Simulate with complex geometries and complex physics

Document created: November 03, 2020

OUTLINE

MESHFREE

Online Documentation for MESHFREE

General information

The original method is called Finite Pointset Method (FPM) and is an originary development of the Fraunhofer Institute for Industrial Mathematics ITWM . The software **MESHFREE** couples FPM and the algebraic multigrid method implemented in SAMG , an originary development of the Fraunhofer Institute for Algorithms and Scientific Computing SCAI .

FPM is the deprecated name of the numerical simulation idea, publications of which can be found for example in https://www.meshfree.eu/en/publications.html . Now and the in future, we prefer the name *Generalized Finite Difference Method (GFDM)* , as this states exactly the character of the method and avoids confusion with other ideas, also abbreviated as FPM.

Note that FPM is still the name of several commercial software-instances outside of ITWM, putting the original FPM-ideas into practice.

How to use MESHFREE

- InstallationGuide : install the software
- GettingStarted : first steps with MESHFREE
- Releases : stay up-to-date with new/current developments
- InputFiles : quick reference to all items and functionalities provided to the user
- Indices and __Constants__ : quick reference to all predefined variables and constants
- RunTimeTools : communication with a running simulation, performance measurements
- Solvers : underlying mathematical and numerical models

Highlights

Useful insight into PerformanceOptimization concerning geometry operations.

List of members:

__Constants__	typical %...%-constants that can be used in the input files
Download	Download executables, documentation and examples
GettingStarted	first steps with MESHFREE
Indices	MESHFREE indices for simulation entities
InputFiles	Input files used for steering MESHFREE
InstallationGuide	Installation of MESHFREE
PerformanceOptimization	useful insight into performance optimization
Releases	Information on the MESHFREE releases
RunTimeTools	tools regarding the run time
Solvers	Overview of numerical and geometrical algorithms used in MESHFREE
Support	How to contact the Support Team

1. InstallationGuide

Installation of MESHFREE

We recommend the usage of a Linux-based system (real or virtual machine). Supported operating systems are

- rhel7: Red Hat Enterprise Linux 7
- centos6: CentOS 6 (equivalent to Red Hat Enterprise Linux 6)

Download

Download an appropriate stable-version of MESHFREE from https://svn.itwm.fraunhofer.de/svn/MESHFREEdocu/Executables/MESHFREE/stable or download an appropriate beta-version of MESHFREE from https://svn.itwm.fraunhofer.de/svn/MESHFREEdocu/Executables/MESHFREE/beta .

Details on the folder structure and the naming scheme can be found here: NamingSchemeExecutables .

The newest developments can be obtained in the beta-versions, however they might not be completely stable towards all aspects of the software. beta-versions are tested only on a limited set of test problems. They are created once per two months. The stable versions are tested on an extended set of reference problems, however they are created only twice per year. For details on the release cycle, see Releases .

Download always the newest version (the older ones are there for reference only). If it is unclear which category of executables to download from, contact our Support team for assistance.

Installation

- Unpack the archive containing MESHFREE into your preferred installation folder. For this, open a shell and execute the following commands.

```
cd /path/to/download/ArchiveName.tar.gz
mkdir -p /path/to/meshfree/installation/folder
tar -x -f ArchiveName.tar.gz -C /path/to/meshfree/installation/folder
cd /path/to/meshfree/installation/folder
```

- Follow the installation steps described in the contained README.txt file.

Note: For installation and subsequent execution we assume a bash-shell or similar. If working on a c-shell, especially the export commands will have to be replaced by setenv and the appropriate syntax.

If you encounter any problems, please contact our Support team.

Execution

After successful installation, first time users are advised to continue with GettingStarted .

Experienced users can procede as follows: Execute .

Analysis

For postprocessing, the simulation results (MESHFREE point cloud as well as geometry elements) can be saved. To view and analyze the results, we recommend to download and install ParaView (see ParaViewTipsAndTricks). Details on the available file formats and their usage can be found here: SAVE .

Furthermore, integrated simulation results can be saved in tabular form, see INTEGRATION for details. This data can be analyzed, e.g. with the help of GNU Octave.

List of members:	
Execute	running MESHFREE
NamingSchemeExecutables	naming scheme of the MESHFREE executables
ParaViewTipsAndTricks	tips and tricks for postprocessing MESHFREE results with ParaView

1.1. Execute

running MESHFREE

We presume that MESHFREE has been installed as described in InstallationGuide . In order to run MESHFREE , open a shell, go into your project directory (including the InputFiles and geometry data) and execute the run script:

```
cd /path/to/my/project
/path/to/meshfree/installation/folder/meshfree_run.sh # serial
/path/to/meshfree/installation/folder/meshfree_run.sh N # on N MPI processes
/path/to/meshfree/installation/folder/meshfree_run.sh N M # on N MPI processes, each with M openMP threads
/path/to/meshfree/installation/folder/meshfree_run.sh N M [other parameters] # for other command line options, see documentation
```

The first (optional) parameter is taken as the number of MPI processes (default 1), provided the executable supports MPI. The second (optional) parameter is taken as the number of openMP threads (default 1), provided the executable supports openMP. For information on which MESHFREE versions support MPI or openMP, see NamingSchemeExecutables . Any further parameters are passed on to the MESHFREE call, see CommandLine .

If working on a Linux cluster (running on more than one compute nodes), make sure that there exists a valid nodefile (listing your compute resources). Please ensure furthermore, that the full name of the nodefile is held by the environment variable $PBS_NODEFILE .

We recommend setting an alias by adding the following line to your ~/.bashrc

```
alias meshfree='/full/path/to/meshfree/installation/folder/meshfree_run.sh'
```

Then the above commands are shortened to

```
cd /path/to/my/project
meshfree # serial
meshfree N # on N MPI processes
meshfree N M # on N MPI processes, each with M openMP threads
meshfree N M [other parameters] # for other command line options, see documentation
```

List of members:	
CommandLine	Command line options for MESHFREE
EnvironmentVariables	Environment variables for MESHFREE

1.1.1. CommandLine

Command line options for MESHFREE

MESHFREE supports several command line parameters and respects a few environment variables.

Option	
-nt {number of threads} --num-threads {number of threads}	Specifies the number of OpenMP threads. This does not have an effect for the pure MPI version of MESHFREE .
-e {/path/prefix/} --exec-dir {/path/prefix/}	This will run MESHREE inside /path/prefix/ as if it had been started directly there.
-r {/path/prefix/} --result-dir {/path/prefix/}	This will prepend /path/prefix/ to every SAVE_path . It makes most sense when using relative paths and terminating the prefix with a slash. Also see EnvironmentVariables .
-wf {file name} --warning-file {file name}	Specify a file name for the warnings file.
-clp {parameter string} --clparam {parameter string}	Specify a general purpose parameter string. Use this via @CLPARAM@ in USER_common_variables.dat
-k --kill	Let MESHFREE kill itself after termination. Under certain circumstances MESHFREE might hang upon exit when used with MPI. In these cases killing it will release the resources immediately.
-enc {filenames} --encrypt {filenames} --expiry-date {days}	Will encrypt all the given files into filename.enc and use it as described in Encryption . Can be added to -enc to specify the amount of time the days the encrypted file is valid
For example:	MESHFREE.x -enc USER_common_variables.dat --expiry-date 10 will encrypt the USER_common_variables.dat into USER_common_variables.dat.enc and will be valid for 10 days
--executeStepByStep -step	execute MESHFREE in step-by-step execution modus from the beginning of the program. See step-by-step-execution for details. This might help debugging cases with complex geometry items.
--iFPM_process_ID	Define the process identification number as an integer value. If this option is not given, MESHFREE will assign the ID as the computers clock time at program startup in seconds. The process ID is part of the names for SIGNAL- and log-files.
-lcs --check-license	Check for a valid license and exit.
--version	Print version number and exit.

Additionally, there are two positional command line options. The first unknown option will be interpreted as the name of the USER_common_variables file and the second as the file name of the common_variables file. The position within the above options can be arbitrary.

List of members:

Encryption	Encrypts files to share UCVs and CVs MESHFREE can work with but cannot be read by a human

Encryption

Encrypts files to share UCVs and CVs MESHFREE can work with but cannot be read by a human

To encrypt files please check the CommandLine section

If MESHFREE cannot find the given UCVs and/or CVs MESHFREE will automatically search for the give name with the appendix '.enc'

For example if no specific CV and UCV name was given the two files MESHFREE looks for are common_variables.dat and USER_common_variables.dat

If one of those is not found MESHFREE looks for common_variables.dat.enc and USER_common_variable.dat.enc
If those encrypted files are not found either MESHFREE will exit with an error message.

It is currently not possible to \include_Ucv{} an encrypted file into an encrypted file.
It is possible to include multiple encrypted files into an unecrypted file via \include_Ucv{}
Note: the name of the file need to be without the .enc, generally you should never specify
the .enc ending into any of your parameters as those will be automatically found once
there is no file found with the original name.

