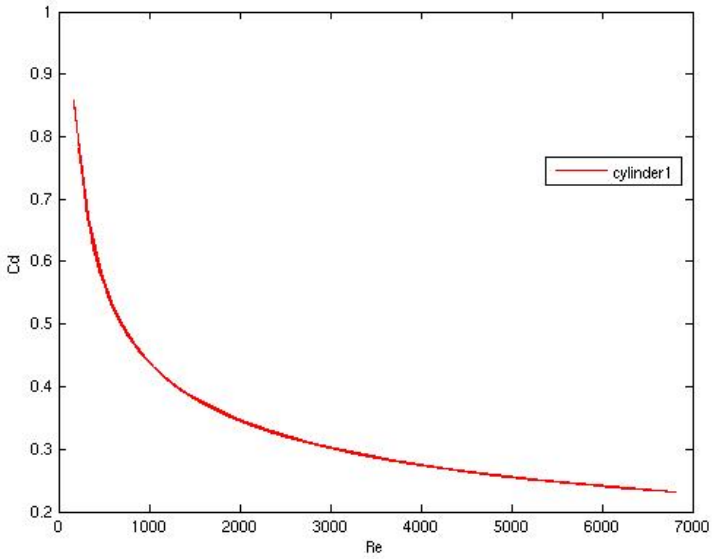

Figure9: Drag coefficient as a function of particle Reynolds number for cylinder1

Regarding the sphere and cylinder with higher initial velocity the plots do not provide clear information about the drag coefficient.

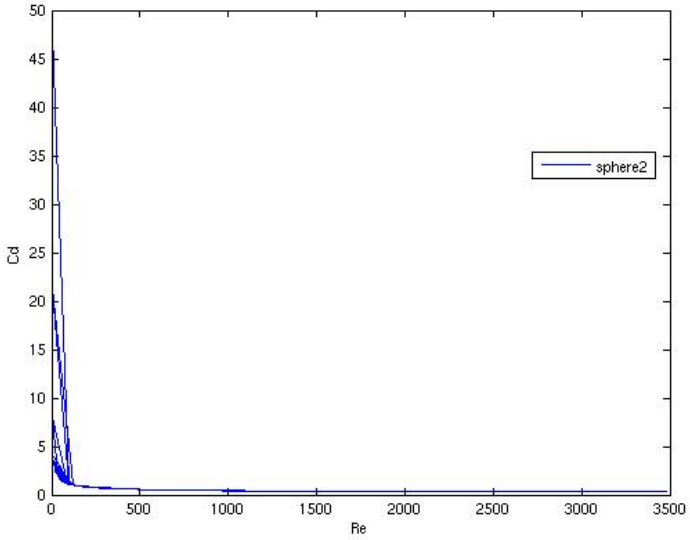

Figure10: Drag coefficient as a function of particle Reynolds number for sphere2

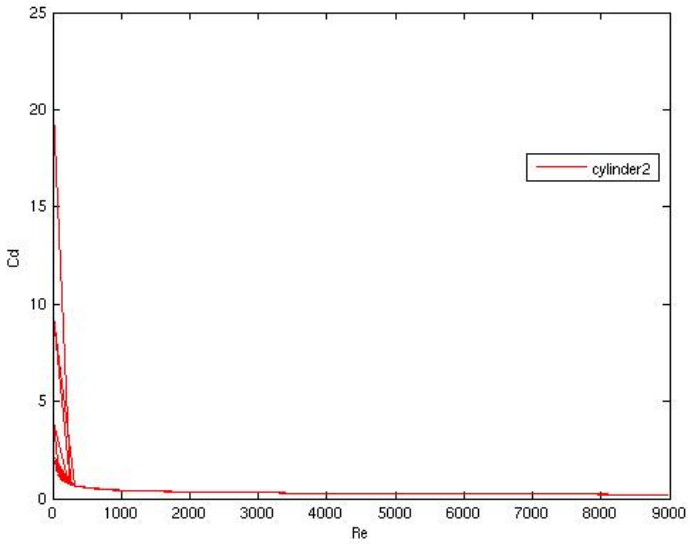

Figure11: Drag coefficient as a function of particle Reynolds number for cylinder2

In order to clarify this drag coefficients for the second sphere and cylinder are plotted versus Re-numbers for the firs 0.8 s. The trend can clearly be seen from the figures bellow. Regarding the previous plots it can be concludes that high drag coefficients correspond to very low Re-numbers that occur for computation time t=0.8-1s.

