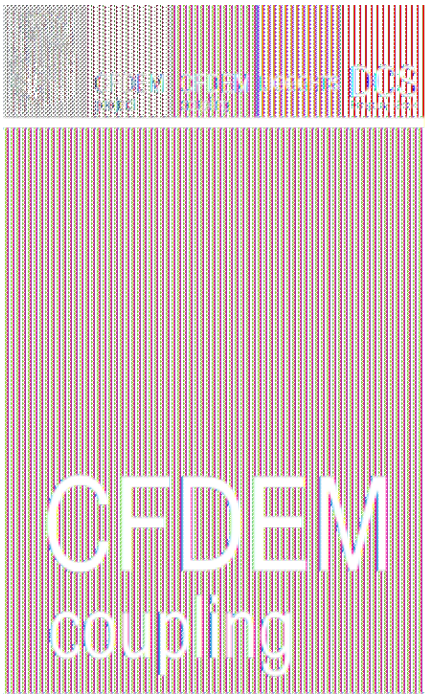
CFDEMcoupling Documentation



1. Contents

The CFDEMcoupling documentation is organized into the following sections. If you find any errors or omissions in this manual or have suggestions for useful information to add, please send an email to the developers so the CFDEMcoupling documentation can be improved.

1.1 About CFDEMcoupling

CFDEM coupling provides an open source parallel coupled CFD-DEM framework combining the strengths of LIGGGHTS DEM code and the Open Source CFD package OpenFOAM(R)(*). The CFDEMcoupling toolbox allows to expand standard CFD solvers of OpenFOAM(R)(*) to include a coupling to the DEM code LIGGGHTS. In this toolbox the particle representation within the CFD solver is organized by "cloud" classes. Key functionalities are organised in sub-models (e.g. force models, data exchange models, etc.) which can easily be selected and combined by dictionary settings.

The coupled solvers run fully parallel on distributed-memory clusters. Features are:

- its modular approach allows users to easily implement new models
- its MPI parallelization enables to use it for large scale problems
- the forum on CFD-DEM gives the possibility to exchange with other users / developers

- the use of GIT allows to easily update to the latest version
- basic documentation is provided

The file structure:

- *src* directory including the source files of the coupling toolbox and models
- *applications* directory including the solver files for coupled CFD-DEM simulations
- *doc* directory including the documentation of CFDEMcoupling
- *tutorials* directory including basic tutorial cases showing the functionality

Details on installation are given on the CFDEMproject WWW Site . The functionality of this CFD-DEM framework is described via tutorial cases showing how to use different solvers and models.

CFDEMcoupling stands for Computational Fluid Dynamics (CFD) -Discrete Element Method (DEM) coupling.

CFDEMcoupling is an open-source code, distributed freely under the terms of the GNU Public License (GPL).

Core development of CFDEMcoupling is done by Christoph Goniva and Christoph Kloss, both at DCS Computing GmbH, 2012

This documentation was written by Christoph Goniva, DCS Computing GmbH, 2012

(*) This offering is not approved or endorsed by OpenCFD Limited, the producer of the OpenFOAM software and owner of the OPENFOAMÂ® and OpenCFDÂ® trade marks.

1.2 Installation

Please follow the installation routine provided at www.cfdem.com. In order to get the latest code version, please use the git repository at http://github.com (githubAccess).

1.3 Tutorials

General:

Each solver of the CFDEMcoupling comes with at least one tutorial example, showing its functionality and correct usage. Provided that the installation is correct, the tutorials can be run via "Allrun.sh" shell scripts. These scripts perform all necessary steps (preprocessing, run, postprocessing, visualization).

Location:

The tutorials can be found in the directory $CFDEM_PROJECT_DIR/tutorials, which can be reached by typing "cfdemTut"

Structure:

Each case is structured in a directory called "CFD" covering the CFD relevant settings and data, and a dirctory called "DEM" covering the DEM relevant settings and data. This allows to easily expand a pure CFD or DEM simulation case to a coupled case.

Usage:

Provided that the installation is correct, the tutorials can be run via "Allrun.sh" shell script, executed by typing "./Allrun.sh". The successful run of the script might need some third party software (e.g. octave, evince, etc.).

Settings:

The main settings of a simulation are done via dictionaries:

The DEM setup of each case is defined by a LIGGGHTS input file located in $caseDir/DEM (e.g. in.liggghts_init). For details on the LIGGGHTS setup, please have a look at the LIGGGHTS manual.

Standard CFD settings are defined in $caseDir/CFD/constant (e.g. transportProperties, RASproperties, etc.) and $caseDir/CFD/system (e.g. fvSchemes, controlDict). You can find more information on that in OpenFOAM(R)(*) documentations (www.openFoam.com)(*).

Settings of the coupling routines are defined in $caseDir/CFD/constant/couplingProperies (e.g. force models, data exchange model, etc.) and $caseDir/CFD/constant/liggghtsCommands (allows to execute a LIGGGHTS command during a coupled simulation).

1.4 "couplingProperties" dictionary

General:

In the "couplingProperties" dictionary the setup of the coupling routines of the CFD-DEM simulation are defined.

Location: $caseDir/CFD/constant

Structure:

The dictionary is divided into two parts, "sub-models & settings" and "sub-model properties".

In "sub-models & settings" the following routines must be specified:

- modelType
- couplingInterval
- voidFractionModel
- locateModel
- meshMotionModel
- regionModel
- IOModel
- dataExchangeModel
- averagingModel
- forceModels
- momCoupleModels
- turbulenceModelType

In "sub-model properties" sub-dictionaries might be defined to specify model specific parameters.

Settings:

Reasonable example settings for the "couplingProperties" dictionary are given in the tutorial cases.

`modelType`

"modelType" refers to the formulation of the equations to be solved. Choose "A", "B" or "Bfull", according to Zhou et al. (2010): "Discrete particle simulation of particle-fluid flow: model formulations and their applicability", JFM. "A" requires the use of the force models gradPForce and viscForce, whereas "B" requires the force model "Archimedes". "Bfull" refers to model type I.

`couplingInterval`

The coupling interval determines the time passing between two CFD-DEM data exchanges.

A useful procedure would be: 1) Set the DEM timestep in the in.xxx file according to the needs of the pure DEM problem. 2) Set the "couplingInterval", which refers to the DEM timesteps. Depending on the problem you will need to have a close (small couplingInterval) or loose coupling. 3) Choose the CFD timestep in the controlDict. It must be equal to or smaller than the coupling time, otherwise you will get the error: "Error - TS bigger than coupling interval!".

Example: DEMts=0.00001s, couplingInterval=10 exchange data (=couple) will happen every 0.0001s.

1.5 "liggghtsCommands" dictionary

General:

In the "liggghtsCommands" dictionary liggghts commands being executed during a coupled CFD-DEM simulation are specified.

Location: $caseDir/CFD/constant

Structure:

The dictionary is divided into two parts, first a list of "liggghtsCommandModels" is defined, then the settings for each model must be specified.

Settings:

Reasonable example settings for the "liggghtsCommands" dictionary are given in the tutorial cases.

1.6 Models/Solvers

This section lists all CFDEMcoupling sub-models and solvers alphabetically, with a separate listing below of styles within certain commands.

IOModel	IOModel_basicIO
IOModel_noIO	IOModel_sophIO
IOModel_trackIO	averagingModel
averagingModel_dense	averagingModel_dilute
cfdemSolverIB	cfdemSolverPiso
cfdemSolverPisoScalar	clockModel
clockModel_noClock	clockModel_standardClock
dataExchangeModel	dataExchangeModel_noDataExchange
dataExchangeModel_oneWayVTK	dataExchangeModel_twoWayFiles
dataExchangeModel_twoWayMPI	forceModel
forceModel_Archimedes	forceModel_ArchimedesIB

forceModel_DiFeliceDrag	forceModel_GidaspowDrag
forceModel_KochHillDrag	forceModel_LaEuScalarTemp
forceModel_MeiLift	forceModel_SchillerNaumannDrag
forceModel_ShirgaonkarIB	forceModel_fieldStore
forceModel_fieldTimeAverage	forceModel_gradPForce
forceModel_noDrag	forceModel_particleCellVolume
forceModel_particleVolume	forceModel_scalarGeneralExchange
forceModel_virtualMassForce	forceModel_viscForce
forceModel_volWeightedAverage	forceSubModel
forceSubModel_ImEx	liggghtsCommandModel
liggghtsCommandModel_execute	liggghtsCommandModel_readLiggghtsData
liggghtsCommandModel_runLiggghts	liggghtsCommandModel_writeLiggghts
locateModel	locateModel_engineSearch
locateModel_engineSearchIB	locateModel_standardSearch
meshMotionModel	meshMotionModel_noMeshMotion
momCoupleModel	momCoupleModel_explicitCouple
momCoupleModel_implicitCouple	momCoupleModel_noCouple
probeModel	probeModel_noProbe
probeModel_particleProbe	regionModel
regionModel_allRegion	smoothingModel
smoothingModel_constDiffSmoothing	smoothingModel_noSmoothing
voidfractionModel	voidfractionModel_GaussVoidFraction
voidfractionModel_IBVoidFraction	voidfractionModel_bigParticleVoidFraction
voidfractionModel_centreVoidFraction	voidfractionModel_dividedVoidFraction
voidfractionModel_noVoidFractionVoidFraction	voidfractionModel_trilinearVoidFraction

averagingModel_dense command

Syntax:

Defined in couplingProperties dictionary.

```
averagingModel dense;
```

Examples:

```
averagingModel dense;
```

Description:

The averaging model performs the Lagrangian->Eulerian mapping of data (e.g. particle velocities). In the "cfdemParticle cloud" this averaging model is used to calculate the average particle velocity inside a CFD cell. The "dense" model is supposed to be applied to cases where the granular regime is rather dense.

Restrictions:

No known restrictions.

Related commands:

averagingModel, dilute

averagingModel_dilute command

Syntax:

Defined in couplingProperties dictionary.

```
averagingModel dilute;
```

Examples:

```
averagingModel dilute;
```

Description:

The averaging model performs the Lagrangian->Eulerian mapping of data (e.g. particle velocities). In the "cfdemParticle cloud" this averaging model is used to calculate the average particle velocity inside a CFD cell. The "dilute" model is supposed to be applied to cases where the granular regime is rather dilute. The particle velocity inside a CFD cell is evaluated from a single particle in a cell (no averaging).

Restrictions:

This model is computationally efficient, but should only be used when only one particle is inside one CFD cell.

Related commands:

averagingModel, dense

averagingModel command

Syntax:

Defined in couplingProperties dictionary.

```
averagingModel model;
```

- model = name of averaging model to be applied

Examples:

```
averagingModel dense;
averagingModel dilute;
```

Note: This examples list might not be complete - please have a look for other averaging models (averagingModel_XY) in this documentation.

Description:

The averaging model performs the Lagrangian->Eulerian mapping of data (e.g. particle velocities).

Restrictions:

None.

Related commands:

dense, dilute

Default: none

cfdemSolverIB command

Description:

"cfdemSolverIB" is a coupled CFD-DEM solver using CFDEMcoupling, an open source parallel coupled CFD-DEM framework, for calculating the dynamics between immersed bodies and the surrounding fluid. Being an implementation of an immersed boundary method it allows tackling problems where the body diameter exceeds the maximal size of a fluid cell. Using the toolbox of OpenFOAM(R)(*) the governing equations of the fluid are computed and the corrections of velocity and pressure field with respect to the body-movement information, gained from LIGGGHTS, are incorporated.

Code of this solver contributions by Alice Hager, JKU.

Algorithm:

For each time step ...

- the motion of the spheres is calculated (position, velocity, angular velocity, force...) with LIGGGHTS using the velocity and pressure-field from the previous time step (initial condition for t=0).
- the Navier-Stokes equations are solved on the whole computational domain, disregarding the solid phase.
- the spheres are located within the mesh: each sphere is represented by a cluster of cells, which are either totally or partially covered by the body, depending on its exact position.
- the correction of the velocity and pressure field of the fluid phase takes place, using the information about the location of the spheres and their (angular) velocity.