1.1.2. EnvironmentVariables

Environment variables for MESHFREE

- FPM_LICENSE_FILE is the most important environment variable as it sets
  the path to the license file. It must include the full path including
  the file name. It is not sufficient to just point it to the directory
  where the license file is located.
- OMP_NUM_THREADS is a default environment variable for OpenMP. It defines
  the number of OpenMP threads to be used if specified. However, the
  command line option -nt will override this environment variable if provided.
- FPM_RESULTDIR_PREFIX specifies a prefix to be prepended to every SAVE_path .
  This environment variable will be overriden by the -r command line option.

1.2. NamingSchemeExecutables

naming scheme of the MESHFREE executables

Structure

The folder structure on https://svn.itwm.fraunhofer.de/svn/MESHFREEdocu/Executables/MESHFREE/ is as follows:

- stable vs beta versions
- release vs debug versions

- operating systems
  - rhel7: Red Hat Enterprise Linux 7
  - centos6: CentOS 6 (equivalent to Red Hat Enterprise Linux 6)
- different versions

Naming

Naming scheme for executable/installation archive:

- meshfree
- optional marker d if debug version
- version, e.g. R2018.1.0 for stable version, beta2020.01.0 for beta version, see Releases
- included SAMG version, e.g. SAMG18.05.00
- optional marker o if SAMG includes openMP parallelisation
- type of executable
  - mpi: MPI parallelisation (incl. MPI shared memory)
  - mpin: MPI parallelisation, NO MPI shared memory
  - omp: openMP parallelisation
  - ompi: MPI and openMP parallelisation (without MPI shared memory)
    </p>
- precision, so far only d=double, but s=single, q=quad also possible
- operating system (see above)
- architecture: x86 = 32 bit, x64 = (64 bit arch, 32 bit integers), x64i = (64 bit arch, 64 bit integers)
- compiler and mpi versions
- optional marker pCS if parallel computational steering is provided, see ComputationalSteering

---

MESHFREE · InstallationGuide · ParaViewTipsAndTricks

1.3. ParaViewTipsAndTricks

tips and tricks for postprocessing MESHFREE results with ParaView

By default, MESHFREE writes two types of result files, one for the boundary elements and one for the point cloud. Both can be visualized by ParaView with already implemented features:

- Switching on the 'Animation View' produces a timeline. Jumping between time steps becomes much easier.
- Switching on the 'Statistics Inspector' provides further information on the loaded data sets, e.g. the number of points.
- For a boundary elements result file, the aliases are listed in the corresponding 'Multi-block Inspector' tab. By checking/unchecking the boxes, only the desired aliases can be visualized.
- For a point cloud results file, it is common to change the representation from 'Surface' (default) to 'Points'.
- The following 'Filters' are useful:
  - 'Clip' with clip types 'Plane' and 'Box' (restrict the result geometrically)
  - 'Threshold' (restrict the result wrt a scalar quantity)
  - 'Glyph' with glyph type 'Arrow' (visualization of vector fields) and 'Sphere' (visualization of simulation points as spheres, especially in case of DROPLETPHASE)
  - 'Calculator' (compute quantities as a function of the loaded simulation data)
- 'Save State' can be used to save the executed commands. Using 'Load State', a previously saved state can be restored.

MESHFREE · GettingStarted

2. GettingStarted

first steps with MESHFREE

Training Courses: Introduction to MESHFREE

- 22-24 September 2020 at Fraunhofer ITWM, Kaiserslautern, Germany. Details to follow.
- 16-18 March 2021 at Fraunhofer ITWM, Kaiserslautern, Germany. Details to follow.

If you are interested in attending, please contact our Support team.

Basics

In the Introduction , basic information on the underlying concepts and the general workflow of MESHFREE are presented. Beginners learn how to run their first simulation.

Tutorials

The Tutorial suite provides an insight into several important features of MESHFREE .

Specials

The LetterCases and SpecialCases from previous or current projects highlight advanced features.

See Download for archives of example setup suites.

List of members:

SpecialCases	Selected cases from current or previous projects or solving classical physics
Introduction	basic concepts and general workflow of MESHFREE
LetterCases	highlighting several capabilities of MESHFREE
Tutorial	simple, comprehensive examples in 3D

2.1. Introduction

basic concepts and general workflow of MESHFREE

Training presentation

In the training presentation you find a detailed introduction to MESHFREE . It explains:

- fundamental concepts of MESHFREE regarding point cloud management
- the general workflow
  - preparation of a surface mesh of the bounding/effective geometry
  - setup of the InputFiles
  - execution of the simulation
  - analysis of the results

Training setup

In the training folder you find the InputFiles and geometry for a first project in MESHFREE , a pipe flow:

- USER_common_variables.dat (main input file for the simulation model)
- common_variables.dat (additional input file for development or debugging)
- pipe.msh (surface mesh of the bounding geometry)

The setup specifies a transient simulation for a pipe flow with constant inflow velocity.

First run of MESHFREE

Download the training folder to your desired location and execute MESHFREE there. For this, open a shell and execute the following commands.

```
cd /path/to/download/TrainingFolder
/path/to/meshfree/installation/folder/meshfree_run.sh
```

This launches a serial execution of MESHFREE on your local machine. For MPI parallel execution, see Execute .

Note: We presume that MESHFREE has been installed as described in InstallationGuide .

While the simulation is running, you can already take a first glance at the transient results.

To view and analyze the results, we recommend to download and install ParaView. Open the MESHFREE result files 'TrainingSetup.case' (MESHFREE point cloud) and 'BE_TrainingSetup.case' (boundary elements, i.e. pipe) in the subfolder 'results' and take a look at the simulation output.

Figure 1 shows an example of a visualization with ParaView. This can be achieved by adapting the paths to the result files in the state file 'TrainingSetup_ParaViewState.py' and, subsequently, loading it in ParaView.

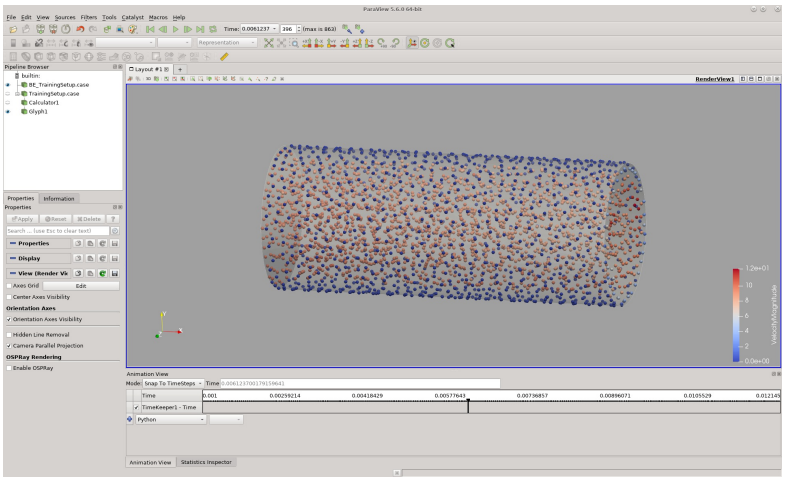


Figure 1: Visualization with ParaView.

Note: Upon loading the state file, the notation of the file names in the Pipeline Browser of ParaView will change to EnSightReader1 and EnSightReader2.

For further information, see the 'Analysis'-section of InstallationGuide .

If you encounter any problems, please **contact our Support team** .

For bold users

Can you build the file USER_common_variables.dat from scratch such that you get the simulation running? What are the necessary sections that you need in the file?

Feel free to make use of the training presentation and this documentation to solve this challenge!

Next steps

After the first successful run of MESHFREE , you should continue with the Tutorial .

DOWNLOAD COMPREHENSIVE EXAMPLE

2.2. LetterCases

highlighting several capabilities of MESHFREE

What we want to simulate
In the LetterCases tutorials we demonstrate several capacities of the MESHFREE software. We do this by taking the geometrical set-up, consisting of the letters "C", "F" and "D" (for Computational Fluid Dynamics) standing on a plate, and sprinkling these letters with droplets, letting them melt, rolling them flat with a cylinder and so on. All LetterCases tutorials are written with the implicit understanding that the user has already worked through the Tutorial cases. In this preliminary section we want to explain a few things found in most or all LetterCases UCV files that may have not been covered by the Tutorial or are worth a short explanation.

Geometry manipulations
When we include the letters we will often do additional modifications of their geometries. The include{ } command for the letter "C" could, for example, look like this:

```
begin_boundary_elements{ }
include{ C.stl}, applyAlias{ "C"}, scale{ &scaleC& }, offset{ [ &offset0C(1)& ],[ &offset0C(2)& ],[ &offset0C(3)& ]} ,
reorientation{ %GEO_Tube%, %GEO_Outside% }
end_boundary_elements
```

With the command offset{ } we can change the position of the letters. This allows us to place the letters nice and ordered in a row, whereas they would otherwise be overlapping each other. With reorientation{ } we can force the directions of the normal vectors of the geometries to the outside or the inside by choosing %GEO_Outside% or %GEO_Inside% respectively. This feature allows us to use the same geometry files for all LetterCases , whether we want the letters to be rigid and to interact with particles from outside itselves or we want the letters to contain particles and change their shape.

