German OpenFOAM User meeting 2018 (GOFUN 2018)

Particle Simulation with OpenFOAM®

Introduction, Fundamentals and Application

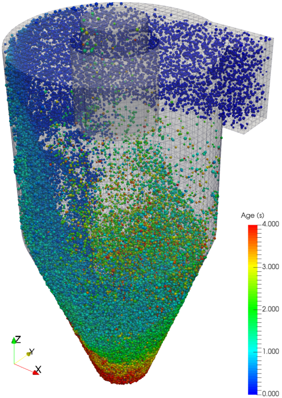
ROBERT KASPER
Chair of Modeling and Simulation,
University of Rostock

Outline

Outline

Why Particle Simulations with OpenFOAM?

- OpenFOAM is free and open source (customization and unlimited parallelization possible)
- OpenFOAM is constantly under development with a continuous growing community (academic research, R&D in companies)
- OpenFOAM includes solvers for any application of particle-laden flows (e.g. process engineering, mechanical engineering, civil engineering, physics,...)



Lagrangian-Particle-Tracking in OpenFOAM

- Solvers for any kind of particle-laden flow are already implemented[1]:
  - ***DPMFoam/MPPICFoam:*** Transient solver for the coupled transport of a single kinematic particle cloud including the effect of the volume fraction of particles on the continuous phase (Multi-Phase Particle In Cell modeling is used to represent collisions without resolving particle-particle interactions)
  - ***uncoupledKinematicParcelFoam:*** Transient solver for the passive transport of a single kinematic particle cloud
  - ***reactingParcelFilmFoam:*** Transient solver for compressible, turbulent flow with a reacting, multiphase particle cloud, and surface film modelling
  - ***sprayFoam***: Transient solver for compressible, turbulent flow with a spray particle cloud
  - ***...***
- No proper solver available? Customize one of the existing...

[1] based on OpenFOAM-5.x

Outline

Dilute Versus Dense Flows

- **Dilute flow:** particle motion is controlled by the fluid forces (e.g. drag and lift)
- **Dense flow:** particle motion is controlled by collisions or continuous contact

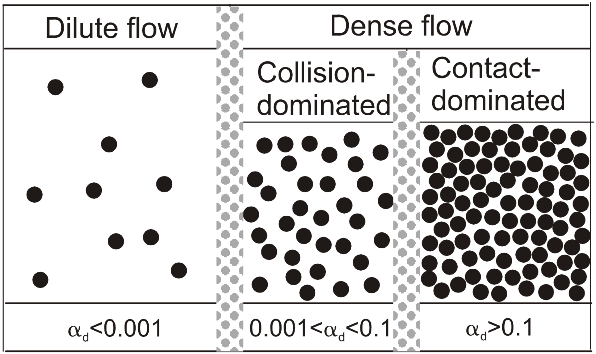


Figure: Flow regimes for dilute and dense flows according to Crowe et al. (2011)

Phase-Coupling Mechanisms

- Phase-coupling mechanisms strongly influences the behavior of the continous and dispersed phase:
  - **One-way coupling:** fluid $\rightarrow$ particles
  - **Two-way coupling:** fluid $\leftrightarrows$ particles
  - **Four-way coupling:** fluid $\leftrightarrows$ particles + particle collisions

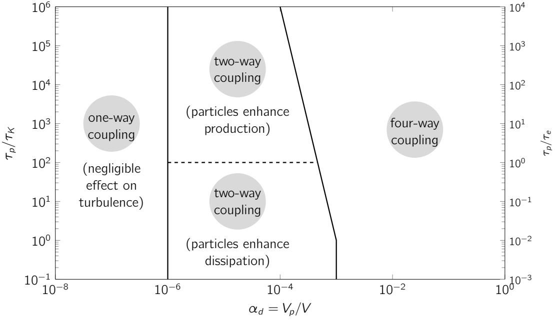


Figure: Classification of phase-coupling mechanisms according to Elghobashi (1994)

Modeling Approaches for Particle Clouds

- **DEM:** each particle is represented by an computational particle $\rightarrow$ particle motion is analyzed incorporating fluid forces, contact forces and moments due to neighboring particles
- **DPM:** parcel of particles is represented by an computational particle $\rightarrow$ dynamic properties (size, velocity, etc.) for each particle in the parcel are the same