Use:

The solver is realized within the Open Source framework CFDEMcoupling. Just as for the unresolved CFD-DEM solver cfdemSolverPiso the file CFD/constant/couplingProperties contains information about the settings for the different models. While IOmodel, DataExchangeModel etc. are applicable for all CFDEMcoupling-solvers, special locate-, force- and void fraction models were designed for the present case:

engineSearchIB, ArchimedesIB, ShirgaonkarIB, IBVoidfraction

References:

GONIVA, C., KLOSS, C., HAGER,A., WIERINK, G. and PIRKER, S. (2011): "A MULTI-PURPOSE OPEN SOURCE CFD-DEM APPROACH", Proc. of the 8th Int. Conf. on CFD in Oil and Gas, Metallurgical and Process Industries, Trondheim, Norway

and

HAGER, A., KLOSS, C. and GONIVA, C. (2011): "TOWARDS AN EFFICIENT IMMERSED BOUNDARY METHOD WITHIN AN OPEN SOURCE FRAMEWORK", Proc. of the 8th Int. Conf. on CFD in Oil and Gas, Metallurgical and Process Industries, Trondheim, Norway

(*) This offering is not approved or endorsed by OpenCFD Limited, the producer of the OpenFOAM software and owner of the OPENFOAMÂ® and OpenCFDÂ® trade marks.

cfdemSolverPiso command

Description:

"cfdemSolverPiso" is a coupled CFD-DEM solver using CFDEMcoupling, an open source parallel coupled CFD-DEM framework. Based on pisoFoam(R)(*), a finite volume based solver for turbulent Navier-Stokes equations applying the PISO algorithm, "cfdemSolverPiso" has additional functionality for a coupling to the DEM code "LIGGGHTS". The volume averaged Navier-Stokes Equations are solved accounting for momentum exchange and volume displacement of discrete particles whose trajectories are calculated in the DEM code LIGGGHTS.

see:

GONIVA, C., KLOSS, C., HAGER,A. and PIRKER, S. (2010): "An Open Source CFD-DEM Perspective", Proc. of OpenFOAM Workshop, GÃ¶teborg, June 22.-24.

---

cfdemSolverPisoScalar command

Description:

"cfdemSolverPisoScalar" is a coupled CFD-DEM solver using CFDEMcoupling, an open source parallel coupled CFD-DEM framework. Based on pisoFoam(R)(*), a finite volume based solver for turbulent Navier-Stokes equations applying PISO algorithm, "cfdemSolverPisoScalar" has additional functionality for a coupling to the DEM code "LIGGGHTS" as well as a scalar transport equation. The volume averaged Navier-Stokes Equations are solved accounting for momentum exchange and volume displacement of discrete particles, whose trajectories are calculated in the DEM code LIGGGHTS. The scalar transport equation is coupled to scalar properties of the particle phase, thus convective heat transfer in a fluid granular system can be modeled with "cfdemSolverPisoScalar".

see:

GONIVA, C., KLOSS, C., HAGER,A. and PIRKER, S. (2010): "An Open Source CFD-DEM Perspective", Proc. of OpenFOAM Workshop, GÃ¶teborg, June 22.-24.

The heat transfer equation is implemented according to Nield & Bejan (2013), Convection in Porous Media, DOI 10.1007/978-1-4614-5541-7_2, Springer

---

clockModel command

Syntax:

Defined in couplingProperties dictionary.

```
clockModel model;
```

- model = name of the clockModel to be applied

Examples:

```
clockModel standardClock;
```

Note: This examples list might not be complete - please look for other models (clockModel_XY) in this documentation.

Description:

The clockModel is the base class for models to examine the code/algorithm with respect to run time.

Main parts of the clockModel classes are written by Josef Kerbl, JKU.

Restrictions: none.

Default: none.

clockModel_noClock command

Syntax:

Defined in couplingProperties dictionary.

```
clockModel off;
```

Examples:

```
clockModel off;
```

Description:

The "noClock" model is a dummy clockModel model which does not measure/evaluate the run time.

Restrictions: none.

Related commands:

clockModel

clockModel_standardClock command

Syntax:

Defined in couplingProperties dictionary.

```
clockModel standardClock;
```

Examples:

```
clockModel standardClock;
```

Description:

The "standardClock" model is a basic clockModel model which measures the run time between every ".start(int arrayPos,string name)" and ".stop(string name)" statement placed in the code. If a ".start(name)" is called more than once (e.g. in a loop) the accumulated times are calculated. After the simulation has finished, the data is stored in $caseDir/CFD/clockData/$startTime/*.txt . Since the measurements are stored in an array, it is necessary to put a variable *arrayPos* (type integer) at the start command. Those do not need to be in ascending order and positions may be omitted. The standard size of this array is 30 and can be changed at the initialization of the standardClock class. If *arrayPos* is out of bounds, the array size will be doubled. The stop command does not need *arrayPos*, since the class remembers the positions. The string name is intended for easier evaluation afterwards an may be omitted like ".start(int arrayPos)" and ".stop()". The command ".stop(string name)" is a safety feature, because if the name is not equal to the started name, output will be produced for information. After the case ran you may use the matPlot.py script located in $CFDEM_UT_DIR/vizClock/ to produce a graphical output of your measurements. The usage is like 'python < matPlot.py' and you have to be in the directory of the desired time step, where there is a file called "timeEvalFull.txt", which contains averaged and maximum data with respect to the number of processes. There is an alias called "vizClock" to run this python routine for visualizing the data.

Restrictions: none.

Related commands:

clockModel

dataExchangeModel command

Syntax:

Defined in couplingProperties dictionary.

```
dataExchangeModel model;
```

- model = name of data exchange model to be applied

Examples:

```
dataExchangeModel twoWayFiles;
dataExchangeModel twoWayMPI;
```

Note: This examples list might not be complete - please look for other models (dataExchangeModel_XY) in this documentation.

Description:

The data exchange model performs the data exchange between the DEM code and the CFD code.

Restrictions:

None.

Related commands:

noDataExchange, oneWayVTK, twoWayFiles, twoWayMPI

Default: none

dataExchangeModel_noDataExchange command

Syntax:

Defined in couplingProperties dictionary.

```
dataExchangeModel noDataExchange;
```

Examples:

```
dataExchangeModel noDataExchange;
```

Description:

The data exchange model performs the data exchange between the DEM code and the CFD code. The noDataExchange model is a dummy model where no data is exchanged.

Restrictions:

None.

Related commands:

dataExchangeModel

dataExchangeModel_oneWayVTK command

Syntax:

Defined in couplingProperties dictionary.

```
dataExchangeModel oneWayVTK;
oneWayVTKProps
{
    DEMts timeStep;
    relativePath "path";
    couplingFilename "filename";
    maxNumberOfParticles number;
};
```

- *timeStep* = time step size of stored DEM data
- *path* = path to the VTK data files relative do simulation directory
- *filename* = filename of the VTK file series
- *number* = maximum number of particles in DEM simulation

Examples:

```
dataExchangeModel oneWayVTK;
oneWayVTKProps
{
    DEMts 0.0001;
    relativePath "../DEM/post";
    couplingFilename "vtk_out%4.4d.vtk";
    maxNumberOfParticles 30000;
}
```

Description:

The data exchange model performs the data exchange between the DEM code and the CFD code. The oneWayVTK model is a model that can exchange particle properties from DEM to CFD based on previously stored VTK data.

Restrictions:

None.

Related commands:

dataExchangeModel

dataExchangeModel_twoWayFiles command

Syntax:

Defined in couplingProperties dictionary.

```
dataExchangeModel twoWayFiles;
twoWayFilesProps
{
    couplingFilename "filename";
    maxNumberOfParticles scalar1;
    DEMts scalar2;
};
```

- *filename* = filename of the VTK file series
- *scalar1* = maximum number of particles in DEM simulation
- *scalar2* = DEM time step width

Examples:

```
dataExchangeModel twoWayFiles;
twoWayFilesProps
{
    couplingFilename "vtk_out%4.4d.vtk";
    maxNumberOfParticles 30000;
}
```

Description:

The data exchange model performs the data exchange between the DEM code and the CFD code. The twoWayFiles model is a model that can exchange particle properties from DEM to CFD and from CFD to DEM. Data is exchanged via files that are sequentially written/read by the codes.

Restrictions:

Developed only for two processors, one for DEM and one for CFD run.

Related commands:

dataExchangeModel

dataExchangeModel_twoWayMPI command

Syntax:

Defined in couplingProperties dictionary.

```
dataExchangeModel twoWayMPI;
twoWayMPIProps
{
    liggghtsPath "path";
};
```

- *path* = path to the DEM simulation input file

Examples:

```
dataExchangeModel twoWayMPI;
twoWayMPIProps
{
    liggghtsPath "../DEM/in.liggghts_init";
}
```

Description:

The data exchange model performs the data exchange between the DEM code and the CFD code. The twoWayMPI model is a model that can exchange particle properties from DEM to CFD and from CFD to DEM. Data is exchanged via MPI technique. The DEM run is executed by the coupling model, via a liggghtsCommandModel object.

Restrictions:

none.

Related commands:

dataExchangeModel

forceModel_Archimedes command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    Archimedes
);
ArchimedesProps
{
    gravityFieldName "gravity";
    twoDimensional;
    suppressProbe   switch1;
    treatForceDEM   switch2;
};
```

- *gravity* = name of the finite volume gravity field
- *twoDimensional* = optional keyword for conducting a two dimensional calculation
- *switch1* = (optional, default false) can be used to suppress the output of the probe model
- *switch2* = (optional, default true) sub model switch, see forceSubModel for details

Examples:

```
forceModels
(
    Archimedes
);
ArchimedesProps
{
    gravityFieldName "g";
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The Archimedes model is a model that calculates the Archimedes' volumetric lift force stemming from density difference of fluid and particle.

Restrictions:

none.

Related commands:

forceModel

forceModel_ArchimedesIB command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    ArchimedesIB
);
ArchimedesIBProps
{
    gravityFieldName "gravity";
    voidfractionFieldName "voidfraction";
    twoDimensional;
    treatForceExplicit  switch1;
};
```

- *gravity* = name of the finite volume gravity field
- *voidfraction* = name of the finite volume voidfraction field
- *twoDimensional* = optional keyword for conducting a two dimensional calculation
- *switch1* = (optional, default true) sub model switch, see forceSubModel for details

Examples:

```
forceModels
(
    ArchimedesIB
);
ArchimedesIBProps
{
    gravityFieldName "g";
    voidfractionFieldName "voidfractionNext";
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The ArchimedesIB model is a model that calculates the ArchimedesIB' volumetric lift force stemming from density difference of fluid and particle. This model is especially suited for resolved CFD-DEM simulations where the particle is represented by immersed boundary method.

Restrictions:

Only for immersed boundary solvers.

Related commands:

forceModel

forceModel_DiFeliceDrag command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    DiFeliceDrag
);
DiFeliceDragProps
{
    velFieldName "U";
    voidfractionFieldName "voidfraction";
    granVelFieldName "Us";
    interpolation switch1;
    voidfractionInterpolationType "type1";
    UInterpolationType "type2";
    suppressProbe       switch2;
    scale               scalar1;
    scaleDrag           scalar2;
    treatForceExplicit  switch3;
    implForceDEM        switch4;
    verbose             switch5;
    scalarViscosity     switch6;
    nu                  scalar3;
};
```

- *U* = name of the finite volume fluid velocity field
- *voidfraction* = name of the finite volume voidfraction field
- *Us* = name of the finite volume granular velocity field
- *switch1* = (optional, normally off) flag to use interpolated voidfraction and velocity values
- *type1* = (optional, default cellPoint) interpolation type for voidfraction field
- *type2* = (optional, default cellPointFace) interpolation type for velocity field
- *switch2* = (optional, default false) can be used to suppress the output of the probe model
- *scalar1* = (optional) scaling of particle diameter
- *scalar2* = (optional) scaling of drag law
- *switch3* = sub model switch, see forceSubModel for details
- *switch4* = sub model switch, see forceSubModel for details
- *switch5* = sub model switch, see forceSubModel for details
- *switch6* = sub model switch, see forceSubModel for details
- *scalar3* = optional, only if switch6 is true

Examples:

```
forceModels
(
    DiFeliceDrag
);
DiFeliceDragProps
{
    velFieldName "U";
    interpolation true;
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The DiFeliceDrag model is a model that calculates the particle based drag force following the correlation of Di Felice (see Zhou et al. (2010), JFM).

Restrictions:

none.