We will often need information about the space one of the letters, all letters together or the plate is occupying. We can get this information with a CONSTRUCT clause. Equipped with the argument %CONSTRUCT_BoxMidPoint% , it draws a box around the geometrical item whose alias it is given as the third argument. It returns a vector containing the position of a point somewhere on a line between the lower left und the upper right corner of the box, its exact position depending on the second argument.

```
begin_construct{ }
"minC" = CONSTRUCT ( %CONSTRUCT_BoxMidPoint% , 0, "C" )
"maxC" = CONSTRUCT ( %CONSTRUCT_BoxMidPoint% , 1, "C" )
end_construct
```

In this case, "minC" would contain the position of the lower left corner of the box, whereas "maxC" would contain the position of its upper right corner. With 0.5 as second argument, we would receive the position of the centre of the box.

Figure 0: The general geometrical set-up.

Stability constraints
Sometimes, for example when a point has few neighbors or when the order used in the approximation of its differential operators is low, we want to vary some boundary conditions to maintain a certain level of numerical stability. We define an equation called "IsCritical" to quickly test a particle for possible stability issues.

```
#--------------------------------------------------------------------------
# stability constraints
#--------------------------------------------------------------------------
begin_equation{ $IsCritical$ } # 1 means critical, -1 means all in butter
if ( 0 ) :: 1
elseif (Y%ind_OrdApprox(2)%<2) :: 1 # bad order of approximation (laplace)
elseif (Y%ind_OrdApprox(1)%<2) :: 1 # bad order of approximation (gradient)
elseif (Y %ind_nbRegularNeighbors% < 15) :: 1 # number of neighbors less than 15
elseif ( Y %ind_nbInteriorNeighbors% < 4 ) :: 1 # if a tear-off point (direct link between free surface and wall) has too few
neighbors
else :: -1 # point NOT critical (regular case)
endif
end_equation
ENFORCE_min_max ( $Mat1$ ,%ind_v(1)%) = (-3.0, 3.0 )
ENFORCE_min_max ( $Mat1$ ,%ind_v(2)%) = (-3.0, 3.0 )
ENFORCE_min_max ( $Mat1$ ,%ind_v(3)%) = (-3.0, 3.0 )
```

For the same reasons we set an upper and lower limit for the velocity. This is achieved by the handy command ENFORCE_min_max . Since these commands are evaluated at the end of each time step, the results of the time integrations are taken and values that are too small or too big are set to the maximum and minimum values, respectively.

See Download for archives of example setup suites.

List of members:	
CleaningJet	Letters getting washed away by a water jet
Coating	Letters getting coated with enamel
Melting	Letters melting in two different ways
Rolling	Letters getting flattened by a rolling cylinder
Spray	Letters getting sprayed with paint
Swelling	Letters swelling like bread
Swelling_b	Letters swelling like muffins

2.2.1. CleaningJet

Letters getting washed away by a water jet

Goals of this Unit:
- Combine several UCV files
- Allow single particles to exist
- Let several materials interact with the same boundary
- Delete particles with EVENT statements

The fluid-mechanical problem
The letters are hit by a water jet and washed away. It will be necessary to model the letters and the water as two different materials and to take into account that the water will meet the letters with such force that a lot of particles might get isolated from the bulk. We should also delete particles that distance themselves too far from the geometry.

Manage two materials
One could manage several materials in two different ways:
- Use one file for all materials
- Use several files, each containing the informations for one material

The first option means a bit less work but can get easily much more confusing than the second option, even with just two materials. The typical way to go about this would be to use option 2 and to include the UCV files of the materials 2-n in the file of material 1 with this simple command:

```
include_Ucv{ Ucv_Water.dat}
```

Allow isolated particles
To allow isolated particels, i.e. particels that do not have any neighbor in their immediate vicinity, one has to add the following lines to the common_variables file:

```
COMP_IsolatedParticles_MinNbOfNeigh = 0
COMP_IsolatedParticles_MinNbOfInteriorNeigh = 0
```

By default, these options are set to 1 and 6 respectively, meaning whenever a particle has less than 1 interior point or less than 6 interior or boundary points near it, it gets deleted.

Two materials using the same boundary
The particles of the letters and of the water will both interact with our geometry "plate". Since we can associate "plate" with only one material, we need to use a little trick: We create a duplicate of "plate" called "plateWater" and can use it for our second material. Because the orientation of a duplicated boundary element is reversed by default, we need to do a revOrient{ } to regain the original orientation.

```
begin_boundary_elements{ }
...
manipulate{ "plate"} duplicate{ "plateWater"}
manipulate{ "plateWater"} revOrient{ }
end_boundary_elements
```

Deleting points with events
EVENTs are defined with at least a condition and the event that will be triggered for a particle which meets that condition. In our case we use the event %EVENT_DeletePoint% , which deletes a particle meeting at least one out of five conditions.

```
EVENT = ( [ if (Y%ind_cham%>0.5) :: Y %ind_x(1)% -( &maxPlate(1)& +1) else :: -1 endif ] , %EVENT_DeletePoint% )
EVENT = ( [ if (Y%ind_cham%>0.5) :: ( &minPlate(1)& -1)-Y %ind_x(1)% else :: -1 endif ] , %EVENT_DeletePoint% )
EVENT = ( [ if (Y%ind_cham%>0.5) :: Y %ind_x(2)% -( &maxPlate(2)& +1) else :: -1 endif ] , %EVENT_DeletePoint% )
EVENT = ( [ if (Y%ind_cham%>0.5) :: ( &minPlate(2)& -1)-Y %ind_x(2)% else :: -1 endif ] , %EVENT_DeletePoint% )
EVENT = ( [ if (Y%ind_cham%>0.5) :: Y %ind_x(3)% -2 else :: -1 endif ] , %EVENT_DeletePoint% )
```

Keep in mind that EVENT can do more than just delete particles, it could also be used to manipulate certain indices of particles meeting its conditions.

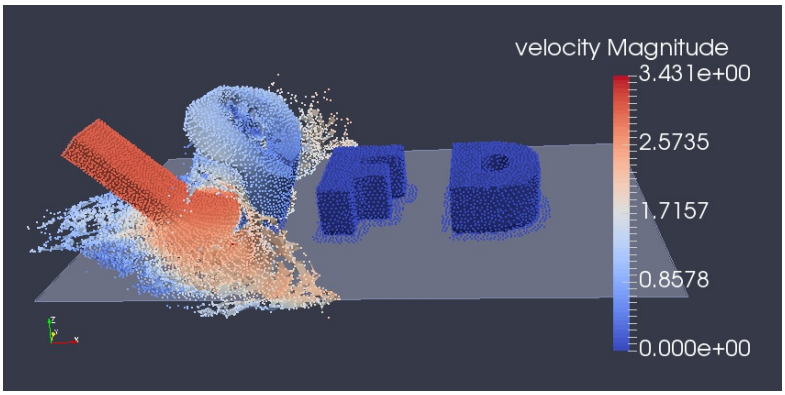


Figure 10: Mid-simulation results.

DOWNLOAD COMPREHENSIVE EXAMPLE

2.2.2. Coating

Letters getting coated with enamel

Goals of this Unit:
- Create plain boundary elements

The fluid-mechanical problem
The letters are getting coated by fluid emanating from a moving inflow boundary. We will only have a short look on how to create the geometry for the inflow boundary since the USER_common_variables file is easy to understand for everyone who completed the 3D tutorial.

Plain boundary elements
We create the inflow boundary as a rectangle by connecting two triangles. We create triangles with the command BND_tria simply by giving it the coordinates of three corner points.

```
begin_boundary_elements{ }
BND_tria &inflow& &inflow1(1)& &inflow1(2)& &inflow1(3)& &inflow2(1)& &inflow2(2)& &inflow2(3)& &inflow3(1)&
&inflow3(2)& &inflow3(3)&
BND_tria &inflow& &inflow3(1)& &inflow3(2)& &inflow3(3)& &inflow4(1)& &inflow4(2)& &inflow4(3)& &inflow1(1)&
&inflow1(2)& &inflow1(3)&
end_boundary_elements
```

We realize that we neither need a different alias for every created geometry item nor any extra commands to unite several geometry items under one alias.
Instead of creating two adjacent triangles with BND_tria , we could also use BND_quad to create the rectangle with only one command. This wouldn´t change a thing however, because FPM creates rectangles internally as a combination of two triangles anyway.