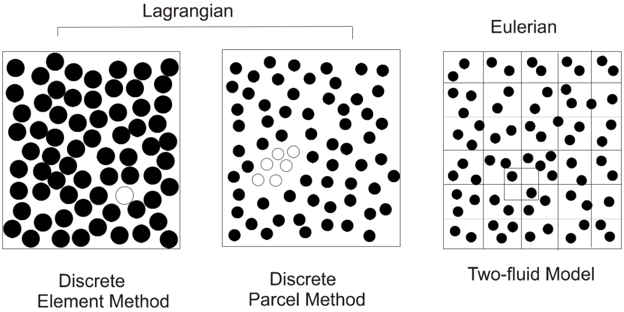


Figure: Different approaches for modeling particle clouds according to Crowe et al. (2011)

Governing Equations for particle motions

- Calculation of isothermal particle motions requires the solution of the following set of ordinary differential equations:

$$\frac{\mathrm{d}\mathbf{x}_p}{\mathrm{d}t} = \mathbf{u}_p, \qquad m_p \frac{\mathrm{d}\mathbf{u}_p}{\mathrm{d}t} = \sum \mathbf{F}_\mathrm{i}, \qquad I_p \frac{\mathrm{d}\omega_p}{\mathrm{d}t} = \sum \mathbf{T} \tag{1}$$

- Newton's second law of motion presupposes the consideration of all relevant forces acting on the particle, e.g., drag, gravitational and buoyancy forces, pressure forces:

$$m_p \frac{\mathrm{d}\mathbf{u}_p}{\mathrm{d}t} = \sum \mathbf{F}_\mathrm{i} = \mathbf{F}_\mathrm{D} + \mathbf{F}_\mathrm{G} + \mathbf{F}_\mathrm{P} + \ldots \tag{2}$$

Drag Force

- Drag is the most important force (approx. 80 % of the total force) and is expressed in terms of the drag coefficient C_D:

$$\mathbf{F_D} = C_D \frac{\pi D_p^2}{8} \rho_f \left(\mathbf{u}_f - \mathbf{u}_p\right) \left|\mathbf{u}_f - \mathbf{u}_p\right| \tag{3}$$

Drag correlations (spherical particle)

- Schiller-Naumann (1935):

$$C_D = \begin{cases} \frac{24}{\mathrm{Re}_p}\left(1 + 0.15\mathrm{Re}_p^{0.687}\right) & \text{if } \mathrm{Re}_p \leq 1000 \\ 0.44 & \text{if } \mathrm{Re}_p > 1000 \end{cases} \tag{4}$$

- **Putnam (1961):**

$$C_D = \begin{cases} \frac{24}{\mathrm{Re}_p}\left(1 + \frac{1}{6}\mathrm{Re}_p^{2/3}\right) & \text{if } \mathrm{Re}_p \leq 1000 \\ 0.424 & \text{if } \mathrm{Re}_p > 1000 \end{cases} \tag{5}$$

Drag Force

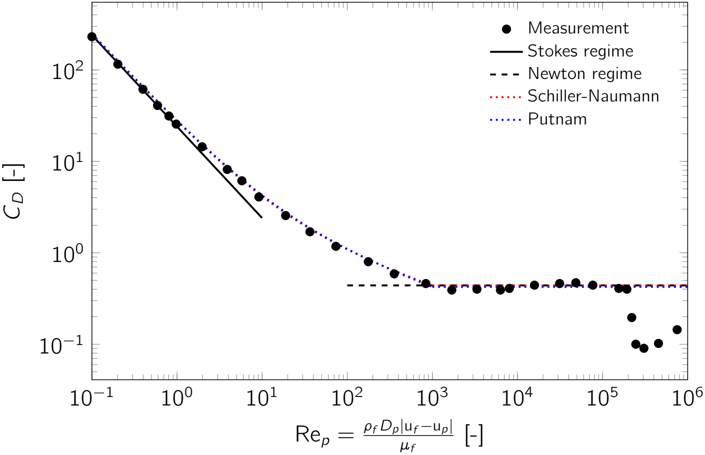


Figure: Drag coefficient as a function of particle Reynolds number, comparison of experimental data with correlations of Schiller-Naumann (1935) and Putnam (1961)

Gravity/Buoyancy and Pressure Gradient Force

- Gravitational and Buoyancy force is computed as one total force:

$$\mathbf{F_G} = m_p \mathbf{g} \left(1 - \frac{\rho_f}{\rho_p}\right) \tag{6}$$

- The force due to a local pressure gradient can be expressed for a spherical particle simply as:

$$\mathbf{F_P} = -\frac{\pi D_p^3}{6} \nabla p \tag{7}$$

- Expressing the local pressure gradient ∇p in terms of the momentum equation leads to the final pressure gradient force:

$$\mathbf{F_p} = \rho_f \frac{\pi D_p^3}{6} \left(\frac{\mathrm{D}\mathbf{u}_f}{\mathrm{D}t} - \nabla \cdot \nu \left(\nabla \mathbf{u}_f + \nabla \mathbf{u}_f^T \right) \right) \tag{8}$$