Related commands:

forceModel

forceModel_fieldStore command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    fieldStore
);
fieldStoreProps
{
    scalarFieldNames
    (
        "scalarField"
    );
    vectorFieldNames
    (
        "vectorField"
    );
};
```

- *scalarField* = names of the finite volume scalar fields to be stored
- *vectorField* = names of the finite volume vector fields to be stored

Examples:

```
forceModels
(
    fieldStore
);
fieldStoreProps
{
    scalarFieldNames
    (
        "voidfraction"
    );
    vectorFieldNames
    (
        "U"
    );
}
```

Description:

This "forceModel" does not influence the particles or the flow - it is a tool to store a scalar/vector field! This is especially useful if you use a boundary condition which cannot interpreted correctly in your postporcessor (e.g. paraview).

Restrictions:

none.

Related commands:

forceModel

forceModel_fieldTimeAverage command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    fieldTimeAverage
);
fieldTimeAverageProps
{
    startTime time;
    scalarFieldNames
    (
        "scalarField"
    );
    vectorFieldNames
    (
        "vectorField"
    );
};
```

- *time* = (optional) time to start temporal averaging
- *scalarField* = names of the finite volume scalar fields to be temporally averaged
- *vectorField* = names of the finite volume vector fields to be temporally averaged

Examples:

```
forceModels
(
    fieldTimeAverage
);
fieldTimeAverageProps
{
    startTime 1.0;
    scalarFieldNames
    (
        "voidfraction"
    );
    vectorFieldNames
    (
        "Us"
    );
}
```

Description:

This "forceModel" does not influence the particles or the simulation - it is a postprocessing tool! Starting at start time the specified fields are temporally averaged and written at "writeTime". They can then be probed using standard function object probes. The output name is timeAverage_scalarField, where scalarField is the name of the original field.

Restrictions:

none.

Related commands:

forceModel

forceModel_GidaspowDrag command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    GidaspowDrag
);
GidaspowDragProps
{
    velFieldName "U";
    voidfractionFieldName "voidfraction";
    granVelFieldName "Us";
    phi                 scalar1;
    interpolation switch1;
    voidfractionInterpolationType "type1"
    UInterpolationType "type2"
    implForceDEM        switch2;
    suppressProbe       switch3;
    scale               scalar2;
    scaleDrag           scalar3;
    switchingVoidfraction scalar4;
    treatForceExplicit  switch4;
    implForceDEM        switch5;
    verbose             switch6;
    scalarViscosity     switch7;
    nu                  scalar5;
};
```

- *U* = name of the finite volume fluid velocity field
- *voidfraction* = name of the finite volume voidfraction field
- *Us* = name of the finite volume cell averaged particle velocity field
- *scalar1* = drag correction factor (in doubt 1)
- *switch1* = (optional, default off) flag to use interpolated voidfraction and fluid velocity values
- *type1* = (optional, default cellPoint) interpolation type for voidfraction field
- *type2* = (optional, default cellPointFace) interpolation type for velocity field
- *switch2* = (optional, default false) flag to use implicit formulation of drag on DEM side:l *switch3* = (optional, default false) can be used to suppress the output of the probe model
- *scalar2* = (optional) scaling of particle diameter
- *scalar3* = (optional) scaling of drag law
- *scalar4* = (optional) voidfraction above which dilute formulation will be used
- *switch4* = (optional, default false) sub model switch, see forceSubModel for details
- *switch5* = (optional, default false) sub model switch, see forceSubModel for details
- *switch6* = (optional, default false) sub model switch, see forceSubModel for details
- *switch7* = (optional, default false) sub model switch, see forceSubModel for details
- *scalar5* = (optional, default false) optional, only if switch6 is true

Examples:

```
forceModels
(
```

```
    GidaspowDrag
);
GidaspowDragProps
{
    velFieldName "U";
    voidfractionFieldName "voidfraction";
    granVelFieldName "Us";
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The GidaspowDrag model is a model that calculates the particle based drag force following the correlation of Gidaspow which is a combination of Ergun (1952) and Wen & Yu (1966) (see Zhu et al. (2007): "Discrete particle simulation of particulate systems: Theoretical developments", ChemEngScience).

Restrictions:

none.

Related commands:

forceModel

forceModel_gradPForce command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    gradPForce;
);
gradPForceProps
{
    pFieldName "pressure";
    velocityFieldName "U";
    interpolation       switch1;
    gradPInterpolationType "type1"
    useAddedMass        scalar1;
    suppressProbe       switch2;
    treatForceExplicit  switch3;
    treatForceDEM       switch4;
};
```

- *pressure* = name of the finite volume fluid pressure field
- *U* = name of the finite volume fluid velocity field
- *switch1* = flag to use interpolated pressure values (normally off)
- *type1* = (optional, default cellPointFace) interpolation type for grad(p) field
- *useAddedMass* = (optional) coefficient of added mass accounted for
- *switch2* = (optional, default false) can be used to suppress the output of the probe model
- *switch3* = (optional, default true) sub model switch, see forceSubModel for details
- *switch4* = (optional, default false) sub model switch, see forceSubModel for details

Examples:

```
forceModels
(
    gradPForce;
);
gradPForceProps
{
    pFieldName "p";
    velocityFieldName "U";
    interpolation true;
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The gradPForce model is a model that calculates the particle based pressure gradient force -(grad(p)) * Vparticle (see Zhou et al. (2010): "Discrete particle simulation of particle-fluid flow: model formulations and their applicability" ,JFM).

Restrictions:

none.

Related commands:

forceModel

forceModel command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    model_x
    model_y
);
```

- model = name of force model to be applied

Examples:

```
forceModels
(
    Archimedes
    DiFeliceDrag
);
```

Note: This examples list might not be complete - please look for other models (forceModel_XY) in this documentation.

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. All force models selected are executed sequentially and the forces on the particles are superposed. If the fluid density field is needed, by default a field named "rho" will be used. Via the forceSubModel an alternative field can be chosen.

Restrictions:

None.

Related commands:

Archimedes, DiFeliceDrag, gradPForce, viscForce

Note: This examples list may be incomplete - please look for other models (forceModel_XY) in this documentation.

Default: none.

forceModel_KochHillDrag command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    KochHillDrag
);
KochHillDragProps
{
    velFieldName "U";
    voidfractionFieldName "voidfraction";
    granVelFieldName "Us"
    interpolation       "switch1";
    voidfractionInterpolationType "type1"
    UInterpolationType  "type2"
    implForceDEM        "switch2";
    suppressProbe       "switch3";
    scale               "scalar1";
    scaleDrag           "scalar2";
    treatForceExplicit  "switch4";
    verbose             "switch5";
    implForceDEMaccumulated "switch6";
    scalarViscosity     "switch7";
    nu                  "scalar3";
};
```

- *U* = name of the finite volume fluid velocity field
- *voidfraction* = name of the finite volume voidfraction field
- *Us* = (optional) name of finite volume granular velocity field
- *switch1* = (optional, normally off) flag to use interpolated voidfraction and fluid velocity values
- *type1* = (optional, default cellPoint) interpolation type for voidfraction field
- *type2* = (optional, default cellPointFace) interpolation type for velocity field
- *switch2* = (optional, normally off) flag to use implicit formulation of drag on DEM side:l *switch3* = (optional, default false) can be used to suppress the output of the probe model
- *scalar1* = (optional) scaling of particle diameter
- *scalar2* = (optional) scaling of drag law
- *switch4* = (optional, default false) sub model switch, see forceSubModel for details
- *switch5* = (optional, default false) sub model switch, see forceSubModel for details
- *switch6* = (optional, default false) sub model switch, see forceSubModel for details
- *switch7* = (optional, default false) sub model switch, see forceSubModel for details
- *scalar3* = optional, only if switch7 is true

Examples:

```
forceModels
(
    KochHillDrag
);
KochHillDragProps
{
    velFieldName "U";
```

```
    voidfractionFieldName "voidfraction";
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The KochHillDrag model is a model that calculates the particle based drag force following the correlation of Koch & Hill (2001) (see van Buijtenen et al. (2011): "Numerical and experimental study on multiple-spout fluidized beds", ChemEngScience).

Restrictions:

none.

Related commands:

forceModel

forceModel_LaEuScalarTemp command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    LaEuScalarTemp
);
LaEuScalarTempProps
{
    velFieldName "U";
    tempFieldName "T";
    voidfractionFieldName "voidfraction";
    partTempName "Temp";
    partHeatFluxName "convectiveHeatFlux";
    partHeatTransCoeffName "heatTransCoeff";
    partHeatFluidName "heatFluid";
    lambda scalar1;
    Cp scalar2;
    interpolation switch1;
    TInterpolationType "type1"
    verbose switch2;
    maxSource scalar3;
    scale scalar4;
    scalarViscosity switch3;
    nu scalar5;
};
```

- *U* = name of the finite volume fluid velocity field
- *T* = name of the finite volume scalar temperature field
- *voidfraction* = name of the finite volume voidfraction field
- *Temp* = name of the DEM data representing the particles temperature
- *convectiveHeatFlux* = name of the DEM data representing the particle-fluid convective heat flux
- *heatTransCoeff* = name of heat transfer coefficient
- *heatFluid* = *scalar1* = fluid thermal conductivity [W/(m*K)]
- *scalar2* = fluid specific heat capacity [W*s/(kg*K)]
- *switch1* = (optional, normally off) flag to use interpolated voidfraction and fluid velocity values
- *type1* = (optional, default cellPoint) interpolation type for T field
- *switch2* = (optional, default false) sub model switch, see forceSubModel for details
- *scalar3* = (optional) limit maximal turbulence
- *scalar4* = scaling of particle diameter
- *switch3* = (optional, default false) sub model switch, see forceSubModel for details
- *scalar5* = optional, only if switch3 is true

Examples:

```
forceModels
(
    LaEuScalarTemp
);
LaEuScalarTempProps
{
```

```
    velFieldName "U";
    tempFieldName "T";
    voidfractionFieldName "voidfraction";
    partTempName "Temp";
    partHeatFluxName "convectiveHeatFlux";
    lambda 0.0256;
    Cp 1007;
}
```

Description:

This "forceModel" does not influence the particles or the fluid flow! Using the particles' temperature a scalar field representing "particle-fluid heatflux" is calculated. The solver then uses this source field in the scalar transport equation for the temperature. The model for convective heat transfer is based on Li and Mason (2000), A computational investigation of transient heat transfer in pneumatic transport of granular particles, Pow.Tech 112

Restrictions:

Goes only with cfdemSolverScalar. The force model has to be the second (!!!) model in the forces list.

Related commands:

forceModel

forceModel_MeiLift command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    MeiLift
);
MeiLiftProps
{
    velFieldName "U";
    useSecondOrderTerms;
    interpolation switch1;
    vorticityInterpolationType "type1"
    UInterpolationType "type2"
    verbose switch2;
    treatForceExplicit switch3;
    scalarViscosity switch4;
    nu scalar1;
};
```

- *U* = name of the finite volume fluid velocity field
- *useSecondOrderTerms* = switch to activate second order terms in the lift force model
- *switch1* = switch to activate tri-linear interpolation of the flow quantities at the particle position
- *type1* = (optional, default cellPoint) interpolation type for vorticity field
- *type2* = (optional, default cellPointFace) interpolation type for velocity field
- *switch2* = switch to activate the report of per-particle quantities to the screen
- *switch3* = (optional, default true) sub model switch, see forceSubModel for details
- *switch4* = (optional, default false) sub model switch, see forceSubModel for details
- *scalar1* = optional, only if switch4 is true

Examples:

```
forceModels
(
    MeiLift
);
MeiLiftProps
{
    velFieldName "U";
    useSecondOrderTerms;
    interpolation true;
    verbose true;
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The MeiLift model calculates the lift force for each particle based on Loth and Dorgan (2009). In case the keyword "useSecondOrderTerms" is not specified, this lift force model uses the expression of McLaughlin (1991, Journal of Fluid Mechanics 224:261-274).

Restrictions:

None.

Related commands:

forceModel

forceModel_noDrag command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    noDrag
);
```

noDragProps { noDEMForce; keepCFDForce; treatForceExplicit switch1; }

- noDragProps are optional. *noDEMForce* = optional
- *keepCFDForce* = optional
- *switch1* = (optional, default false) sub model switch, see forceSubModel for details

Examples:

```
forceModels
(
    noDrag
);

noDragProps (optional)
{
    noDEMForce; (optional, default false) do not apply the previously calculated forces in DEM in
    keepCFDForce; (optional, default false) do not delete the previously calculated forces and us
};
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The noDrag model sets the forces acting on the particle (which were previously caclulated) to zero. If several force models are selected and noDrag is the last model being executed, the fluid particle force will be set to zero. If the variable noDEMForce is set, then the forces communicated to the DEM solver are also set to zero.