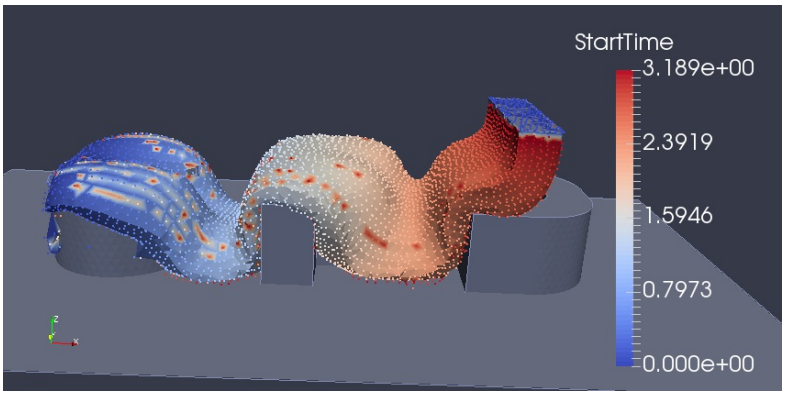


Figure 8: Mid-simulation results.

DOWNLOAD COMPREHENSIVE EXAMPLE

2.2.3. Melting

Letters melting in two different ways

Goals of this Unit:

- Get to know the temperature boundary condition %BND_AVERAGE%.
- Use CODI to buffer information from bygone time steps.

The fluid-mechanical problem
We are simulating two problems within this LetterCase:
(i) The letters are standing on a warm plate of constant temperature. By getting heated they melt and dissolve from bottom to top, just like butter in a frying pan.
(ii) The letters are standing in an oven. They are getting heated by recirculating air and start to dissolve from top to bottom.

We will first have a look at at problem (i).

Setting appropriate boundary conditions
A constant temperature for the plate (BC_pool) is easily set with a Dirichlet condition. For the free surface particles (BC0) of the letters to be gradually heated by the plate, we apply %BND_AVERAGE% as boundary condition. Thus the current temperature of a free surface point is calculated as a weighted average of the temperature of its neighboring points.

```
BC_T (0) = (%BND_AVERAGE%, 0, 0)
BC_T ( $BC_pool$ ) = ( %BND_DIRICH% , 1 )
```

Manipulate the viscosity with a UserDefinedIndex

We do not want the melting to be too aprupt. Therefore we need to constrain the decrease of viscosity in every timestep via its value in the previous timestep. This requires us to buffer the viscosity. The typical way to go about this would be by using a CODI variable to store this information, because all CODI commands are evaluated at the end of each time step after the time integration, while physical properties like the viscosity are updated before the time integration. This enables us to carry the information on physical properties from a previous timestep into the time integration of the following time step. We introduce a so-called UserDefinedIndex %indU_ETA_lastTS% and set it to be equal to Y %ind_ETA% .

```
"eta_min" = "10"
...
INITDATA ( $Mat1$ ,%indU_ETA_lastTS%) = &eta_min&
CODI_eq ( $Mat1$ ,%indU_ETA_lastTS%) = [Y %ind_ETA% ]
```

With this information up our sleeve we define the viscosity like this:

```
eta( $Mat1$ ) = [max( &eta_min& *exp(-12*Y %ind_T% ) , 0.00001* &eta_min& , 0.5*Y%indU_ETA_lastTS% )]
```

The first argument of max is a model of the viscosity decrease caused by rising temperature, the following arguments are constraints, averting the viscosity to plummet below a minimal value and by more than a half respectively.

The picture below shows how the letters are slowly melting from bottom to top, changing their shape and sliding across the plate.

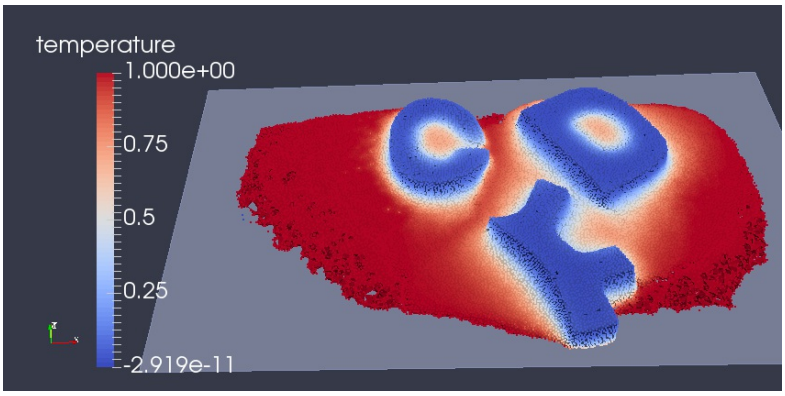


Figure 4: Melting from the bottom

Swap the boundary conditions

It is very easy to change the UCV to represent case (ii) instead of (i). One only needs to change the thermal conductivity lambda, the minimum viscosity eta_min and Tend to more convenient values and to swap the temperature boundary conditions for the free surface and for the plate.

```
"eta_min" = "1000"
...
Tend = 250
...
lambda( $Mat1$ ) = 1
...
BC_T (0) = ( %BND_DIRICH% , 1 )
BC_T ( $BC_pool$ ) = (%BND_AVERAGE%, 0, 0)
```

The result should look like this:

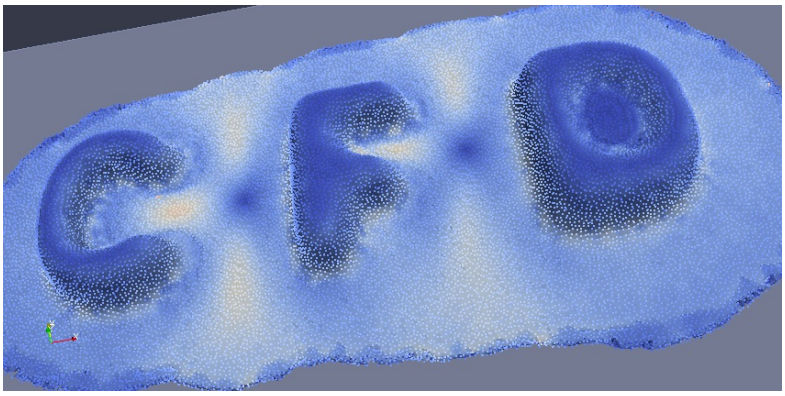


Figure 5: Melting from the top

Looking at a cross-section of the simulation shows the difference in temperature distribution compared to (i):

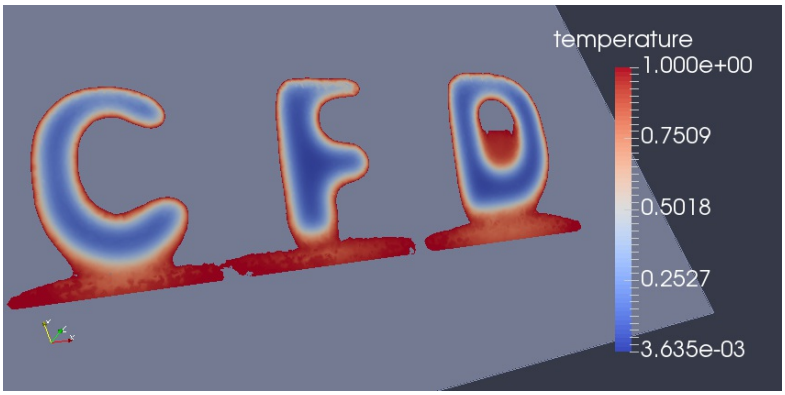


Figure 6: A cross-section of the letters

DOWNLOAD COMPREHENSIVE EXAMPLE

2.2.4. Rolling

Letters getting flattened by a rolling cylinder

Goals of this Unit:

- Construct a cylinder.
- Define its movement (translations and rotations) via curves.
- A short excursion about BC_TearOffCriterion .

The fluid-mechanical problem
The letters are flattened by a huge cylinder rolling back and forth, just like dough getting flattened by a rolling pin. We want to simulate a situation where the pin is coated in flour, thereby preventing the dough from sticking to it. We also have to take into account that the cylinder does not only do a translation but also a simultaneous rotation.