Other Forces

- **Added mass force:** particle acceleration or deceleration in a fluid requires also an accelerating or decelerating of a certain amount of the fluid surrounding the particle (important for liquid-particle flows)
- **Slip-shear lift force:** particles moving in a shear layer experience a transverse lift force due to the nonuniform relative velocity over the particle and the resulting nonuniform pressure distribution
- **Slip-rotation lift force:** particles, which are freely rotating in a flow, may also experience a lift force due to their rotation (Magnus force)
- **Thermophoretic force:** a thermal force moves fine particles in the direction of negative temperature gradients (important for gas-particle flows)
- ...

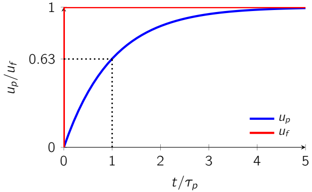
Particle Response Time/Stokes number

- Particle response time is used to characterize the capability of particles to follow sudden velocity changes in the flow
- From equation of motion for a spherical particle considering a Stokes flow (divided through by particle mass and in terms of particle Reynolds number):

$$\frac{\mathrm{d}u_p}{\mathrm{d}t} = \underbrace{\frac{18\mu_f}{\rho_p D_p^2}}_{=1/\tau_p} \underbrace{\frac{C_D \mathrm{Re}_p}{24}}_{\approx 1} (u_f - u_p) \rightarrow u_p = u_f \left[1 - \exp\left(-t/\tau_p\right)\right] \tag{9}$$

Particle response time & Stokes number

$$\tau_p = \frac{\rho_p D_p^2}{18\mu_f}, \quad \mathrm{St} = \tau_p/\tau_f \tag{10}$$



Stokes number

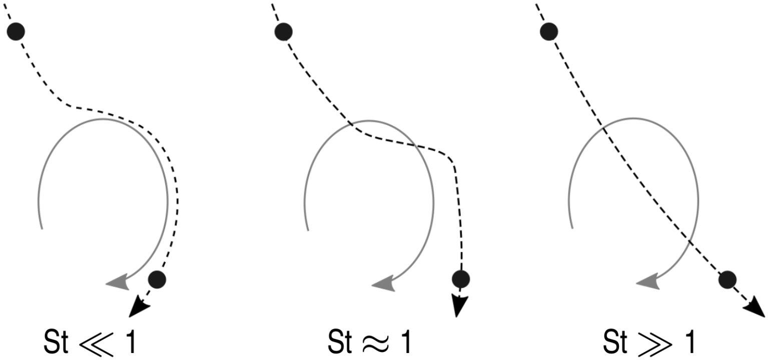


Figure: Effect of an eddy (solid line) on particle trajectory for different Stokes numbers according to Benavides and van Wachem (2008)

Particle-Particle Interaction

- OpenFOAM uses mainly the deterministic soft sphere model (modified Cundall-Strack model)
- Particle-particle collisions are considered using a spring, friction slider and dash-pot
- **Normal force** is expressed according to the *Hertzian contact theory*:

$$F_{n,ij} = \left(-k_n \delta_n^{3/2} - \eta_{n,j} \mathbf{G} \cdot \mathbf{n}\right) \mathbf{n} \qquad (11)$$

- **Tangential force** is expressed by:

$$F_{t,ij} = -k_n \delta_t - \eta_{t,j} \mathbf{G}_{ct} \quad \text{or} \qquad (12)$$

$$F_{t,ij} = -f \left|\mathbf{F}_{n,ij}\right| \mathbf{t} \quad \text{if} \quad \left|F_{t,ii}\right|_j > f \left|\mathbf{F}_{n,ij}\right| \qquad (13)$$

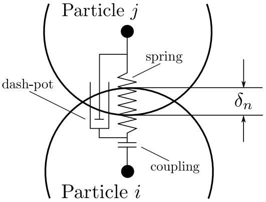


Figure: Normal force

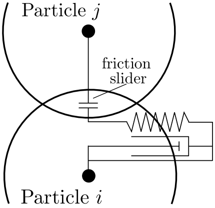


Figure: Tangential force

Let's get some practice... ☺

Outline

How to build your own Eulerian-Lagrangian Solver in OpenFOAM?