Restrictions:

None.

Related commands:

forceModel

forceModel_particleCellVolume command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    particleCellVolume
);
particleCellVolumeProps
{
    verbose switch1;
    writeToFile switch2;
    startTime number1;
    upperThreshold number2;
    lowerThreshold number3;
    verbose;
};
```

- *switch1* = (optional, default true) switch for output to screen
- *switch2* = (optional, default false) switch for output to file
- *number1* = (optional, default 0) start time of volume calculation and output
- *number2* = only cells with a field value (magnitude) lower than this upper threshold are considered
- *number3* = only cells with a field value (magnitude) greater than this lower threshold are considered

Examples:

```
forceModels
(
    particleCellVolume
);
particleCellVolumeProps
{
    upperThreshold 0.999;
    lowerThreshold 0;
}
```

Description:

This "forceModel" does not influence the particles or the simulation - it is a postprocessing tool! The total volume of the particles as they are represented on the CFD mesh is calculated. Further the total volume of the cells particles are in is calculated. At "writeTime" a field named particleCellVolume , where scalarField is the name of the original field, is written. This can then be probed using standard function object probes. Analogously a field named cellVolume is written. Using the verbose option a screen output is given.

Restrictions:

None.

Related commands:

forceModel

forceModel_particleVolume command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    particleVolume
);
particleVolumeProps
{
    verbose switch1;
    writeToFile switch2;
    scale scalar1;
    startTime scalar2;
};
```

- *switch1* = (optional, default false) switch for output to screen
- *switch2* = (optional, default true) switch for output to file
- *scalar1* = (optional, default 1) scaling of the particle volume d=dSphere/scale
- *scalar2* = (optional, default 0) start time of volume calculation and output

Examples:

```
forceModels
(
    particleVolume
);
particleVolumeProps
{
    writeToFile false;
}
```

Description:

This "forceModel" does not influence the particles or the simulation - it is a postprocessing tool! The total volume of the particles is calculated.

Restrictions:

None.

Related commands:

forceModel

forceModel_scalarGeneralExchange command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    scalarGeneralExchange  // must be 2nd position!
);
scalarGeneralExchangeProps
{
    useLiMason "switch1"; //default: DeenEtAl
    useGeneralCorrelation "switch3"; //default: DeenEtAl
    generalCorrelationParameters (1 2 3 4 5 6 7 8);
    verbose "switch2";
    velFieldName "U";
    voidfractionFieldName "voidfraction";
    tempFieldName "T";
    partTempName "Temp";
    /* partHeatFluxName "convectiveHeatFlux"; //switch off for implicit coupling, e.g., to ParSca
    partHeatTransCoeffName "heatTransCoeff";
    partHeatFluidName "heatFluid";
    lambda value;
    Cp value1;
    //Lists with information for each species FOR THE PARTICLES
    //MUST be in the same order as eulerian species in 'scalarTransportProperties'
    //MUST correspond to property/atom in ligghts (use 'couple/cfd/speciesConvection' to auto-gen
    partSpeciesNames
    (
    speciesC
    );
    partSpeciesFluxNames
    (
    speciesCFlux
    );
    partSpeciesTransCoeffNames
    (
    speciesCTransCoeff
    );
    partSpeciesFluidNames
    (
    speciesCFluid
    );
    DMolecular
    (
    value2
    );
    interpolation "bool1";
    voidfractionInterpolationType "type1"
    UInterpolationType "type2"
    fluidScalarFieldInterpolationType "type2"
    scalarViscosity switch5;
    nu scalar5;
    suppressProbe switch6;
    scale scalar6;
    maxSource scalar7;
}
```

- *switch1* = (optional) flag to use Nusselt correlations of Li and Mason (2000)
- *switch2* = (normally off) for verbose run
- *switch3* = (optional) flag to use a general Nusselt number correlation (must specify parameters of this correlation in a list called 'generalCorrelationParameters')
- generalCorrelationParameters = list with a predefined number of parameters (for length see src code, only read if useGeneralCorrelation is set to true)
- *U* = name of the finite volume fluid velocity field
- *voidfraction* = name of the finite volume voidfraction field
- *T* = name of the finite volume scalar temperature field
- *Temp* = name of the DEM data representing the particles temperature
- *convectiveHeatFlux* = name of the DEM data representing the particle-fluid convective heat flux
- *heatTransCoeff* = name of the DEM data representing the particle-fluid heat transfer coefficient
- *heatFluid* = name of the DEM data representing the fluid heat
- *value* = fluid thermal conductivity [W/(m*K)]
- *value1* = fluid specific heat capacity [W*s/(kg*K)]
- *speciesC* = name of the DEM data representing the transport species of the particles
- *speciesCFlux* = name of the DEM data representing the particle-fluid species flux
- *speciesCTransCoeff* = name of the DEM data representing the particle-fluid species transfer coefficient
- *speciesCFluid* = name of the DEM data representing the transport species of the fluid
- *value2* = molecular diffusion coefficient [m^2/s]
- *bool1* = (optional, normally off) flag to use interpolated voidfraction and fluid velocity values
- *type1* = (optional, default cellPoint) interpolation type for voidfraction field
- *type2* = (optional, default cellPointFace) interpolation type for velocity field
- *type3* = (optional, default cellPoint) interpolation type for fluidScalarField field
- *switch5* = (optional, default false) sub model switch, see forceSubModel for details
- *scalar5* = (optional) optional, only if switch5 is true
- *switch6* = (optional, default false) can be used to suppress the output of the probe model
- *scalar7* = (optional) scaling of particle diameter
- *scalar7* = limit maximal turbulence

Examples:

```
forceModels
(
    scalarGeneralExchange  // must be 2nd position!
);
scalarGeneralExchangeProps
{
    useLiMason false; //default: DeenEtAl
    useGeneralCorrelation true; //default: DeenEtAl
    generalCorrelationParameters
    (
     7.0 -10 5
     1.0 0.17
     1.33 -2.31 1.16
    );
    verbose false;
    velFieldName "U";
    voidfractionFieldName "voidfraction";
    tempFieldName "T";
    partTempName "Temp";
    /* partHeatFluxName "convectiveHeatFlux"; //switch off for implicit coupling, e.g., to ParSca
    partHeatTransCoeffName "heatTransCoeff";
    partHeatFluidName "heatFluid";
```

```
    lambda 0.0271;
    Cp 1007;
    //Lists with information for each species FOR THE PARTICLES
    //MUST be in the same order as eulerian species in 'scalarTransportProperties'
    //MUST correspond to property/atom in ligghts (use 'couple/cfd/speciesConvection' to auto-gen
    partSpeciesNames
    (
    speciesC
    );
    partSpeciesFluxNames
    (
    speciesCFlux
    );
    partSpeciesTransCoeffNames
    (
    speciesCTransCoeff
    );
    partSpeciesFluidNames
    (
    speciesCFluid
    );
    DMolecular
    (
    1e-5
    );
}
```

Description:

This "forceModel" does not influence the particles or the fluid flow! Using the particles' temperature and/or species a scalar field representing "particle-fluid heatflux" and/or "particle-fluid speciesflux" is calculated.

This code is designed to realize coupled CFD-DEM simulations using LIGGGHTS and OpenFOAM(R). Note: this code is not part of OpenFOAM(R) (see DISCLAIMER).

Two way general scalar exchange between DEM and CFD convective heat and species transfer model. The standard model is that of Deen, N.G. et al., Review of direct numerical simulation of fluid-particle mass, momentum and heat transfer in dense gas-solid flows. Chemical Engineering Science 116 (2014) 710-724. This correlation is based on that of Gunn (1978).

The switch 'useGeneralCorrelation' allows one to specify the parameters of the Gunn correlation as a list called 'generalCorrelationParameters'.

Alternatively, the correclation of Li and Mason (2000), A computational investigation of transient heat transfer in pneumatic transport of granular particles, Pow.Tech 112 can be activated. However, this correlation is not suitable for dense granular flows.

WARNING: This model REQUIRES the 'generalManual' speciesTransportModel

Restrictions:

Goes only with cfdemSolverPimpleImEx and cfdemSolverPisoSTM. The force model has to be the second (!!!) model in the forces list.

Related commands:

forceModel forceModel_LaEuScalarTemp

forceModel_SchillerNaumannDrag command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    SchillerNaumannDrag
);
SchillerNaumannDragProps
{
    velFieldName "U";
    voidfractionFieldName "voidfraction";
    interpolation "bool1";
    voidfractionInterpolationType "type1"
    UInterpolationType "type2"
    implForceDEM "bool2";
};
```

- *U* = name of the finite volume fluid velocity field
- *voidfraction* = name of the finite volume voidfraction field
- *bool1* = (optional, normally off) flag to use interpolated voidfraction and fluid velocity values
- *type1* = (optional, default cellPoint) interpolation type for voidfraction field
- *type2* = (optional, default cellPointFace) interpolation type for velocity field *bool2* = (optional, normally off) flag to use implicit formulation of drag on DEM side:l

Examples:

```
forceModels
(
    SchillerNaumannDrag
);
SchillerNaumannDragProps
{
    velFieldName "U";
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The SchillerNaumannDrag model is a model that calculates the particle based drag force following the correlation of Schiller and Naumann.

Restrictions:

none.

Related commands:

forceModel

forceModel_ShirgaonkarIB command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    ShirgaonkarIB
);
ShirgaonkarIBProps
{
    velFieldName "U";
    pressureFieldName "p";
    twoDimensional;
    depth scalar1;
    verbose switch1;
    treatForceExplicit switch2;
};
```

- *U* = name of the finite volume fluid velocity field
- *p* = name of the finite volume pressure field
- *twoDimensional* = optional keyword for conducting a two dimensional calculation
- *scalar1* = optional, only necessary if twoDimensional is active
- *switch1* = (optional, default false) sub model switch, see forceSubModel for details
- *switch2* = (optional, default false) sub model switch, see forceSubModel for details

Examples:

```
forceModels
(
    ShirgaonkarIB
);
ShirgaonkarIBProps
{
    velFieldName "U";
    pressureFieldName "p";
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The ShirgaonkarIB model calculates the drag force (viscous and pressure force) acting on each particle in a resolved manner (see Shirgaonkar et al. (2009): "A new mathematical formulation and fast algorithm for fully resolved simulation of self-propulsion", Journal of Comp. Physics). This model is only suited for resolved CFD-DEM simulations where the particle is represented by immersed boundary method.

References:

SHIRGAONKAR, A.A., MACIVER, M.A. and PATANKAR, N.A., (2009), â A new mathematical formulation and fast algorithm for fully resolved simulation of self-propulsionâ , J. Comput. Phys., 228, 2366-2390.

Restrictions:

Only for immersed boundary solvers.

Related commands:

forceModel

forceModel_virtualMassForce command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    virtualMassForce
);
virtualMassForceProps
{
    velFieldName "U";
    phiFieldName "phi";
    splitUrelCalculation switch1;
    Cadd scalar1;
    treatForceExplicit switch2;
    treatForceDEM switch3;
    interpolation switch4;
    UInterpolationType "type1"
    DDtUInterpolationType "type2"
};
```

- *U* = name of the finite volume fluid velocity field
- *phi* = name of the finite volume flux field
- *switch1* = indicator to split calculation of Urel between CFDEM and LIGGGHTS
- *scalar1* = scalar value
- *switch2* = (optional, default true) sub model switch, see forceSubModel for details
- *switch3* = (optional, default false) sub model switch, see forceSubModel for details
- *switch4* = (optional, default false) sub model switch, see forceSubModel for details
- *type1* = (optional, default cellPointFace) interpolation type for U field
- *type2* = (optional, default cellPointFace) interpolation type for ddt(U) field