Construction of the cylinder
First we create an alias for the cylinder. Its boundary conditions will later be referenced by BC_roll und its movement by $MOVE_roll$. We arbitrarily set its radius to 0.5.

```
begin_alias{ }
...
"roll" = " BC$BC_roll$ ACTIVE$noinit_always$ IDENT%BND_slip% MAT$Mat1$ TOUCH%TOUCH_liquid%
MOVE$MOVE_roll$ LAYER0 CHAMBER1 "
...
end_alias
...
begin_alias{ }
"rRoll" = "0.5"
end_alias
```

We need more information on the measurements of our boundary elements to construct the cylinder with the correct length and to define its movement from one edge of the plate to the opposite edge. Therefore we get the points at the lower left and upper right corners of enclosing boxes around the 3 letters and the plate respectively. With these we can define three important values.

```
begin_construct{ }
"minALL" = CONSTRUCT ( %CONSTRUCT_BoxMidPoint% , 0.0, "C","F","D" )
"maxALL" = CONSTRUCT ( %CONSTRUCT_BoxMidPoint% , 1.0, "C","F","D" )
"minPlate" = CONSTRUCT ( %CONSTRUCT_BoxMidPoint% , 0.0, "plate" )
"maxPlate" = CONSTRUCT ( %CONSTRUCT_BoxMidPoint% , 1.0, "plate" )
"rollCenter" = " [&minALL(1)&-&rRoll&] , [&minPlate(2)&] , [&maxALL(3)&+&rRoll&] "
"rollTravelLength" = " [&maxALL(1)&-&minALL(1)&+2*&rRoll&]"
"rollOmega" = " [ (2*3.1415926/2)*( &rollTravelLength&/(2*3.1415926*&rRoll&) ) ] "
end_construct
```

The first is "rollCenter", which will be used as the central point of the bottom endpiece (with respect to the y-axis) of the cylinder. The second is "rollTravelLength", which is the traveling distance of the cylinder. The third is "rollOmega", the angular velocity of the cylinder. Its definition means that the cylinder will do 0.5*&rollTravelLength&/&rRoll& full rotations every second.
With these values we can now easily define the cylinder via the command BND_cylinder .

```
begin_boundary_elements{ }
BND_cylinder &roll& &rollCenter(1)& &rollCenter(2)& &rollCenter(3)& 0 1 0 [ &maxPlate(2)& - &minPlate(2)& ] &rRoll&
&rRoll& 40
manipulate{ "roll"} revOrient{ }
end_boundary_elements
```

It needs an alias ("roll"), the position of the central point of the bottom endpiece of the cylinder (&rollCenter(1)& &rollCenter(2)& &rollCenter(3)&), a direction (0 1 0), the length of the cylinder ([&maxPlate(2)&-&minPlate(2)&]) and the radii for the bottom and the top endpiece (both &rRoll&). The number 40 is given as an optional argument and determines the fineness of the resolution of the round cylinder.

Defining the movement of the cylinder
We define the translation and the rotation of the cylinder by two different curves. The first curve $CRV_centerOfRoll$ describes the translation of the point "rollCenter" dependent on time.

```
begin_curve{ $CRV_centerOfRoll$ }, nb_functions {4}
0.0 %MOVE_position% 0 0 0
0.1 %MOVE_position% 0 0 -0.25
2.1 %MOVE_position% &rollTravelLength& 0 -0.25
2.2 %MOVE_position% &rollTravelLength& 0 -0.375
4.2 %MOVE_position% 0 0 -0.375
4.3 %MOVE_position% 0 0 -0.4375
6.3 %MOVE_position% &rollTravelLength& 0 -0.4375
6.4 %MOVE_position% &rollTravelLength& 0 -0.46875
8.4 %MOVE_position% 0 0 -0.46875
end_curve
```

The point is moved from left to right and vice versa. It crosses the distance after two seconds. At the start and every time it reaches an edge, it is lowered a bit.
The rotation is described by the curve $CRV_omegaOfRoll$. The direction of the rotation is changed every time "rollCenter" reaches one of the two edges.

```
begin_curve{ $CRV_omegaOfRoll$ }
0.0 0
0.1 &rollOmega&
2.1 &rollOmega&
2.2 - &rollOmega&
4.2 - &rollOmega&
4.3 &rollOmega&
6.3 &rollOmega&
6.4 - &rollOmega&
8.4 - &rollOmega&
end_curve
```

The translation statement only concerns "rollCenter". We need to link this movement and the rotation with "roll", the actual rigid body, via a fitting MOVE statement. This can be done with the command %MOVE_TranslationRotation% . As the name suggests, it lets us combine a translational with a rotational movement for a boundary element. It needs a point on the initial centre of rotation (&rollCenter(1)&, &rollCenter(2)&, &rollCenter(3)&), a MOVE statement describing the movement of this centre ($MOVE_centerOfRoll$) and a vector for the angular velocity (0, curve{$CRV_omegaOfRoll$}{0}, 0).

```
MOVE ( $MOVE_centerOfRoll$ ) = curve{ $CRV_centerOfRoll$ }{0}
MOVE ( $MOVE_roll$ ) = ( %MOVE_TranslationRotation% , &rollCenter(1)& , &rollCenter(2)& , &rollCenter(3)& ,
$MOVE_centerOfRoll$ , 0, curve{ $CRV_omegaOfRoll$ }{0}, 0 )
```

About tear-off criteria

MESHFREE offers its users the opportunity to create their very own tear-off criteria. Tear-off criteria determine when a boundary point becomes a free surface point. This is, for example, important when one is considering gravity effects. Boundary points that experience a strong acceleration away from their boundary elements should not be glued to these possibly unmoving boundaries but rather become free surface particles instead.

```
BC_TearOffCriterion ( $BC_roll$ ) = ( [(Y %ind_v(3)% )] , [(Y %ind_act% -3)] )
```

We can define our own tear-off criteria with BC_TearOffCriterion (BC_roll). A boundary point of BC_roll becomes a free surface particle when all statements on the right hand side of the expression above are true. In our case it would mean that a boundary particle of the cylinder becomes free, when it is both moving upward and when it was active for more than three time steps. This ensures that our material is not sticking to the cylinder after being flattened.

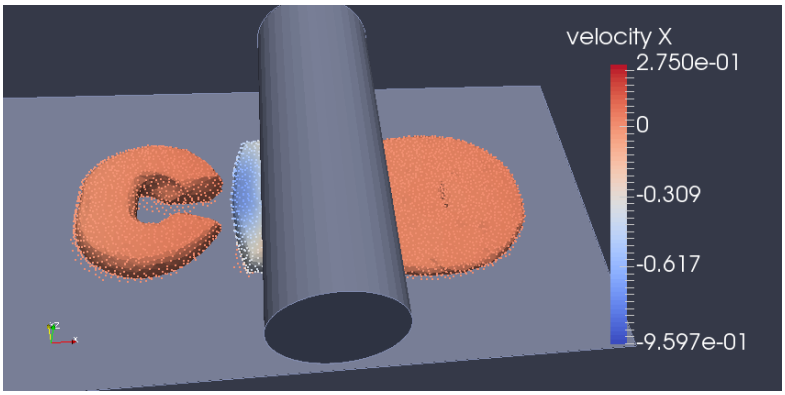


Figure 9: Rolling : Mid-simulation results.

DOWNLOAD COMPREHENSIVE EXAMPLE

2.2.5. Spray

Letters getting sprayed with paint

Goals of this Unit:

- Learn how to use DropletSource
- Learn about the rand function

The fluid-mechanical problem

The letters are getting sprinkled by droplets. The droplets are randomly distributed along a line that is moving along the plate.

Creating a DropletSource

A DropletSource produces droplets non-stop with a time lag between the individual droplets that is determined by its first two arguments: The very first argument defines the volume flux in m^3/s to be created by the source and the second argument the volume of each droplet. The next three Arguments determine the (potentially time-dependent) spatial position of the source, while the last two arguments determine the chamber and material index of the droplets respectively.

```
DropletSource = ( 0.020, [(1.5* &H_min& )**3], curve{ $CRV_centerOfinflow$ }{0}, [ &minALL(2)& + rand(1)*( &maxALL(2)& - &minALL(2)& )], 1, 1, $Mat1$ )
```

In our case we set the source to be at a fixed height above the letters. In x-direction it moves slowly along the plate from left to right and vice versa as defined in $CRV_centerOfinflow$. The y-coordinate is changed randomly every time a new droplet is produced, but within the boundaries of the plate.

How rand() works

The function rand(a) produces a random number when it is called. It produces a number between 0 and a if a is a positive real number and a number between -a and a if a is negative.

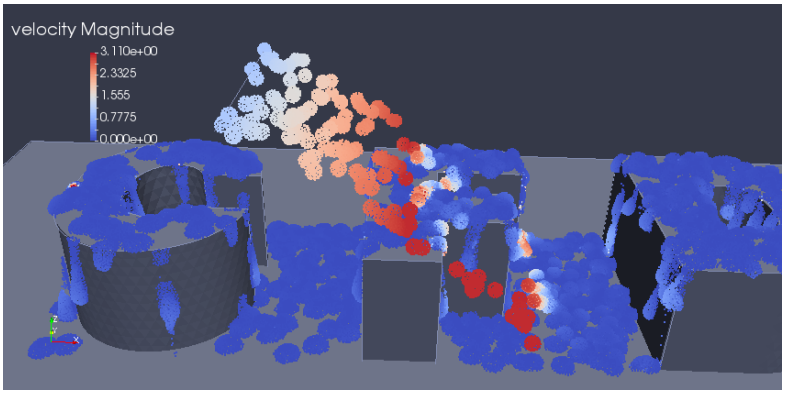


Figure 7: Mid-simulation results.