- **Problem:** no proper solver is available for your requirements? ☹
- **Solution:** customize an existing solver for your own purposes! ☺

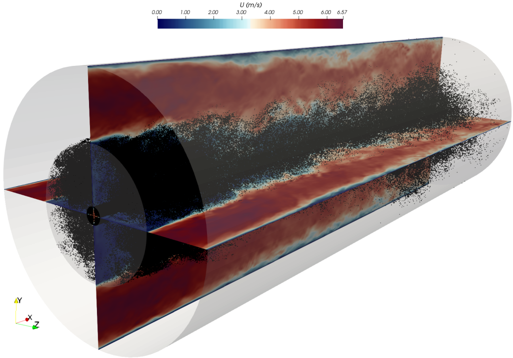


Figure: Particle-laden flow in a simplified (cold) combustion chamber

How to build your own Eulerian-Lagrangian Solver in OpenFOAM?

1. Open a terminal and source OpenFOAM-5.x (if not already done)

2. Create a working directory for our Eulerian-Lagrangian solver and move into it:

```
$ mkdir particle_tutorial/ && mkdir particle_tutorial/solver/
$ cd particle_tutorial/solver/
```

3. Copy the original **pimpleFoam** solver (Large time-step transient solver for incompressible, turbulent flow, using the PIMPLE (merged PISO-SIMPLE) algorithm) from OpenFOAM-5.x and rename it:

```
$ cp -r $FOAM_SOLVERS/incompressible/pimpleFoam/ .
$ mv pimpleFoam pimpleLPTFoam
```

How to build your own Eulerian-Lagrangian Solver in OpenFOAM?

4. Move into the pimpleLPTFoam directory, change the name of the pimpleFoam.C file and remove the pimpleDyMFoam and SRFPimpleFoam sub-solver directories:

```
$ cd pimpleLPTFoam
$ mv pimpleFoam.C pimpleLPTFoam.C
$ rm -r pimpleDyMFoam/ SRFPimpleFoam/
```

5. Copy the lagrangian library **intermediate** (includes submodels for particle forces, particle collisions, injection and dispersion models,...) from OpenFOAM-5.x:

```
$ cp -r $FOAM_SRC/lagrangian/intermediate/ .
```

6. Open the createFields.H file with a text editor for some customizations:

```
$ vi createFields.H
```

How to build your own Eulerian-Lagrangian Solver in OpenFOAM?

7. Add the following code lines after `#include "createMRF.H"` to create and read the fluid density from the transportProperties and calculate the inverse fluid density:

createFields.H

```
Info« "Reading transportProperties\n" « endl;
IOdictionary transportProperties
(
    IOobject
    (
        "transportProperties",
        runTime.constant(),
        mesh,
        IOobject::MUST_READ_IF_MODIFIED,
        IOobject::NO_WRITE
    )
);
dimensionedScalar rhoInfValue
(
    transportProperties.lookup("rhoInf")
);
dimensionedScalar invrhoInf("invrhoInf",(1.0/rhoInfValue));
```

How to build your own Eulerian-Lagrangian Solver in OpenFOAM?

8. Create a volScalarField for the fluid density and the dynamic fluid viscosity:

createFields.H

```
volScalarField rhoInf
(
    IOobject
    (
        "rho",
        runTime.constant(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    mesh,
    rhoInfValue
);
```

createFields.H

```
volScalarField mu
(
    IOobject
    (
        "mu",
        runTime.constant(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    laminarTransport.nu()
    *rhoInfValue
);
```

How to build your own Eulerian-Lagrangian Solver in OpenFOAM?

9. Initialize the basicKinematicCollidingCloud (includes particle-particle interactions):

createFields.H

```
const word kinematicCloudName
(
      args.optionLookupOrDefault<word>("cloudName", "kinematicCloud")
);
Info« "Constructing kinematicCloud " « kinematicCloudName « endl;
basicKinematicCollidingCloud kinematicCloud
(
      kinematicCloudName,
      rhoInf,
      U,
      mu,
      g
);
```

10. Open the pimpleLPTFoam.C file for some customizations:

```
$ vi pimpleLPTFoam.C
```

How to build your own Eulerian-Lagrangian Solver in OpenFOAM?

11. Add the basicKinematicCollidingCloud.H and readGravitationalAcceleration.H to the existing header files:

pimpleLPTFoam.C

```
...
#include "turbulentTransportModel.H"
#include "pimpleControl.H"
#include "fvOptions.H"
#include "basicKinematicCollidingCloud.H"

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

int main(int argc, char *argv[])
{
     #include "setRootCase.H"
     #include "createTime.H"
     #include "createMesh.H"
     #include "readGravitationalAcceleration.H"
     #include "createControl.H"
     #include "createTimeControls.H"
     #include "createFields.H"

...
```

How to build your own Eulerian-Lagrangian Solver in OpenFOAM?