Examples:

```
forceModels
(
    virtualMassForce
);
virtualMassForceProps
{
    velFieldName "U";
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The virtualMassForce model calculates the virtual mass force for each particle.

Restrictions:

Model not validated!

Related commands:

forceModel

forceModel_viscForce command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    viscForce;
);
viscForceProps
{
    velocityFieldName "U";
    useAddedMass scalar1;
    suppressProbe switch1;
    treatForceExplicit switch2;
    treatForceDEM switch3;
    interpolation switch4;
    divTauInterpolationType "type1";
    scalarViscosity switch5;
    nu scalar2;
};
```

- *U* = name of the finite volume fluid velocity field
- *scalar1* = (optional) coefficient of added mass accounted for
- *switch1* = (optional, default false) can be used to suppress the output of the probe model
- *switch2* = (optional, default true) sub model switch, see forceSubModel for details
- *switch3* = (optional, default false) sub model switch, see forceSubModel for details
- *switch4* = (optional, default false) sub model switch, see forceSubModel for details
- *type1* = (optional, default cellPointFace) interpolation type for div(Tau) field
- *switch5* = (optional, default false) sub model switch, see forceSubModel for details
- *scalar2* = optional, only if switch5 is true

Examples:

```
forceModels
(
    viscForce;
);
viscForceProps
{
    velocityFieldName "U";
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The viscForce model calculates the particle based viscous force, -(grad(tau)) * Vparticle (see Zhou et al. (2010): "Discrete particle simulation of particle-fluid flow: model formulations and their applicability", JFM).

Restrictions:

none.

Related commands:

forceModel

forceModel_volWeightedAverage command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    volWeightedAverage
);
volWeightedAverageProps
{
    startTime time;
    scalarFieldNames
    (
        scalarField
    );
    vectorFieldNames
    (
        vectorField
    );
    upperThreshold scalar1;
    lowerThreshold scalar2;
    useVolumeFraction switch0;
    volumeFractionName word1;
    verbose ;
    writeToFile switch1;
};
```

- *time* = (optional, default 0.) time to start the averaging
- *scalarField* = names of the finite volume scalar fields to be temporally averaged
- *vectorField* = names of the finite volume vector fields to be temporally averaged
- *scalar1* = only cells with a field value (magnitude) lower than this upper threshold are considered
- *scalar2* = only cells with a field value (magnitude) greater than this lower threshold are considered
- *switch0* = (optional, default false) consider a volume fraction for the calculation
- *word1* = (optional, default "voidfraction") name of the volume fraction, only used if useVolumeFraction is true
- *verbose* = (optional, default false) keyword only (mostly used for debugging)
- *switch1* = (optional, default false) switch for the output.

Examples:

```
forceModels
(
    volWeightedAverage
);
volWeightedAverageProps
{
    startTime 0.1;
    scalarFieldNames
    (
        voidfraction
    );
    vectorFieldNames
    (
```

```
    );
    upperThreshold 0.999;
    lowerThreshold 0;
}
```

Description:

This "forceModel" does not influence the particles or the simulation - it is a postprocessing tool! Starting at start time the volume weighted averages of those cells of the fields within the threshold are calculated. At "writeTime" a field named volAverage_field , where scalarField is the name of the original field, is written. This can then be probed using standard function object probes.

Restrictions:

Currently all fields have the same threshold value!

Related commands:

forceModel

forceSubModel command

Syntax:

Defined in couplingProperties sub-dictionary of the force model in use. If no force sub-model is applied ImEx is used as default. If the keyword "forceSubModels" is provided, a choice of sub model is demanded.

```
forceSubModels
(
    model_x
    model_y
);
```

- model = name of force sub-model to be applied

Examples:

```
forceSubModels
(
    ImEx
);
```

Note: This examples list might not be complete - please look for other models (forceSubModel_XY) in this documentation.

Description:

The force sub model is designed to hold the settings a force model can have. For now it handles the treatExplicit, treatDEM and implDEM option.

Restrictions:

None.

Related commands:

ImEx

Note: This examples list may be incomplete - please look for other models (forceSubModel_XY) in this documentation.

Default: none.

Swtiches:

- Depending on the availability within the respective force model, a number of switches can be activated:
- treatForceExplicit: switch for the purely explicit consideration of the force term in the equation of motion on the CFD side (if switched off, the force is considered semi-implicitly)

- treatForceDEM: switch for the consideration of the forces on the DEM side only
- implForceDEM: If true, the fluid velocity and drag coefficient are communicated to the DEM calculation at each coupling time step and the drag force is calculated at each DEM time step, using the current particle velocity. If false, a force term is communiated to the DEM calculation at each coupling time step, the term is not within a coupling interval.
- verbose: switch for debug output to screen
- interpolation: switch for the usage of interpolation models when getting data for the Lagrangian calculation from Eulerian fields; If false, the cell centre values are used.
- useFilteredDragModel: switch for using a coarse-grid version of the Beetstra drag model (takes grid-size effects into account; default = off)
- useParcelSizeDependentFilteredDrag: switch for using a coarse-grid version of the Beetstra drag model (takes parcel-size effects into account, will force the switch useFilteredDragModel to "on"; default = off)
- implForceDEMaccumulated: Can only be used in combination with implForceDEM switch, drag force values of each DEM time step are accumulated and passed on to the CFD-calculation.
- scalarViscosity: switch for the usage of a user-defined viscosity nu for the calculation of the drag force; The CFD calculation always uses the value of the transport model.

forceSubModel_ImEx command

Syntax:

Defined in couplingProperties sub-dictionary of the force model in use.

forceSubModels (ImEx;);

treatExplicit true; // optional for some force models. treatDEM true; // optional for some force models. implDEM true; // optional for some force models.

Examples:

forceSubModels (ImEx;); treatExplicit true; // optional for some force models.

Description:

If no force sub-model is applied ImEx is used as default. If the keyword "forceSubModels" is provided, a choice of sub model is demanded. Depending on the force model different keywords are read and can therefrore be set (see the log file). If the keyword is provided, its value is used.

Restrictions:

none.

Related commands:

forceSubModel

githubAccess_public

Description:

This routine describes how to pull repositories of the CFDEM(R)project from github.com. After setting some environment variables, LIGGGHTS(R) and CFDEM(R)coupling can be compiled.

Procedure:

Basically the following steps have to be performed:

- *git clone* and setup OpenFOAM
- *git clone* the desired repositories
- update your repositories by *git pull*
- set environment variables
- compile LIGGGHTS(R) and CFDEM(R)coupling
- run your own cases

***git clone* and setup OpenFOAM:**

Have a look at the latest compatible OpenFOAM(R)-version in the versionInfo.H file at github. This file will later be downloaded as a part of the source-code. Look for the git commit hashtag in the following line:

```
word OFversion="<OF-Release>-commit-<commitHashtag>";

e.g. word OFversion="2.4.x-commit-3d8da0e960c717ff582f1517a27724144f086b83";
```

However sometimes even newer versions are supported, please check the release notes and the "Advanced Settings"-section.

Basically follow the OpenFOAM(R) git compilation instructions, with a small number of exceptions:

When you git clone the repository, replace the release-version with <OF-Release>.

with git-protocol:

```
git clone git://github.com/OpenFOAM/OpenFOAM-<OF-Release>.git
```

or with https:

```
git clone https://github.com/OpenFOAM/OpenFOAM-<OF-Release>.git
```

Now change into the new directory and checkout the correct compatible version:

```
cd OpenFOAM-<OF-Release>
git checkout <commitHashtag>
```

The result will be a status report, that indicates a 'detached head state'. Now continue with installing and compiling OpenFOAM(R). Make sure that OpenFOAM(R) works properly with a parallel Simulation!

If you want to use an older OpenFOAM(R)-version, please have a look at the "Backwards Compatibility"-section.

***git clone* the desired repositories:**

You may want to take a look around on CFDEMproject on github: github.com/CFDEMproject_gitCFDEM

If not already done, open a terminal and create a directory for LIGGGHTS(R) in $HOME:

```
cd
mkdir LIGGGHTS
cd LIGGGHTS
```

To clone the public LIGGGHTS repository, open a terminal and execute: with git-protocol:

```
git clone git://github.com/CFDEMproject/LIGGGHTS-PUBLIC.git LIGGGHTS-PUBLIC
```

or with https:

```
git clone https://github.com/CFDEMproject/LIGGGHTS-PUBLIC.git LIGGGHTS-PUBLIC
```

If not already done, open a terminal and create a directory for CFDEM(R)coupling in $HOME:

```
cd
mkdir CFDEM
cd CFDEM
```

To clone the public CFDEM(R)coupling repository, open a terminal and execute: with git-protocol:

```
git clone git://github.com/CFDEMproject/CFDEMcoupling-PUBLIC.git CFDEMcoupling-PUBLIC-$WM_PROJECT
```

or with https:

```
git clone https://github.com/CFDEMcoupling-PUBLIC.git CFDEMcoupling-PUBLIC-$WM_PROJECT_VERSION

Additionally the lpp tool for converting LIGGGHTS dump-files into the paraview readable VTK-forma
with git-protocol:
git clone git://github.com:CFDEMproject/LPP.git $HOME/LIGGGHTS/mylpp

or with https:
git clone https://github.com:CFDEMproject/LPP.git $HOME/LIGGGHTS/mylpp
```

Please have a look at README and INSTALL.txt in the root directory of LPP for further information.

Troubles? See Troubleshooting git section below.

Update your repositories by *git pull*:

To get the latest version, open a terminal, go to the location of your local installation and type: *Warning: git stash will remove your changes in $HOME/CFDEM/CFDEMcoupling-PUBLIC-$WM_PROJECT_VERSION !*

```
cd $HOME/CFDEM/CFDEMcoupling-PUBLIC-$WM_PROJECT_VERSION
git stash
```

```
git pull
```

Set Environment Variables:

Now you need to set some environment variables in ~/.bashrc (if you use c-shell, manipulate ~/.cshrc accordingly). Open ~/.bashrc

```
gedit ~/.bashrc &
```

add the lines (you find them also in CFDEMCoupling-$WM_PROJECT_VERSION/src/lagrangian/cfdemParticle/etc/bashrc and cshrc respectively):

```
#============================================#
#- source cfdem env vars
export CFDEM_VERSION=PUBLIC
export CFDEM_PROJECT_DIR=$HOME/CFDEM/CFDEMcoupling-$CFDEM_VERSION-$WM_PROJECT_VERSION
export CFDEM_SRC_DIR=$CFDEM_PROJECT_DIR/src
export CFDEM_SOLVER_DIR=$CFDEM_PROJECT_DIR/applications/solvers
export CFDEM_DOC_DIR=$CFDEM_PROJECT_DIR/doc
export CFDEM_UT_DIR=$CFDEM_PROJECT_DIR/applications/utilities
export CFDEM_TUT_DIR=$CFDEM_PROJECT_DIR/tutorials
export CFDEM_PROJECT_USER_DIR=$HOME/CFDEM/$LOGNAME-$CFDEM_VERSION-$WM_PROJECT_VERSION
export CFDEM_bashrc=$CFDEM_SRC_DIR/lagrangian/cfdemParticle/etc/bashrc
export CFDEM_LIGGGHTS_SRC_DIR=$HOME/LIGGGHTS/LIGGGHTS-PUBLIC/src
export CFDEM_LIGGGHTS_MAKEFILE_NAME=fedora_fpic
export CFDEM_LPP_DIR=$HOME/LIGGGHTS/mylpp/src
export CFDEM_PIZZA_DIR=$HOME/LIGGGHTS/PIZZA/gran_pizza_17Aug10/src
. $CFDEM_bashrc
#============================================#
```

If you installed LIGGGHTS(R) or CFDEM(R)coupling in non-standard paths, please have a look at least at CFDEM_PROJECT_DIR and CFDEM_LIGGGHTS_SRC_DIR. The standard CFDEM_LIGGGHTS_MAKEFILE_NAME is fedora_fpic, which works on most systems. However please checkout LIGGGHTS-PUBLIC/src/MAKE for additional makefiles, wich are available. The most used ones are fedora_fpic and ubuntuVTK_fpic. Beware that the CFDEMcoupling needs a fpic compilation to use LIGGGHTS as a library. Please check the "Advanced Settings" for VTK information.

Save the ~/.bashrc, open a new terminal and test the settings. The commands:

```
$CFDEM_PROJECT_DIR
$CFDEM_SRC_DIR
$CFDEM_LIGGGHTS_SRC_DIR
```

should give "...: is a directory" otherwise something went wrong and the environment variables in ~/.bashrc are not set correctly.