Suggestions to explore MESHFREE

- Play around with the first two arguments of DropletSource and see how they can speed up or slow down the droplet generation
- You could also try to replace rand() with some self-written equation to make the droplets fall in a certain order

DOWNLOAD COMPREHENSIVE EXAMPLE

2.2.6. Swelling

Letters swelling like bread

Goals of this Unit:

- Heat the letters gradually from the outside.
- Make density and viscosity dependent on temperature.

The fluid-mechanical problem
We want the letters to behave like bread dough getting heated in an oven. To do this we need to apply heat gradually to the outside of the letters and we need the letters to swell and to change their texture during the heating process.

Apply heat to the letters
First of all, we want the letters to have a free surface so they can change their shape. This is achieved by setting their ACTIVE flags to ACTIVE $init_never$, which lets the boundaries of the letters participate in the initial filling of the point cloud but ignores them afterwards.

```
"C" = " BC0 ACTIVE$init_never$ IDENT%BND_slip% MAT$Mat1$ TOUCH%TOUCH_liquid% MOVE-1 LAYER0
CHAMBER1 SYMMETRYFACE2 "
"F" = " BC0 ACTIVE$init_never$ IDENT%BND_slip% MAT$Mat1$ TOUCH%TOUCH_liquid% MOVE-1 LAYER0
CHAMBER1 SYMMETRYFACE3 "
"D" = " BC0 ACTIVE$init_never$ IDENT%BND_slip% MAT$Mat1$ TOUCH%TOUCH_liquid% MOVE-1 LAYER0
CHAMBER1 SYMMETRYFACE4 "
```

This creates free surface particles at the boundaries "C", "F" and "D", which can now be referenced by the boundary condition "0". We force their temperature to grow linear with time. Its value starts by 0 at Y %ind_time% = 0 and scales up to a maximum of 1 at Y %ind_time% = 2.

```
BC_T (0) = ( %BND_DIRICH% , [min( 0 + 0.5*Y %ind_time% , 1 )] )
```

Manipulate density and viscosity
By letting the density of the particles increase with temperature, we can induce an expansion of the letters. We also want the viscosity to increase with temperature, thus simulating the hardening of the dough during the baking process. Finally, we restrain both parameters, thus modeling the end condition when the dough has fully transformed into bread. All of this can be achieved very simply via the max-function.

```
density( $Mat1$ ) = [max(1-0.7*Y %ind_T% , 1-0.7)]
...
eta( $Mat1$ ) = [ &eta_min& + (max(Y %ind_T% ,0.001)^1.5)*30]
```

Here is an intermediate result of the simulation, where one can see the temperature distribution throughout the letters:

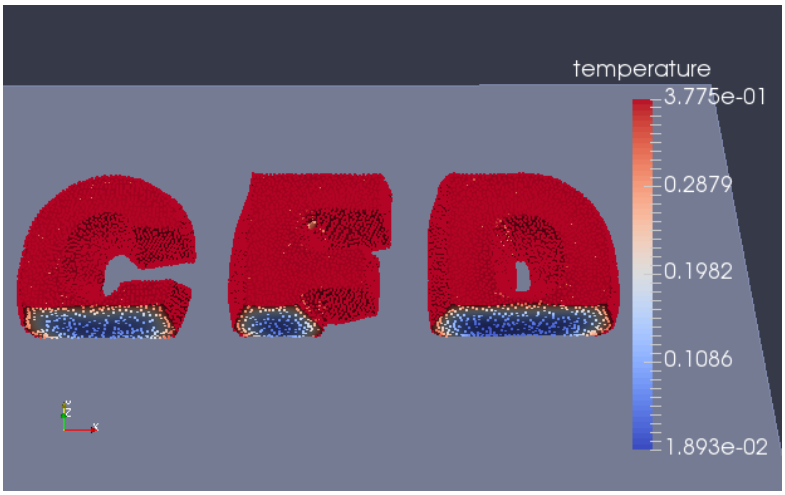


Figure 1: A cross-section of the letters taken mid-simulation

Suggestions for exploring MESHFREE

- Exchange the temperature boundary conditions for the letters and the plate
- Tinker with the provided expressions for density and viscosity. You could for example impose smaller or higher boundaries on the density

DOWNLOAD COMPREHENSIVE EXAMPLE

2.2.7. Swelling_b

Letters swelling like muffins

Goals of this Unit:

- Learn about advanced geometry manipulations.
- Establish a metaplane.

The fluid-mechanical problem

We want the letters to behave like muffin dough getting heated in a muffin pan. Since we can simulate the physical properties of the dough very similar to the first Swelling case, this tutorial will instead focus on manipulating the geometry of our letters to make their shape more alike to a muffin pan.

Transform the letters into a conus-like shape

We start off by shifting our letters to the center of the x-y-plane. This enables us to deform the letters in x- and y-direction in a way that is symmetric to the z-axis. With scale{ [1+Y %ind_x(3)% *0.7] we achieve a conus-like shape. After that we shift the letters back to their initial position.

```
begin_construct{ }
"midC" = CONSTRUCT ( %CONSTRUCT_BoxMidPoint% , 0.5, "C" )
"midF" = CONSTRUCT ( %CONSTRUCT_BoxMidPoint% , 0.5, "F" )
"midD" = CONSTRUCT ( %CONSTRUCT_BoxMidPoint% , 0.5, "D" )
end_construct
begin_boundary_elements{ }
manipulate{ "C"} offset{ - &midC(1)& , - &midC(2)& , 0 }, scale{ [1+Y %ind_x(3)% *0.7] , [1+Y %ind_x(3)% *0.7] , 1 },
offset{ &midC(1)& , &midC(2)& , 0 }
manipulate{ "F"} offset{ - &midF(1)& , - &midF(2)& , 0 }, scale{ [1+Y %ind_x(3)% *0.7] , [1+Y %ind_x(3)% *0.7] , 1 },
offset{ &midF(1)& , &midF(2)& , 0 }
manipulate{ "D"} offset{ - &midD(1)& , - &midD(2)& , 0 }, scale{ [1+Y %ind_x(3)% *0.7] , [1+Y %ind_x(3)% *0.7] , 1 },
offset{ &midD(1)& , &midD(2)& , 0 }
end_boundary_elements
```

Since we want the geometry to be similar to a muffin pan, we also need it to be open at the top. We construct a box around all three letters and get the position "maxALL" of a point which is just a bit bellow the top of the box. This point is also just below the top of every individual letter, because all letters have the same height. With the command removeBEonCondition we can now delete every particle whose position is above "maxALL", thereby removing the top of every letter.

```
begin_construct{ }
"maxALL" = CONSTRUCT ( %CONSTRUCT_BoxMidPoint% , 0.999, "C", "F", "D" )
end_construct
begin_boundary_elements{ }
manipulate{ "C","F","D"} removeBEonCondition{ %GEO_removeBasedOnNodes%, [Y %ind_x(3)% > &maxALL(3)& ] }
end_boundary_elements
```

Restrict the initial point cloud with a metaplane

When baking muffins one does not fill the muffing pan full to the brim but rather to the half. This means in terms of our simulation that we want to restrict the initial point cloud to remain below a certain plane parallel to the x-y-plane. We can easily achieve this by using a metaplane. A metaplane cuts off all points outside of it. By choosing $init_never$ for its active flag, we can restrict the point cloud during the initial filling and make the metaplane inactive for the rest of the simulation. A metaplane needs to be defined with a number to distinguish it from other possibly existing metaplanes:

```
"plane" = " METAPLANE1 BC$BC_free$ ACTIVE$init_never$ "
```

The plane itself can easily be defined by two vectors: The postion of an arbitrary point of the plane and the direction of the normal vector of the plane.

```
begin_boundary_elements{ }
BND_plane &plane& 0 0 0.3 0 0 -1
end_boundary_elements
```

In this case the position is (0, 0, 0.3) and (0, 0, -1) is the direction.