12. Add the kinematicCloud.evolve() function after the PIMPLE corrector loop:

pimpleLPTFoam.C

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

      // -- Pressure corrector loop

      while (pimple.correct())
      {
           #include "pEqn.H"
      }
      if (pimple.turbCorr())
      {
           laminarTransport.correct();
           turbulence->correct();
      }
}
Info« "\nEvolving " « kinematicCloud.name() « endl;
kinematicCloud.evolve();

runTime.write();
```

How to build your own Eulerian-Lagrangian Solver in OpenFOAM?

13. Open the UEqn.H file for some customizations:

```
$ vi UEqn.H
```

14. Expand the momentum equation for two-way coupling (dusty gas equation with point-force approach):

UEqn.H

```
tmp<fvVectorMatrix> tUEqn
(
      fvm::ddt(U)
      + fvm::div(phi, U)
      + MRF.DDt(U)
      + turbulence->divDevReff(U)
      ==
      fvOptions(U)
      + invrhoInf*kinematicCloud.SU(U)
);
fvVectorMatrix& UEqn = tUEqn.ref();
UEqn.relax();
```

How to build your own Eulerian-Lagrangian Solver in OpenFOAM?

15. The implementation is (almost) done, but we need some customizations within the Make directory of the intermediate library in order to compile everything correctly:

```
$ vi intermediate/Make/files
```

16. We want our own customized intermediate library (maybe to implement a own particle force model or similar), so replace the last code line of the files file with:

files

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

17. Tell the solver where he can find our intermediate library (and some additional too):

```
$ vi Make/options
```

How to build your own Eulerian-Lagrangian Solver in OpenFOAM?

options

```
EXE_INC =
      -Ilagrangian/intermediate/lnInclude \
      -I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
      -I$(LIB_SRC)/TurbulenceModels/incompressible/lnInclude \
      -I$(LIB_SRC)/transportModels \
      -I$(LIB_SRC)/transportModels/incompressible/singlePhaseTransportModel \
      -I$(LIB_SRC)/finiteVolume/lnInclude \
      -I$(LIB_SRC)/meshTools/lnInclude \
      -I$(LIB_SRC)/sampling/lnInclude \
      -I$(LIB_SRC)/lagrangian/basic/lnInclude \
      -I$(LIB_SRC)/regionModels/surfaceFilmModels/lnInclude \
      -I$(LIB_SRC)/regionModels/regionModel/lnInclude

EXE_LIBS = \
      -L$(FOAM_USER_LIBBIN) \
      -lPimpleLPTLagrangianIntermediate \
      -llagrangian\
      -lturbulenceModels \
      -lincompressibleTurbulenceModels \
      -lincompressibleTransportModels \
      -lfiniteVolume \

...
```

How to build your own Eulerian-Lagrangian Solver in OpenFOAM?

18. Tell the compiler the name of our new Eulerian-Lagrangian solver:

```
$ vi Make/files
```

files

```
pimpleLPTFoam.C

EXE = $(FOAM_USER_APPBIN)/pimpleLPTFoam
```

19. Finally, we can compile the intermediate library and the solver:

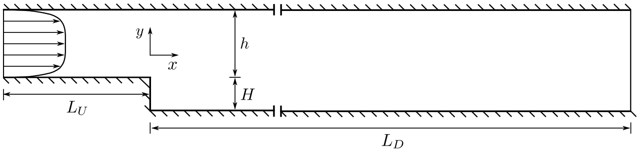
```
$ wmake all
```

You received no error messages from the compiler?
Congratulations, your new Eulerian-Lagrangian solver is ready...
but how to use it? ☺

How to use your own Eulerian-Lagrangian Solver in OpenFOAM?

Particle-laden backward facing step flow (Fessler & Eaton, 1999)

- Geometry:
  - Step height: $H = 26.7\,\text{mm}$
  - Channel height/width: $h = 40\,\text{mm}$, $B = 457\,\text{mm}$
  - Length inlet and expansion channel: $L_U = 10h$, $L_D = 35h$



- Flow and particle characteristics:
  - Centerline velocity and Reynolds number: $U_0 = 10.5\,\text{m/s}$, $\text{Re}_0 = U_0 H/\nu = 18{,}600$
  - Particle type: copper $\rightarrow D_p = 70\,\mu\text{m}$, $\rho_p = 8{,}800\,\text{kg/m}^3$
  - Particle mass loading ratio: $\eta = \dot{m}_p/\dot{m}_f = 0.1$

How to use your own Eulerian-Lagrangian Solver in OpenFOAM?

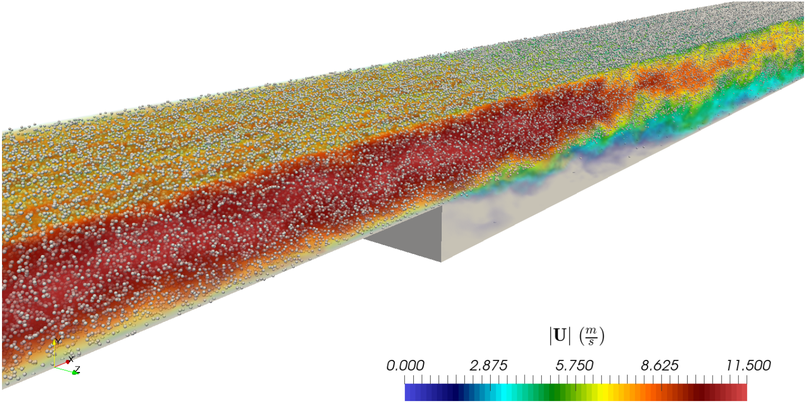


Figure: Particle-laden backward-facing step flow according to Fessler & Eaton (1999)

How to use your own Eulerian-Lagrangian Solver in OpenFOAM?

- Basic folder structure of any OpenFOAM case:

0: includes the initial boundary conditions

constant: includes the mesh (polyMesh folder), physical properties of the fluid (transportProperties), particle properties and settings (kinematicCloudProperties),...

system: includes the simulation settings (controlDict), settings for numerical schemes (fvSchemes) and solver for the algebraic equations systems (fvSolution), decomposition methods (decomposeParDict), ...

- Download the current tutorial case setup using the git clone command:

Git repository on Bitbucket