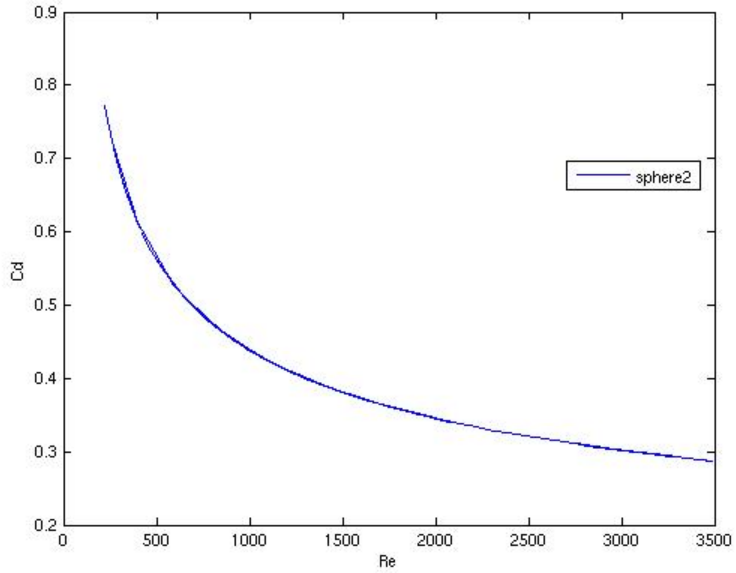

Figure12: Drag coefficient as a function of particle Reynolds number for sphere2 (computational time t=0.8s)

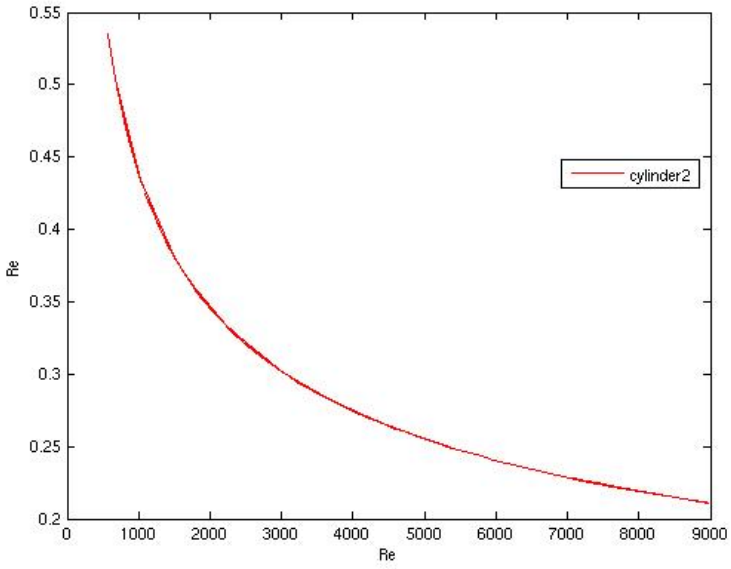

Figure13: Drag coefficient as a function of particle Reynolds number for cylinder2 (computational time t=0.8s)

References:

1. ERCOFTAC , The Best Practice Guidelines for Computational Fluid Dynamics of turbulent dispersed multiphase flows, 2008
2. Crowe, C.,Sommerfeld,M.,Tsuji, Y., Multiphase flows with droplets and particles, CRC Press, 1998
3. Loth, E., Drag of non-spherical solid particles of regular and irregular shape, Science Direct, 2007
4. Sasic, S., Van Wachem, B., Direct numerical simulation (DNS) of an individual fiber in an arbitrary flow field – an implicit immersed boundary method, Multiphase Science and Technology, Vol21, Issues 1-2, 2009
5. Lectures –PhD course in CFD with OpenSource software, Quarter2, 2009, Chalmers University of Technology
6. http://foam.sourceforge.net/doc/Doxygen/html/
7. http://www.cplusplus.com/doc/tutorial