To specify the paths of pizza, please check the settings in $CFDEM_SRC_DIR/lagrangian/cfdemParticle/etc/bashrc.

If $CFDEM_SRC_DIR is set correctly, you can type

```
cfdemSysTest
```

to get some information if the paths are set correctly.

Compile LIGGGHTS(R) and CFDEM(R)coupling:

If above settings were done correctly, you can compile LIGGGHTS(R) by typing:

```
cfdemCompLIG
```

and you can then compile CFDEM(R)coupling by typing:

```
cfdemCompCFDEM
```

or compile both at once with:

```
cfdemCompCFDEMall
```

You can run the tutorial cases by executing .../etc/testTutorial.sh through the alias *cfdemTestTUT*. Alternatively you can run each tutorial using the *Allrun.sh* scripts in the tutorial directories.

In case questions concerning the installation arise, please feel free to contact our forum at www.cfdem.com.

Run Your Own Cases:

If you want to run your own cases, please do so in $CFDEM_PROJECT_USER_DIR/run which is automatically being generated. E.g. copy one of the tutorial cases there, adapt it to your needs. Changes in $CFDEM_TUT_DIR will be lost after every *git stash*!

Additional Installations:

Optionally you can install lpp which will help you convert the DEM (dump) data to VTK format. For standard CFD-DEM runs this will not be necessary. To get the DEM postprocessing tool "lpp" you need python-numpy package installed:

```
sudo apt-get install python-numpy
```

You can pull the latest version of lpp with:

```
cd $HOME/LIGGGHTS
```

with git-protocol: git clone git://github.com/CFDEMproject/LPP.git with https: git clone https://github.com/CFDEMproject/LPP.git

Backwards Compatibility:

Basically CFDEM(R)coupling supports one OpenFOAM(R) version therefore all settings are prepared for that. Nevertheless we try to maintain backwards compatibility as long as it works with reasonable effort.

The supported OpenFOAM(R) and LIGGGHTS(R) versions are stated in: src/lagrangian/cfdemParticle/cfdTools/versionInfo.H

For using other versions you can manipulate: src/lagrangian/cfdemParticle/etc/OFversion/OFversion.H (still not all functionality might work then!)

Advanced Settings:

Here some advanced settings and hints for non-standard compilations are presented. As stated in the "Backwards Compatibility"-section, there are compiler flags for different OpenFOAM versions. Checkout src/lagrangian/cfdemParticle/etc/OFversion/OFversion.H for compatibility settings. Just comment the current "#define version2X" in the top-section and uncomment the one you want to compile it with.

There are advanced compilation settings for library-paths, includes and libraries are within the additionalLibs file in src/lagrangian/cfdemParticle/etc/additionalLibs. There are predefined files for different OpenFOAM versions. To use a different version, add the following lines to your .bashrc (.cshrc) before the standard CFDEM variables:

```
export CFDEM_ADD_LIBS_DIR=FOLDER_OF_NEW_additionalLibs_FILE/

export CFDEM_ADD_LIBS_NAME=additionalLibs30x
```

This is an example to use a predefined additionalLibs file for OpenFOAM-3.0.x.

To enable direct VTK-dump (dump custom/vtk) support of LIGGGHTS and CFDEMcoupling, you have to install the VTK libraries. Either 5.8 or 6 are predefined for ubuntu.

```
sudo apt-get libvtk5.8 libvtk5-dev
```

Change "export CFDEM_LIGGGHTS_MAKEFILE_NAME=fedora_fpic" in your .bashrc according to your preferred LIGGGHTS makefile. If you have a non-standard installation location you have to adapt the LIGGGHTS makefile accordingly. To enable this feature in a coupled run the additionalLibs file has to be modified. It basically needs to include the same libraries as the LIGGGHTS-Makefile. E.g. for Ubuntu-14.04 with vtk-5.8:

```
CFDEM_ADD_LIB_PATHS  = -L/usr/include/vtk-5.8

CFDEM_ADD_LIBS       = -lvtkCommon -lvtkFiltering -lvtkIO
```

Troubleshooting git:

- Troubles with git clone?

a) The git protocol will not work if your computer is behind a firewall which blocks the relevant TCP port, you can use alternatively https instead of git (write command in one line):

git clone https://github.com/CFDEMproject/CFDEMcoupling-PUBLIC.git CFDEMcoupling-PUBLIC-$WM_PROJECT_VERSION

b) If you face the error: "error: SSL certificate problem, verify that the CA cert is OK. Details: error:14090086:SSL routines:SSL3_GET_SERVER_CERTIFICATE:certificate verify failed while accessing https://github.com/...",

please use: env GIT_SSL_NO_VERIFY=true git clone https://github...

(see http://stackoverflow.com/questions/3777075/https-github-access)

c) If you face the error: "Agent admitted failure to sign using the key. Permission denied (publickey).", after ssh -T git@github.com

please type: "ssh-add"

(see: https://help.github.com/articles/error-agent-admitted-failure-to-sign)

IOModel_basicIO command

Syntax:

Defined in couplingProperties dictionary.

```
IOModel "basicIO";
```

Examples:

```
IOModel "basicIO";
```

Description:

The basic IO-model writes particle positions velocities and radii to files. The default output directory ($casePath/CFD/proc*/time/lagrangian). Using the keyword "serialOutput;" in couplingProperties the IO is serial to the directory ($casePath/CFD/lagrangian). In the latter case only the data on processor 0 is written! Data is written every write time of the CFD simulation.

Restrictions: None.

Related commands:

IOModel

IOModel command

Syntax:

Defined in couplingProperties dictionary.

```
IOModel "model";
```

- model = name of IO-model to be applied

Examples:

IOModel "off";

Note: This examples list might not be complete - please look for other models (IOModel_XY) in this documentation.

Description:

The IO-model is the base class to write data (e.g. particle properties) to files.

Restrictions:

none.

Related commands:

Note: This examples list may be incomplete - please look for other models (IOModel_XY) in this documentation.

Default: none.

IOModel_noIO command

Syntax:

Defined in couplingProperties dictionary.

```
IOModel "off";
```

Examples:

```
IOModel "off";
```

Description:

The noIO-model is a dummy IO model.

Restrictions: None.

Related commands:

IOModel

IOModel_sophIO command

Syntax:

Defined in couplingProperties dictionary.

```
IOModel "sophIO";
```

Examples:

```
IOModel "sophIO";
```

Description:

The sophIO-model is based on basicIO model and additionally writes voidfraction, implicit forces, explicit forces. Data is written every write time of the CFD simulation.

Restrictions: None.

Related commands:

IOModel

IOModel_trackIO command

Syntax:

Defined in couplingProperties dictionary.

```
IOModel "trackIO";
```

Examples:

```
IOModel "trackIO";
```

Description:

The trackIO-model is based on sophIO model and additionally writes fields necessary to use the particleTracks utility (which needs a particleTrackProperties file in the constant dir). The particleTracks utility generates tracks of the particles and writes them to a vtk file.

Restrictions: None.

Related commands:

IOModel

liggghtsCommandModel_execute command

Syntax:

Defined in liggghtsCommmands dictionary.

```
liggghtsCommandModels
(
   execute
);
executeProps0
{
    command
    (
        run
        $couplingInterval
    );
    runFirst switch1;
    runLast switch2;
    runEveryCouplingStep switch3;
    runEveryWriteStep switch4;
    verbose;
}
```

- *command* = LIGGGHTS command to be executed. Each word in a new line, numbers and symbols need special treatment (e.g. $couplingInterval will be replaced by correct coupling interval in the simulation)
- *switch1* = switch (choose on/off) if the command is executed only at first time step
- *switch2* = switch (choose on/off) if the command is executed only at last time step
- *switch3* = switch (choose on/off) if the command is executed at every coupling step
- *switch4* = switch (choose on/off) if the command is executed at every writing step
- *verbose* = (normally off) for verbose run

Examples:

```
liggghtsCommandModels
(
   execute
   execute
);
executeProps0
{
    command
    (
        run
        $couplingInterval
    );
    runFirst off;
    runLast off;
    runEveryCouplingStep on;
}
executeProps1
{
    command
    (
```

```
        write_restart
        noBlanks
        dotdot
        slash
        DEM
        slash
        liggghts.restart_
        timeStamp
    );
    runFirst off;
    runLast off;
    runEveryCouplingStep off;
    runEveryWriteStep on;
}
```

Description:

The execute liggghtsCommand Model can be used to execute a LIGGGHTS command during a CFD run. In above example execute_0 for instance executes "run $couplingInterval" every coupling step. $couplingInterval is automatically replaced by the correct number of DEM steps. Additionally execute_1 executes "write_restart ../DEM/liggghts.restart_$timeStamp" every writing step, where $timeStamp is automatically set.

These rather complex execute commands can be replaced by the "readLiggghts" and "writeLiggghts" commands!

Restrictions: None.

Related commands:

liggghtsCommandModel

liggghtsCommandModel command

Syntax:

Defined in liggghtsCommmands dictionary.

```
liggghtsCommandModels
(
    model_x
    model_y
);
```

- model = name of the liggghtsCommandModel to be applied

Examples:

```
liggghtsCommandModels
(
   runLiggghts
   writeLiggghts
);
```

Note: This examples list might not be complete - please look for other models (liggghtsCommandModel_XY) in this documentation.

Description:

The liggghtsCommandModel is the base class to execute DEM commands within a CFD run.

Restrictions:

Works only with MPI coupling.

Default: none.

liggghtsCommandModel_readLiggghtsData command

Syntax:

Defined in liggghtsCommmands dictionary.

```
liggghtsCommandModels
(
   readLiggghtsData
);
readLiggghtsDataProps0
{
    startIndex "scalar1";
    verbose ;
    exactTiming ;
    filePath
    (
        "word"
    );
    runFirst "bool1";
    runEveryCouplingStep "bool2";
    startTime "scalar2";
    endTime "scalar3";
    timeInterval "scalar4";
}
```

- *scalar1* = start index of data file to be read
- *verbose* = (default off) flag for verbose run
- *exactTiming* = flag indicating that start time should be kept even during a coupling interval
- *filePath* = path to LIGGGHTS data file. Each word in a new line, numbers and symbols need special treatment (e.g. $couplingInterval will be replaced by correct coupling interval in the simulation)
- *bool1* = true if to be run at first timestep only (prio 1)
- *bool2* = true if to be run at every coupling step (prio 2)
- *scalar2* = if bool2 and bool3 false then starts at scalar2 (prio 3) run
- *scalar3* = if bool2 and bool3 false then it ends at scalar3 (prio 3) run
- *scalar4* = if bool2 and bool3 false then it repeats at scalar3 increasing the data file index (prio 3) run

Examples:

```
liggghtsCommandModels
(
   readLiggghtsData
);
readLiggghtsDataProps0
{
    startIndex 0;
    exactTiming true;
    filePath
    (
        dotdot
        slash
        DEM
        slash
        packing.data
    );
```

```
    runFirst off;
    runEveryCouplingStep off;
    startTime 0.002;
    endTime 0.012;
    timeInterval 0.001;
}
```

Description:

The readLiggghtsData liggghtsCommand Model can be used to read liggghts data files into liggghts during runtime of a coupled simulation.

Restrictions:

Note: Model is not up to date.

Related commands:

liggghtsCommandModel

liggghtsCommandModel_runLiggghts command

Syntax:

Defined in liggghtsCommmands dictionary.

```
liggghtsCommandModels
(
   runLiggghts
);
//- optional
runLiggghtsProps
{
    preNo true;
    verbose; (optional)
}
```

Examples:

```
liggghtsCommandModels
(
   runLiggghts
);
```

Description:

The liggghtsCommand models can be used to execute a LIGGGHTS command during a CFD run. The "runLiggghts" command executes the command "run $nrDEMsteps", where $nrDEMsteps is automatically set according to the coupling intervals, every coupling step. Optionally a dictionary called runLiggghtsProps can be specified where the "preNo" switch can be set, which uses the command "run $nrDEMsteps pre no" for every time step except the first.

Restrictions: Warning: the "pre no" option can cause troubles (dump data of particles changing the domain might be erroneous)!