```
$ git clone https://slint@bitbucket.org/slint/gofun2018_particletut.git
```

How to use your own Eulerian-Lagrangian Solver in OpenFOAM?

1. We start with the mesh generation → move into the tutorial directory and build the 2D mesh using OpenFOAM's blockMesh utility and check the mesh quality:

```
$ cd gofun2018_particletut/case/BFS/
$ blockMesh
$ checkMesh
```

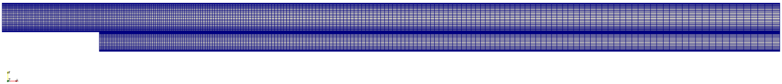


Figure: Two-dimensional block-structured mesh for the particle-laden backward facing step flow

How to use your own Eulerian-Lagrangian Solver in OpenFOAM?

2. Let's see how to define initial boundary conditions (at the example of the velocity field):

U

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

internalField uniform (0 0 0);

boundaryField
{
        inlet
        {
                type fixedValue;
                value uniform (9.39 0 0);
        }
        outlet
        {
                type zeroGradient;
        }
        walls
        {
                type noSlip;
        }
        sides
        {
                type empty;
        }

}
```

- OpenFOAM needs the dimension of the flow field in SI-units
- You can set an initial flow field if present
- Each patch needs an initial boundary condition
- Boundary conditions in OpenFOAM:
    - Dirichlet (fixedValue)
    - Neumann (fixedGradient/zeroGradient)
    - Special types: cyclic, symmetry, empty (for 2D caes), ...

How to use your own Eulerian-Lagrangian Solver in OpenFOAM?

3. Let's see how to set up the particle cloud:

kinematicCloudProperties

```
solution
{
        active true;
        coupled true;
        transient yes;
        cellValueSourceCorrection off;
        maxCo 0.5;
        interpolationSchemes
        {
              rho cell;
              U cellPoint;
              mu cell;
        }
        integrationSchemes
        {
              U Euler;
        }
sourceTerms
{
        schemes
        {
              U semiImplicit 1;
        }

}
```

- Activate/de-activate the particle cloud
- Enable/disable phase coupling
- Transient/steady-state solution (max. Courant number)
- Enable/disable correction of momentum transferred to the Eulerian phase
- Choose interpolation/integration schemes for the LPT and treatment of source terms

How to use your own Eulerian-Lagrangian Solver in OpenFOAM?

kinematicCloudProperties

```
constantProperties
{
        parcelTypeId 1;
        rhoO 8800;
        youngsModulus 1e4;
        poissonsRatio 0.001;
}
subModels
{
        particleForces
        {
              sphereDrag;
              gravity;
        }
```

- Define the physical particle properties:
  - Density
  - Young's module (elastic modulus)
  - Poisson's ratio
- Define the relevant particle forces:
  - Drag force
  - Gravity/Buoyancy force

How to use your own Eulerian-Lagrangian Solver in OpenFOAM?