The restricted initial point cloud in the modified letter forms should look like this:

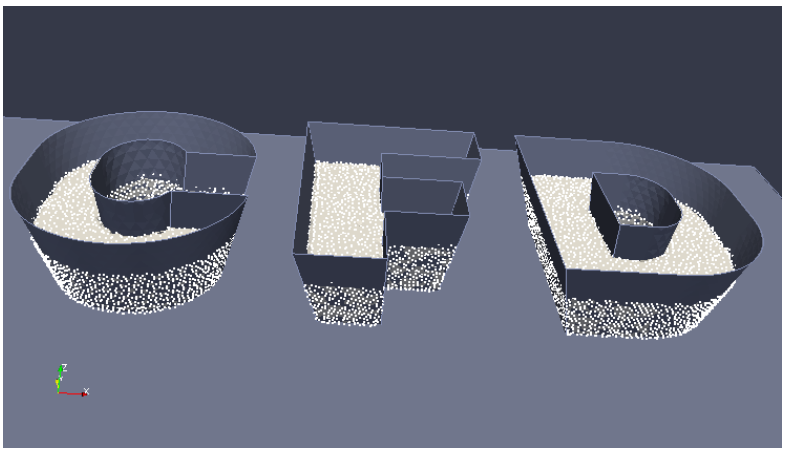


Figure 2: Point distribution at the beginning of the simulation

If we look at the temperature distribution of the points during the simulation, we can clearly see the different boundary conditions for the boundary points on the letter forms and the free surface boundary points:

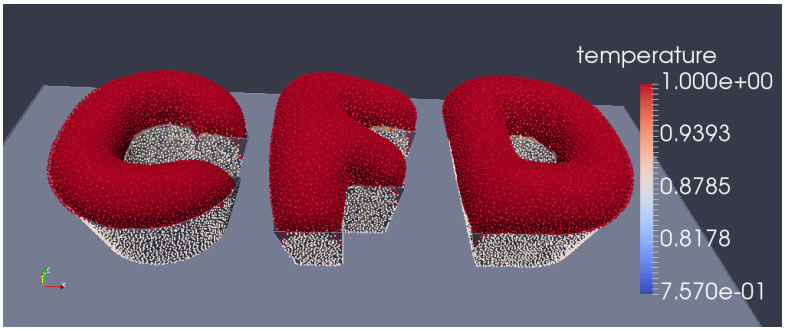


Figure 3: Mid-simulation results

DOWNLOAD COMPREHENSIVE EXAMPLE

2.3. SpecialCases

Selected cases from current or previous projects or solving classical physics

See Download for archives of example setup suites.

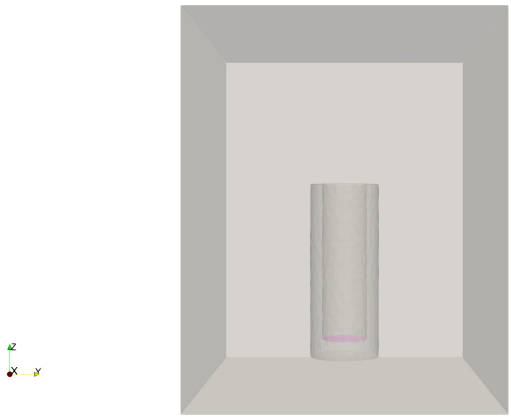
List of members:

BasicPhysics	Solve selected cases from classical physics and fluid mechanics
AirIntake	Air intake example to create a stable air flow field
MultiPhaseCoupling	Solve selected test cases in the field of multi-phase simulations
SimulationSplittingWithMEMORIZE	usage of the MEMORIZE-feature to split a simulation
WaterCrossing	Solve selected test cases in the field of water crossing simulations
WaterManagement	Solve selected test cases in the field of water management simulations

2.3.1. AirIntake

Air intake example to create a stable air flow field

This example shows how to set up an air intake simulation to get a stable and stationary air flow field. Furthermore it focuses on the EULERIMPL solver to save computation time for such test cases. The setup consists of a simple double walled tube within an air box:



At the bottom of the tube the air is sucked in with a user given velocity. To check the results, the dynamic pressure is compared to the Bernoulli pressure based on the maximum velocity:

```
INTEGRATION ( $PDYN_MIN$ ) = ( %MINIMUM_INT% , [Y %ind_p_corr% ], $air$ , %INTEGRATION_Header%,
"p_dyn min")
INTEGRATION ( $P_Bernoulli$ ) = ( %PUBLICVALUE% , [-0.5*1.0*(integ( $VEL_MAX$ ))^2],
%INTEGRATION_Header%, "p_Bernoulli")
INTEGRATION ( $DIFF_P_DYN_P_Bernoulli$ ) = ( %PUBLICVALUE% , [abs(integ( $PDYN_MIN$ ) - integ(
$P_Bernoulli$ ))], %INTEGRATION_Header%, "difference p_dyn - p_Bernoulli")
```

Recommended Settings

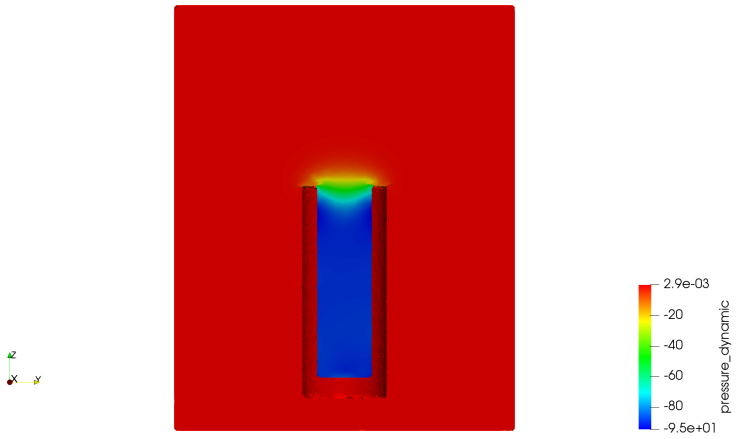
The best results can be achieved with the following settings:

- Use constant density (purely incompressible).
- damping_p_corr(1) = 0.0, so that the dynamic pressure is not considered for the initial guess in the next time level.
- No use of boundary conditions for the dynamic pressure, e.g. BCON (xxx,%ind_p_corr%) resp. BCON (xxx,%ind_p_dyn%).
- Static/Bernoulli pressure condition at box surface dependent on flow direction (see input file):

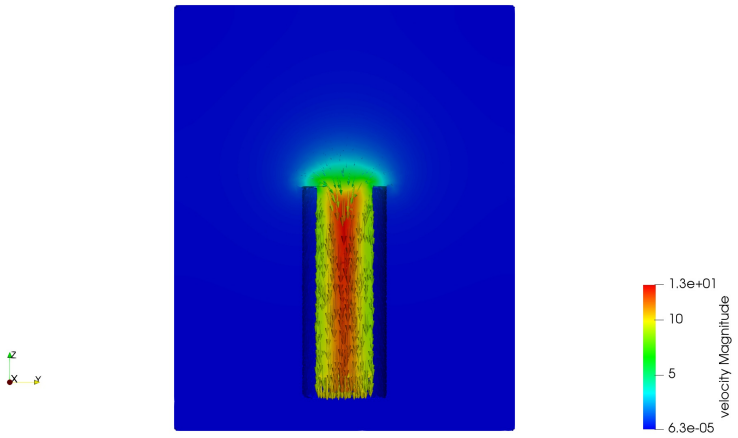
```
BC_p ( $air_out$ ) = ( %BND_DIRICH% , equn{ $StaticPressureAtOutflow$ })
```

Results of Stationary Air Flow Field

Dynamic Pressure at t = 5:



Velocity at t = 5:



DOWNLOAD COMPREHENSIVE EXAMPLE

2.3.2. BasicPhysics

Solve selected cases from classical physics and fluid mechanics

Examples comparing the numerical MESHFREE results with analytical approaches or with measurement results.

See Download for archives of example setup suites.

List of members:

Sand	applications for sand as continuous phase
Bernoulli	Compare numerical results to the Bernoulli equation
CollidingDropletsInCone	Colliding droplets in cone geometry
TwoPhaseDarcy	water jet deflected by air

MESHFREE · GettingStarted · SpecialCases · BasicPhysics · Bernoulli

Bernoulli

Compare numerical results to the Bernoulli equation

In many quasi-stationary applications with negligible viscous forces, one can use the Bernoulli equation to give an analytical estimate of the flow results (or parts of it). Bernoulli states

$$p_0 + \frac{1}{2}\rho \boldsymbol{v}^T \boldsymbol{v} + \rho \boldsymbol{g}^T \boldsymbol{x} = \text{const}$$

It is valid throughout the flow domain (in this case we have potential flow, i.e. flow with no rotation) - at least, it is valid for each pathline of the flow.

List of members:

FlowOutOfSimpleTank	flow of a liquid out of a tank

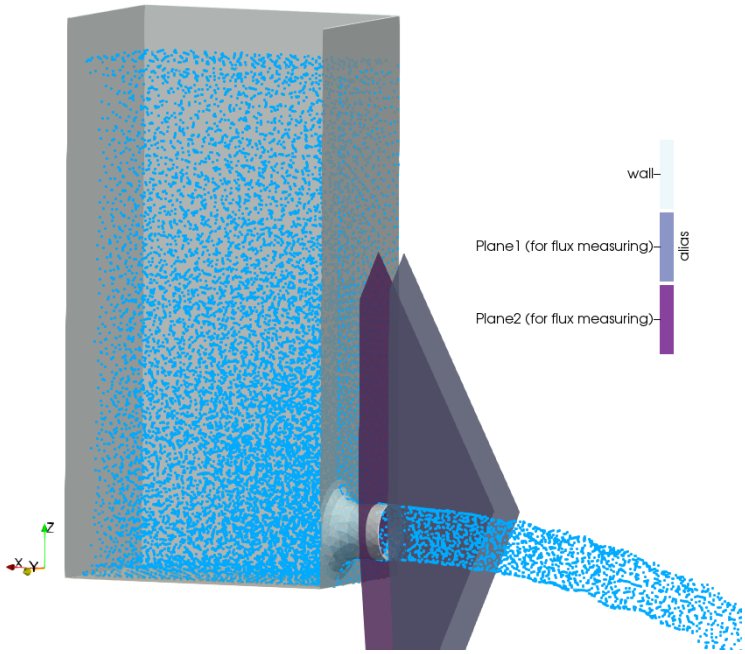
MESHFREE · GettingStarted · SpecialCases · BasicPhysics · Bernoulli · FlowOutOfSimpleTank

FlowOutOfSimpleTank

flow of a liquid out of a tank

The flux of a liquid out of a tank is given by Torricellis law .