Related commands:

liggghtsCommandModel

liggghtsCommandModel_writeLiggghts command

Syntax:

Defined in liggghtsCommmands dictionary.

```
liggghtsCommandModels
(
   writeLiggghts
);
//- optional
writeLiggghtsProps
{
    writeLast switch1;
    path "path";
    writeName "name";
    overwrite switch2;
    verbose;
}
```

- *switch1* = switch (choose on/off) to select if only last step is stored or every write step (default on).
- *path* = optionally an alternative path (relative to execution directory) for saving the restart file can be defined (default "../DEM").
- *name* = name of the restart file to be written in /$caseDir/DEM/ default (default "liggghts.restartCFDEM")
- *switch2* = switch (choose on/off) to select if only one restart file $name or many files $name_$timeStamp are written (default off):l *verbose* = (default off) for verbose run

Examples:

```
liggghtsCommandModels
(
   runLiggghts
   writeLiggghts
);
```

Description:

The liggghtsCommand models can be used to execute a LIGGGHTS command during a CFD write. The "writeLiggghts" command executes the command "write_restart $name", where $name is the name of the restart file, every write step.

Restrictions: None.

Related commands:

liggghtsCommandModel

locateModel_engineSearch command

Syntax:

Defined in couplingProperties dictionary.

```
locateModel engine;
engineProps
{
    treeSearch switch1;
}
```

- *switch1* = switch to use tree search algorithm

Examples:

```
locateModel engine;
engineProps
{
    treeSearch true;
}
```

Description:

The locateModel "engine" locates the CFD cell and cellID corresponding to a given position. The engineSearch locate Model can be used with different settings to use different algorithms:

- treeSearch false; will execute some geometric (linear) search using the last known cellID
- treeSearch true; will use a recursive tree structure to find the cell (recommended).

Restrictions: none.

Related commands:

locateModel

locateModel_engineSearchIB command

Syntax:

Defined in couplingProperties dictionary.

```
locateModel engineIB;
engineIBProps
{
    engineProps
    {
        treeSearch switch1;
    }
    zSplit value1;
    xySplit value2;
    checkPeriodicCells;
}
```

- *switch1* = names of the finite volume scalar fields to be temporally averaged
- *value1* = number of z-normal layers for satellite points
- *value2* = number of satellite points in each layer
- *checkPeriodicCells* = (optional, default false) flag for considering the minimal distance to all periodic images of this particle

Examples:

```
locateModel engineIB;
engineIBProps
{
    engineProps
    {
        treeSearch false;
    }
    zSplit 8;
    xySplit 16;
}
```

Description:

The locateModel "engine" locates the CFD cell and cellID corresponding to a given position. This locate model is especially designed for parallel immersed boundary method. Each particle is represented by "satellite points" if it is distributed over several processors.

The engineSearchIB locate Model can be used with different settings to use different algorithms:

- treeSearch false; will execute some geometric (linear) search using the last known cellID (recommended)
- treeSearch true; will use a recursive tree structure to find the cell.

This model is a modification of the engine search model. Instead of using the centre-cell as starting point for the engine search, further satellite points located on the surface of the sphere are checked. This ensures that (parts of) spheres can be located even when their centre is on another processor. This is especially important

for parallel computations, when a sphere is about to move from one processor to another.

Restrictions:

Only for immersed boundary solvers!

Related commands:

locateModel

locateModel command

Syntax:

Defined in couplingProperties dictionary.

```
locateModel model;
```

- model = name of the locateModel to be applied

Examples:

```
locateModel engine;
```

Note: This examples list might not be complete - please look for other models (locateModel_XY) in this documentation.

Description:

The locateModel is the base class for models which search for the CFD cell and cellID corresponding to a position. In general it is used to find the cell a particle is located in.

Restrictions: none.

Default: none.

locateModel_standardSearch command

Syntax:

Defined in couplingProperties dictionary.

```
locateModel standard;
```

Examples:

```
locateModel standard;
```

Description:

The locateModel "standard" locates the CFD cell and cellID corresponding to a given position. A very straight-forward (robust!) locate algorithm is used.

Restrictions: none.

Related commands:

locateModel

meshMotionModel command

Syntax:

Defined in couplingProperties dictionary.

```
meshMotionModel model;
```

- model = name of the meshMotionModel to be applied

Examples:

```
meshMotionModel noMeshMotion;
```

Note: This examples list might not be complete - please look for other models (meshMotionModel_XY) in this documentation.

Description:

The meshMotionModel is the base class for models which manipulate the CFD mesh according to the DEM mesh motion.

Restrictions: none.

Default: none.

meshMotionModel_noMeshMotion command

Syntax:

Defined in couplingProperties dictionary.

```
meshMotionModel noMeshMotion;
```

Examples:

```
meshMotionModel noMeshMotion;
```

Description:

The noMeshMotion-model is a dummy meshMotion model.

Restrictions: None.

Related commands:

meshMotionModel

momCoupleModel_explicitCouple command

Syntax:

Defined in couplingProperties dictionary.

```
momCoupleModels
(
    explicitCouple
);
explicitCoupleProps
{
    fLimit vector;
}
```

- *vector* = limiter vector for explicit force term (default (1e10,1e10,1e10))

Examples:

```
momCoupleModels
(
    explicitCouple
);
explicitCoupleProps
{
    fLimit (1e3 1e2 1e4);
}
```

Description:

The explicitCouple-model is a momCoupleModel model providing an explicit momentum source term for the CFD solver and additionally it superposes an additional source field which can be set via the function setSourceField.

Restrictions:

Only for solvers that include explicit momentum exchange.

Related commands:

momCoupleModel

momCoupleModel command

Syntax:

Defined in couplingProperties dictionary.

```
momCoupleModels
(
    model
);
```

- model = name of the momCoupleModel to be applied

Examples:

```
momCoupleModels
(
    implicitCouple
);
```

Note: This examples list might not be complete - please look for other models (momCoupleModel_XY) in this documentation.

Forces can be coupled in an implicit way to the fluid solver (i.e., when solving the Navier-Stokes equations, the fluid velocity at the new time will be considered for the coupling force). This implicit coupling is typically done for the drag forces (look for "impForces()" in the implementation of the drag model). Implicit coupling is more stable (especially important for dense flows), but conflicts Newton's third law. Explicit forces are imposed on the flow solver in an explicit fashion (look for "expForces()" in the implementation of the drag model), which is less stable, but does not conflict Newton's third law.

Note that the variable "imExSplitFactor" can be set in the couplingProperties in order to treat implicitly defined forces (in the implementation of the force model) as explicit ones. "imExSplitFactor 1.0;" is set by default, meaning that all implicit forces will be considered implicitly, whereas "imExSplitFactor 0.0;" would mean that implicitly defined forces will be treated in an explicit fashion.

Note that the switch "treatVoidCellsAsExplicitForce true;" can be set in the couplingProperties in order to change the treatment of cells which are void of particles. This is only relevant if (i) smoothing is used, and (ii) implicit force coupling is performed. By default, the particle veloctiy field (Us) will be smoothed to obtain a meaningful reference quantity for the implicit force coupling. In case "treatVoidCellsAsExplicitForce true;" is set, however, Us will not be smoothed and implicit forces (after the smoothing has been performed) in cells void of particles be treated as explicit ones. This avoids the problem of defining Us in cells that are void of particles, but for which an implicit coupling force is obtained in the smoothing process. **Description:**

The momCoupleModel is the base class for momentum exchange between DEM and CFD simulation.

Restrictions: none.

Default: none.

momCoupleModel_implicitCouple command

Syntax:

Defined in couplingProperties dictionary.

```
momCoupleModels
(
    implicitCouple
);
implicitCoupleProps
{
    velFieldName "U";
    granVelFieldName "Us";
    voidfractionFieldName "voidfraction";
    minAlphaP number;
}
```

- *U* = name of the finite volume fluid velocity field
- *Us* = name of the finite volume granular velocity field
- *voidfraction* = name of the finite volume voidfraction field *number* = minimum value for local particle volume fraction to calculate the exchange filed (default SMALL):l

Examples:

```
momCoupleModels
(
    implicitCouple
);
implicitCoupleProps
{
    velFieldName "U";
    granVelFieldName "Us";
    voidfractionFieldName "voidfraction";
}
```

Description:

The implicitCouple-model is a momCoupleModel model providing an implicit momentum source term for the CFD solver.

Restrictions:

Only for solvers that include implicit momentum exchange.

Related commands:

momCoupleModel

momCoupleModel_noCouple command

Syntax:

Defined in couplingProperties dictionary.

```
momCoupleModels
(
    off
);
```

Examples:

```
momCoupleModels
(
    off
);
```

Description:

The noCouple-model is a dummy momCoupleModel model providing a no momentum source term for the CFD solver.

Restrictions:

Only for solvers that include no momentum exchange, e.g. immersed boundary.

Related commands:

momCoupleModel

probeModel command

Syntax:

To be activated via couplingProperties dictionary.

```
probeModel myProbeModel;
```

Use probe model "off" to disable this feature.

```
myProbeModelProps

{

};
```

Examples:

See particleProbe

Note: This examples list might not be complete - please check below for the list of force models that can perform particle probing.

Description:

The probeModel feature allows to implement various probing features in CFDEM. Currently, only the particleProbe model is implemented, that performs probing of particle forces.

Restrictions:

None.

Default: none.

probeModel_noProbe command

Syntax:

To be activated via couplingProperties dictionary.

```
forceModels
{
    myForceModel1
    myForceModel2
    myForceModel3
};
```

probeModel off;

Examples:

```
probeModel off;
```

Note: This examples list might not be complete - please check below for the list of force models that can perform particle probing.

Description:

Does not perform any probing.

Restrictions:

None.

Related commands which are currently enabled for particle probing:

particleProbe

Default: none.

probeModel_particleProbe command

Syntax:

To be activated via couplingProperties dictionary.

forceModels (myForceModel1 myForceModel2 myForceModel3);

probeModel particleProbe;

```
particleProbeProps
{
    particleIDsToSample (ID1 ID2 ID3 ...);  //list of particleIDs to sample
    verboseToFile;                          //main switch
    verbose;                                //currently not used
    printEvery       xEvery;                //print every this many CFD time steps
    printOnlyAtStep  xStep;                 //print only at this CFD time step (overrides "printE
    sampleAll;                              //Activate sampling for all particles
    probeDebug;                             //probes additional fields
    includePosition;                        //will include particle position in the output file
    writePrecision xPrecision;              //number of significant digits to print
};
```

- *particleIDsToSample* = list of particle IDs to be sampled.
- *myForceModeli* = list of force models in the simulation, the particleProbe will be applied to all of these models!
- *verboseToFile* = main switch to activate the particle probe (default = off).
- *verbose* = main switch to activate output to Info (currently not implemented).
- *xEvery* = integer to specify the interval for sampling (default = 1, i.e., probing occurs every CFD time step).
- *xStep* = integer to specify the step for sampling (default = deactivated, i.e., it will print accordingly to "printEvery").
- *sampleAll* = switch to activate sampling of all particles. Otherwise (default) only particles specified via "particleIDsToSample" in the couplingProperties dictionary will be sampled.
- *probeDebug* = switch to activate probing of debug properties of secondary importance (specific for each force model).
- *includePosition* = switch to add the particle position in the log file (default = off).
- *xPrecision*= number of significant digits of the text output (default = 3).

Examples:

```
particleIDsToSample (0 1 2 3);
forceModels
(
    gradPForce
);
particleProbeProps
{
    verboseToFile;       //main switch
    verbose;               //currently not used
    printEvery  100;      //print every this many CFD time steps
    sampleAll;            //Activate sampling for all particles
    probeDebug;  //probes additional fields
```

```
    includePosition;  //will include particle position in the output file
    writePrecision 4;           //number of significant digits to print
};
```

Note: This examples list might not be complete - please check below for the list of force models that can perform particle probing.

Description:

The particleProbe feature keeps track of per-particle quantities (e.g., the fluid-particle interaction forces) acting on each DEM particle, and handles its storage during the simulation. Data is saved in the CFD/particleProbes/startTime directory, where startTime is the time at which the simulation is started (this avoids unwanted deletion of particleProbe data).

Restrictions:

You can manually switch off the probe model for each force model by specifying the Switch "suppressProbe" in the corresponding force properties dictionary.

Related commands which are currently enabled for particle probing:

gradPForce, viscForce, BeetstraDrag, HollowayDrag, MeiLift, as well as most other forceModels (see src directory for details, i.e., use "grep -r 'probeM(' ./" in the terminal).