kinematicCloudProperties

```
injectionModels
{
        model1
        {
                type patchInjection;
                patchName inlet;
                duration 1;
                parcelsPerSecond 33261;
                massTotal 0;
                parcelBasisType fixed;
                flowRateProfile constant 1;
                nParticle 1;
                SOI 0.4;
                U0 (9.39 0 0);
                sizeDistribution
                {
                        type fixedValue;
                        fixedValueDistribution
                        {
                                value 0.00007;
                        }
                }
        }
}
```

- Define the particle injection:
  - Injection model + injection patch name
  - Total duration of particle injection
  - Injected parcels/particles per second
  - Number of particles per parcel
  - Start-of-injection time (SOI)
  - Initial parcel/particle velocity (U_0)
  - Size distribution model (normal size distribution, ...)

How to use your own Eulerian-Lagrangian Solver in OpenFOAM?

kinematicCloudProperties

```
dispersionModel none;
patchInteractionModel
standardWallInteraction;
standardWallInteractionCoeffs
{
        type rebound;
}
localInteractionCoeffs
heatTransferModel none;
surfaceFilmModel none;
collisionModel pairCollision;
stochasticCollisionModel none;

radiation off;
```

- Define the particle injection:
  - Injection model + injection patch name
  - Total duration of particle injection
  - Injected parcels/particles per second
  - Number of particles per parcel
  - Start-of-injection time (SOI)
  - Initial parcel/particle velocity (U_0)
  - Size distribution model (normal size distribution, ...)

How to use your own Eulerian-Lagrangian Solver in OpenFOAM?

kinematicCloudProperties

```
pairCollisionCoeffs
{
        maxInteractionDistance 0.00007;
        writeReferredParticleCloud no;
        pairModel pairSpringSliderDashpot;
        pairSpringSliderDashpotCoeffs
        {
                useEquivalentSize no;
                alpha 0.12;
                b 1.5;
                mu 0.52;
                cohesionEnergyDensity 0;
                collisionResolutionSteps 12;
        };
        wallModel wallSpringSliderDashpot;
        wallSpringSliderDashpotCoeffs
        {
                useEquivalentSize no;
                collisionResolutionSteps 12;
                youngsModulus 1e10;
                poissonsRatio 0.23;
                alpha 0.12;
                b 1.5;
                mu 0.43;
                cohesionEnergyDensity 0;
        };
}
```

- Set up the particle-particle and particle-wall interaction model coefficients:
  - α: coefficient related to the coefficient of restitution e (diagram!)
  - b: Spring power $\rightarrow b = 1$ (linear) or $b = 3/2$ (Hertzian theory)
  - μ: friction coefficient

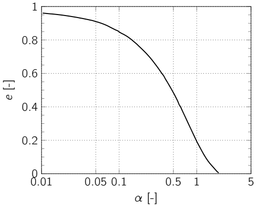


Figure: Relationship between α and the coefficient of restitution e (Tsuji et al., 1992)

How to use your own Eulerian-Lagrangian Solver in OpenFOAM?

kinematicCloudProperties

```
cloudFunctions
{
        voidFraction1
        {
                type voidFraction;
        }

}
```

- Set up some cloudFunctions (record particle tracks, calculate particle erosion, ...)

4. The last step is to define the vector of the gravitational acceleration:

g

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

value (9.81 0 0);
```

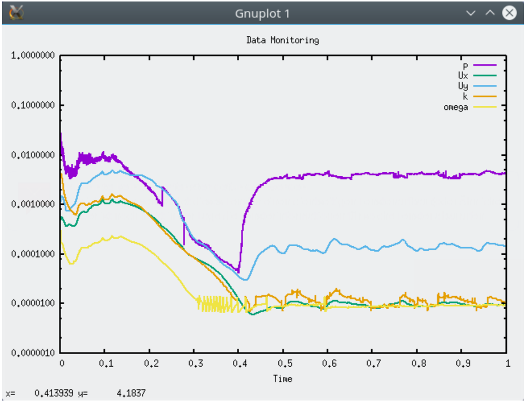
5. Finally, start our solver (and write a log file):

```
$ pimpleLPTFoam > run.log
```

How to use your own Eulerian-Lagrangian Solver in OpenFOAM?

6. Use OpenFOAM's foamMonitor utiltiy to check the convergence of our solution:

```
$ foamMonitor -l postProcessing/residuals/0/residuals.dat
```

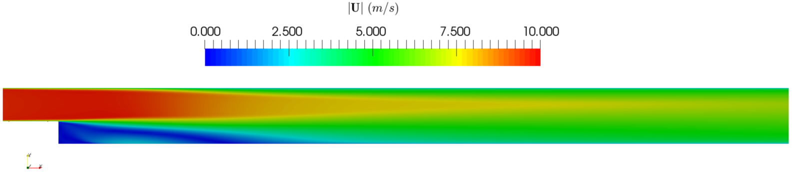


Post-Processing with OpenFOAM/Paraview

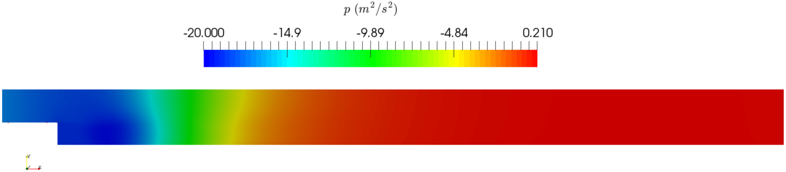
- OpenFOAM provides many utilities (e.g. sampling of data) and functionObjects (e.g. calculation of forces and turbulence fields) for the analysis of simulation results
- The standard program for the graphical post-processing of OpenFOAM cases is Paraview (see OpenFOAM user guide)
  1. Start post-processing with Paraview by typing:

```
$ paraFoam
```

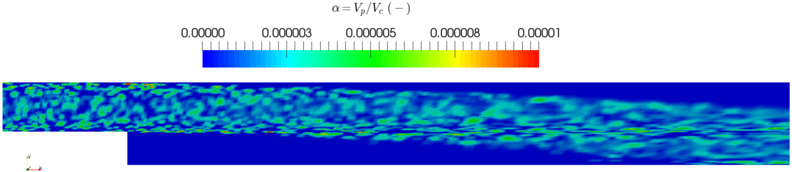
  2. Load the last time step and check the velocity and pressure field:



Post-Processing with OpenFOAM/Paraview

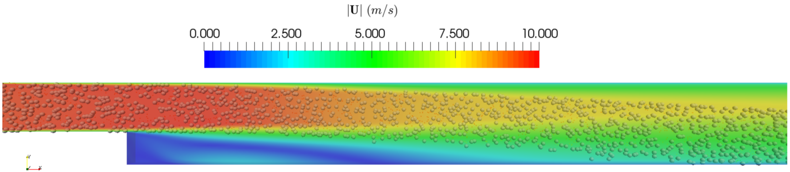


3. Let's check how much volume of each grid cell is occupied by particles (volume fraction $\alpha = V_p / V_c$ of the dispersed phase):

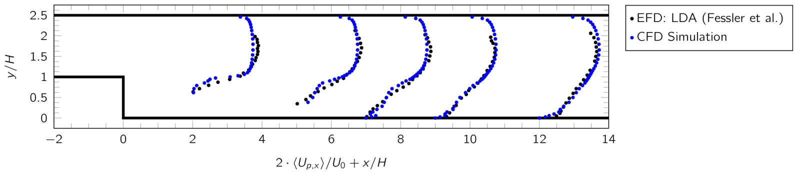


Post-Processing with OpenFOAM/Paraview

4. Apply the **Extract Block** filter on the kinematicCloud and scale the particles using the **Glyph** filter:



5. Sample the flow and particle velocity using OpenFOAM's sample utility (see OpenFOAM user guide) and plot the velocity profiles:



Literature

OpenFOAM User/Programmers Guide (www.openfoam.org)

Crow, T. C., Schwarzkopf, J. D., Sommerfeld, M. and Tsuji, Y., 2011, Multiphase flows with droplets and particles, 2nd ed., CRC Press, Taylor & Francis.

Sommerfeld, M., 2010, Particle Motion in Fluids, VDI Heat Atlas, Springer.

Benavides, A. and van Wachem, B., 2008, Numerical simulation and validation of dilute turbulent gas–particle flow with inelastic collisions and turbulence modulation, Powder Tech., Vol. 182(2), pp. 294-306.

Elghobashi, S., 1994, On predicting particle-laden turbulent flows, Applied Scientific Research, Vol. 52, pp. 309-329.

Fessler, J. R. and Eaton, J. K., 1999, Turbulence modification by particles in a backward-facing step flow, J. Fluid Mech., Vol. 394, pp. 97-117.

Tsuji, Y., Tanaka, T. and Ishida, T., 1992, Lagrangian numerical simulation of plug flow of collisionless particles in a horizontal pipe, Powder Tech., Vol. 71, 239-250.

Thank you for your attention!

Any questions?

Robert Kasper, M.Sc.

University of Rostock
Department of Mechanical Engineering and Marine Technology
Chair of Modeling and Simulation
Albert-Einstein-Str. 2
18059 Rostock

Email: robert.kasper@uni-rostock.de