Default: none.

regionModel_allRegion command

Syntax:

Note: In the current CFDEMcoupling version, this model is no longer used. Defined in couplingProperties dictionary.

```
regionModel allRegion;
```

Examples:

```
regionModel allRegion;
```

Description:

The allRegion-model is a region model including the whole CFD region for the coupling.

Restrictions: None.

Related commands:

regionModel

regionModel command

Syntax:

Note: In the current CFDEMcoupling version, this model is no longer used. Defined in couplingProperties dictionary.

```
regionModel model;
```

- model = name of the regionModel to be applied

Examples:

```
regionModel allRegion;
```

Note: This examples list might not be complete - please look for other models (regionModel_XY) in this documentation.

Description:

The regionModel is the base class for region models to select a certain region for coupled simulation.

Restrictions: none.

Default: none.

smoothingModel_constDiffSmoothing command

Syntax:

Defined in couplingProperties dictionary.

```
smoothingModel constDiffSmoothing;
constDiffSmoothingProps
{
    lowerLimit number1;
    upperLimit number2;
    smoothingLength lengthScale;
    smoothingLengthReferenceField lengthScaleRefField;
    verbose;
}
```

- *number1* = scalar fields will be bound to this lower value
- *number2* = scalar fields will be bound to this upper value
- *lengthScale* = length scale over which the exchange fields will be smoothed out
- *lengthScaleRefField* = length scale over which reference fields (e.g., the average particle velocity) will be smoothed out. Should be always larger than lengthScale. If not specified, will be equal to lengthScale.
- *verbose* = (optional, default false) flag for debugging output

Examples:

```
constDiffSmoothingProps
{
    lowerLimit 0.1;
    upperLimit 1e10;
    smoothingLength 1500e-6;
    smoothingLengthReferenceField 9000e-6;
}
```

Description:

The "constDiffSmoothing" model is a basic smoothingModel model which reads a smoothing length scale being used for smoothing the exchange fields (voidfraction, Ksl, f if present). This model can be used for smoothing explicit force coupling fields, as well as implicit force coupling algorithms. Smoothing for reference fields is performed to "fill in" values in cells in which these reference fields are not specified. Values calculated in the cells (via Lagrangian-To-Euler mapping) are NOT changed! These reference fields are, e.g., the average particle velocity, which are not specified in all cells in case the flow is rather dilute.

Restrictions: This model is tested in a limited number of flow situations.

ATTENTION: In case a smoothing model is used in conjunction with "PimpleImEx" solvers, the fields "f" and "fSmooth" must be placed in the initial time directory! This is because zeroGradient boundary conditions for the fields "f" and "fSmooth" must be specified, otherwise the smoothing operation will give an Error.

Related commands:

smoothingModel

smoothingModel command

Syntax:

Defined in couplingProperties dictionary.

```
smoothingModel model;
```

- model = name of the smoothingModel to be applied

Examples:

```
smoothingModel off;

smoothingModel  constDiffSmoothing;

smoothingModel  localPSizeDiffSmoothing;
```

Note: This examples list might not be complete - please look for other models (smoothingModel_XY) in this documentation.

ATTENTION: In case a smoothing model is used in conjunction with "PimpleImEx" solvers, the fields "f" and "fSmooth" must be placed in the initial time directory! This is because zeroGradient boundary conditions for the fields "f" and "fSmooth" must be specified, otherwise the smoothing operation will give an Error.

Description:

The smoothingModel is the base class for models that smoothen the exchange fields (i.e., voidfraction and the Ksl field in case of implicit force coupling). This is relevant in case one uses a small grid resolution compared to the local particle diameter (or parcel diameter in case one uses a parcel approach).

Restrictions: These models are in beta testing.

Default: none.

smoothingModel_noSmoothing command

Syntax:

Defined in couplingProperties dictionary.

```
smoothingModel off;
```

Examples:

```
smoothingModel off;
```

Description:

The "noSmoothing" model is a dummy smoothingModel model which does no smoothing.

Restrictions: none.

Related commands:

smoothingModel

voidfractionModel_bigParticleVoidFraction command

Syntax:

Defined in couplingProperties dictionary.

```
voidfractionModel bigParticle;
bigParticleProps
{
    maxCellsPerParticle number1;
    alphaMin number2;
    weight number3;
    porosity number4;
}
```

- *number1* = maximum number of cells covered by a particle (search will fail when more than *number1* cells are covered by the particle)
- *number2* = minimum limit for voidfraction
- *number3* = (optional) scaling of the particle volume to account for porosity or agglomerations.
- *number4* = (optional) diameter of the particle's representation is artificially increased according to *number2* * Vparticle, volume remains unaltered!

Examples:

```
voidfractionModel bigParticle;
bigParticleProps
{
    maxCellsPerParticle 1000;
    alphaMin 0.10;
    weight 1.;
    porosity 5.0;
}
```

Description:

The bigParticle voidFraction model is supposed to be used when a particle (or its representation) is bigger than a CFD cell. The voidfraction field is set in those cell whose centres are inside the particle which results in a stairstep representation of the bodies within the mesh (i.e. voidfraction is either 1 (fluid) of zero (solid)). For archiving accurate results, approx. 8 cells per particle diameter are necessary.

The region of influence of a particle can be increased artificially by "porosity", which blows up the particles, but keeps their volume (for voidfraction calculation) constant.

The particle volume occupied in the CFD domain can be adjusted by the parameter "weight", using Vparticle=dsphere^3*pi/6*weight.

Parts of this sub-model contributed by Alice Hager, JKU.

Restrictions: none.

Related commands:

voidfractionModel

voidfractionModel_centreVoidFraction command

Syntax:

Defined in couplingProperties dictionary.

```
voidfractionModel centre;
centreProps
{
    alphaMin number1;
    weight number2;
}
```

- *number1* = minimum limit for voidfraction
- *number2* = (optional) scaling of the particle volume to account for porosity or agglomerations.

Examples:

```
voidfractionModel centre;
centreProps
{
    alphaMin 0.1;
    weight 1.;
}
```

Description:

The centre voidFraction model calculates the voidfraction in a CFD cell accounting for the volume of the particles whose centres are inside the cell.

The particle volume occupied in the CFD domain can be adjusted by the parameter "weight", using Vparticle=dsphere^3*pi/6*weight.

Restrictions: none.

Related commands:

voidfractionModel

voidfractionModel_dividedVoidFraction command

Syntax:

Defined in couplingProperties dictionary.

```
voidfractionModel divided;
dividedProps
{
    alphaMin number1;
    interpolation;
    weight number2;
    porosity number3;
    verbose;
    cfdemUseOnly;
}
```

- *number1* = minimum limit for voidfraction
- *interpolation* = flag to interpolate voidfraction to particle positions (normally off)
- *number2* = (optional) scaling of the particle volume to account for porosity or agglomerations.
- *number3* = (optional) diameter of the particle's representation is artificially increased according to *number2* * Vparticle, volume remains unaltered!
- *verbose* = (optional, default false) flag for debugging output
- *cfdemUseOnly* = optional flag, default false

Examples:

```
voidfractionModel divided;
dividedProps
{
    alphaMin 0.2;
}
```

Description:

The divided voidFraction model is supposed to be used when a particle (or its representation) is in the size range of a CFD cell. Satellite points are used to divide the particle's volume to the touched cells.

The region of influence of a particle can be increased artificially by "porosity", which blows up the particles, but keeps their volume (for voidfraction calculation) constant.

The particle volume occupied in the CFD domain can be adjusted by the parameter "weight", using Vparticle=dsphere^3*pi/6*weight.

In the basic implementation of solvers, the void fraction is calculated based on all particles. Depending on the solver used, the void fraction calculation is also performed for a certain type of particles. The void fraction calculation is based on a three-step approach (reset, set and interpolate), i.e., the void fraction is time interpolated from a previous and a next void fraction field. Appropriate names for these fields have to be specified in the sub-dictionaries voidFracFieldNamesPrev and voidFracFieldNamesNext in the couplingProperties dictionary.

Restrictions: none.

Related commands:

voidfractionModel

voidfractionModel_GaussVoidFraction command

Syntax:

Defined in couplingProperties dictionary.

```
voidfractionModel Gauss;
GaussProps
{
    maxCellsPerParticle number1;
    alphaMin number2;
    weight number3;
    porosity number4;
}
```

- *number1* = maximum number of cells covered by a particle (search will fail when more than *number1* cells are covered by the particle)
- *number2* = minimum limit for voidfraction
- *number3* = (optional) scaling of the particle volume to account for porosity or agglomerations.
- *number4* = (optional) diameter of the particle's representation is artificially increased according to *number2* * Vparticle, volume remains unaltered!

Examples:

```
voidfractionModel Gauss;
GaussProps
{
    maxCellsPerParticle 1000;
    alphaMin 0.10;
    weight 1.;
    porosity 1.;
}
```

Description:

The Gauss voidFraction model is supposed to be used when a particle (or its representation) is bigger than a CFD cell. The voidfraction field is set in those cell whose centres are inside the particle. The volume is here distributed according to a Gaussian distribution.

The region of influence of a particle can be increased artificially by "porosity", which blows up the particles, but keeps their volume (for voidfraction calculation) constant.

The particle volume occupied in the CFD domain can be adjusted by the parameter "weight", using Vparticle=dsphere^3*pi/6*weight.

Restrictions: none.

Related commands:

voidfractionModel , bigParticle

voidfractionModel command

Syntax:

Defined in couplingProperties dictionary.

```
voidfractionModel model;
```

- model = name of the voidfractionModel to be applied

Examples:

```
voidfractionModel centre;
```

Note: This examples list might not be complete - please look for other models (voidfractionModel_XY) in this documentation.

Description:

The voidfractionModel is the base class for models to represent the DEM particle's volume in the CFD domain via a voidfraction field.

Restrictions: none.

Default: none.

voidfractionModel_IBVoidFraction command

Syntax:

Defined in couplingProperties dictionary.

```
voidfractionModel IB;
IBProps
{
    maxCellsPerParticle number1;
    alphaMin number2;
    scaleUpVol number3;
    checkPeriodicCells ;
}
```

- *number1* = maximum number of cells covered by a particle (search will fail when more than *number1* cells are covered by the particle)
- *number2* = minimum limit for voidfraction
- *number3* = diameter of the particle's representation is artificially increased according to *number3* * Vparticle, volume remains unaltered!
- *checkPeriodicCells* = (optional, default false) flag for considering the minimal distance to all periodic images of this particle

Examples:

```
voidfractionModel IB;
IBProps
{
    maxCellsPerParticle 1000;
    alphaMin 0.10;
    scaleUpVol 5.0;
}
```

Description:

The IB voidFraction model is supposed to be used when a particle (or its representation) is bigger than a CFD cell. The voidfraction field is set in those cell whose centres are inside the particle. The model is specially designed for cfdemSolverIB and creates a smooth transition of the voidfraction at the particle surface. Cells which are only partially covered by solid are marked by voidfraction values between 0 and 1 respectively.

The region of influence of a particle can be increased artificially by "scaleUpVol", which blows up the particles, but keeps their volume (for voidfraction calculation) constant.

Code of this sub-model contributed by Alice Hager, JKU.

Restrictions: none.

Related commands:

voidfractionModel

voidfractionModel_noVoidFractionVoidFraction command

Syntax:

Defined in couplingProperties dictionary.

voidfractionModel noVoidFraction;

Examples:

voidfractionModel noVoidFraction;

Description:

The noVoidFraction voidFraction model is a dummy model and has no physical meaning.

Restrictions: none.

Related commands:

voidfractionModel

voidfractionModel_trilinearVoidFraction command

Syntax:

Defined in couplingProperties dictionary.

```
voidfractionModel trilinear;
trilinearProps
{
    alphaMin number1;
}
```

- *number1* = minimum limit for voidfraction

Examples:

```
voidfractionModel trilinear;
trilinearProps
{
    alphaMin 0.3;
}
```

Description:

The trilinear voidFraction model is supposed to be used when a particle (or its representation) is in the size range of a CFD cell. The particle's volume is distributed over 8 neighbouring cell centres usind trilinear interpolation. This allows for a very smooth transition of particle volume when a particle travels from one cell to another cell.

Restrictions: The model works only for a structured mesh with equal cubic cells and a clean x/y/z parallel distribution of the cells. WARNING: the alphaMin parameter is not yet considered in the model!!!

Related commands:

voidfractionModel