In this example, we measure the numerical flux of a liquid out of a tank through two measurement planes and compare it to the theoretical value of Bernoulli /Torricelli. As their theory bases on non-viscous flow, we switch off the turbulence model and impose pure slip boundary conditions at the walls.



Note: For the flux measurement, we employ the flux integration (see %INTEGRATION_FLUX% and %INTEGRATION_FLUX_TIME%).

DOWNLOAD COMPREHENSIVE EXAMPLE

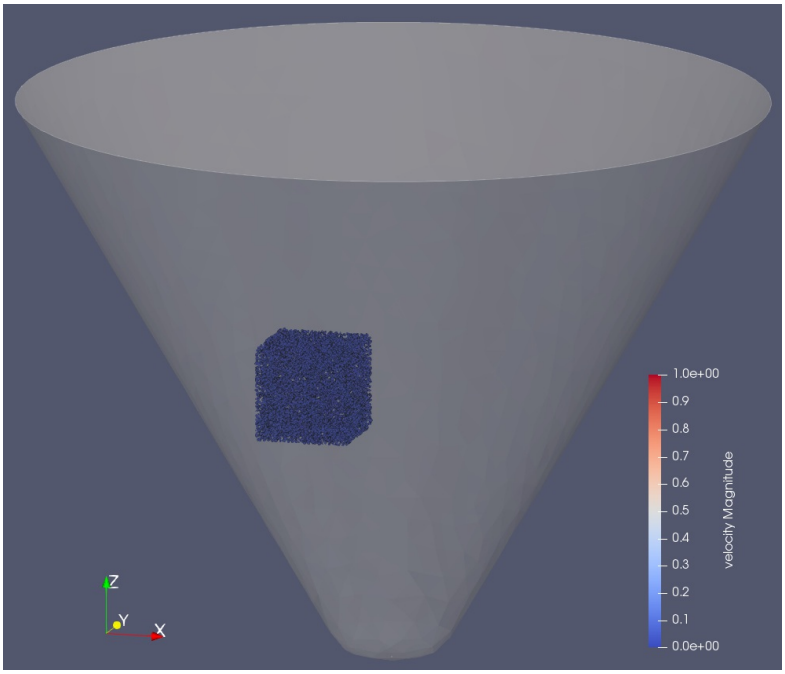
CollidingDropletsInCone

Colliding droplets in cone geometry

This example showcases some of the capabilities of the DROPLETPHASE solver in resolving collisions. The corresponding models and UCV syntax can also be found on the DropletCollisions page.

Starting point

As a starting point, we consider a block of DROPLETPHASE points which are filled within a cone geometry and with randomly varied droplet diameters:



The random variation of droplet size was defined via

```
INITDATA ( $DUST$ ,%ind_d30%) = [ nrand( &InitD30& , &D30sigma& ) ]
```

where %ind_d30% is the index storing the diameter of DROPLETPHASE particles. After initialization we will let these particles fall down under the effect of gravity which is pointing towards the tip of the cone, i.e. in negative z-direction.

Defining the collision model

To model the expected rebound of particles from the walls of the cone, we have to enable the collision model for boundaries by specifying

```
BC_v ( $wallt$ ) = ( %BND_COLLISION% , &kn_pw& , &en_pw& , &Ea_pw& , &Ra_pw& , &mu_pw& , &SF_pw& ,
&theta_pw& )
```

for the velocity boundary condition of all parts of the cone. Details on the arguments may be found under %BND_COLLISION% . It is important to note that the spring stiffness needs to be adapted to the configuration at hand. Specifically this means that it has to be chosen large enough so that overlaps don´t become too large for the given particle masses and collision velocity. In addition to boundary collisions, we also want to consider interactions between the particles so that they can stack on top of each other when gathering in the tip of the cone. This may be enabled via

```
ParticleInteraction( $DUST$ ) = ( &kn_pp& , &en_pp& , &Ea_pp& , &Ra_pp& , &mu_pp& )
```

Again, for an explanation of the parameters we refer to the ParticleInteraction. The models behind both %BND_COLLISION% and ParticleInteraction are described in detail in DropletCollisions .

Time step restrictions and subcycling

To accurately reproduce collision dynamics it is important to make sensible choices for the time step within the DROPLETPHASE . In this example, two special time step restrictions are used:

Restriction via COEFF_dt_d30 :
This is similar to COEFF_dt in that it ensures that points only travel a certain distance within each time step. The DROPLETPHASE -specific COEFF_dt_d30 only distinguishes itself from COEFF_dt by taking the particle radius instead of the smoothing length as reference distance. By supplying a value smaller than 1.0 it is guaranteed that all collisions captured by at least a single time step.

Restriction via DELT_dt_AddCond :
While no collision would go unnoticed for COEFF_dt_d30 smaller than 1.0, there is no guarantee that with this restriction alone the theoretical behavior of the collision model is reproduced accurately for all values of the spring stiffness. To alleviate this problem and make sure that every collision is reproduced to a satisfactory degree, we prescribe the additional condition

```
DELT_dt_AddCond ( $DUST$ ) = [ &frac_dtcoll& *min(equn{ $DUST_dt_coll_pw$ },equn{ $DUST_dt_coll_pp$ }) ]
```

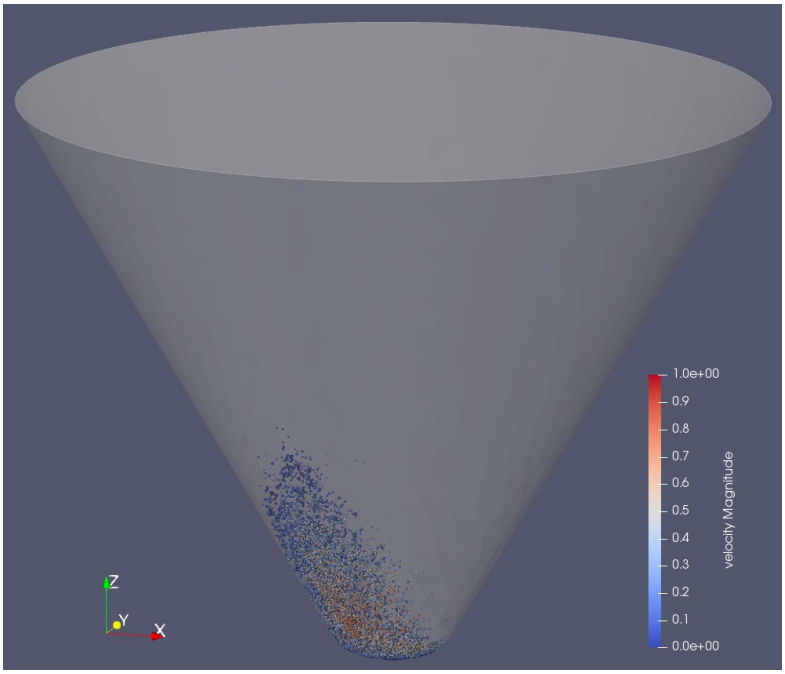
which ensures that multiple timesteps are within the theoretical contact duration of a collision.

Subcycling:
In order to avoid that all point organization routines are called in every one of the small time steps imposed by the above conditions, we further enable the subcycling within the DROPLETPHASE by setting

```
COMP_DropletphaseSubcycles = 200
```

Simulation result

Running the simulation with the predefined settings will show that particles are colliding with the side wall of the cone and slide downwards toward the tip, as is depicted in the following image:



The user is encouraged to try out modifications of the predefined switches and in particular the time step parameters above. An important step towards setting up own simulations using the DROPLETPHASE collision capabilities is building an understanding of why different time step restrictions are necessary and which behavior has to be expected whenever they are violated.

DOWNLOAD COMPREHENSIVE EXAMPLE

MESHFREE · GettingStarted · SpecialCases · BasicPhysics · Sand

Sand

applications for sand as continuous phase

In the examples given below, we use a continuous approach to model the behavior of sand, namely the DruckerPragerModel .

List of members:

SandPileDeposition	sand pile deposition
SandGuidedSphereImpact	guided sphere impact into sand

MESHFREE · GettingStarted · SpecialCases · BasicPhysics · Sand · SandGuidedSphereImpact

SandGuidedSphereImpact

guided sphere impact into sand

A sphere impacts a box filled with sand, see Figure 1. The movement of the sphere is guided, i.e. it moves with given constant velocity in z-direction.

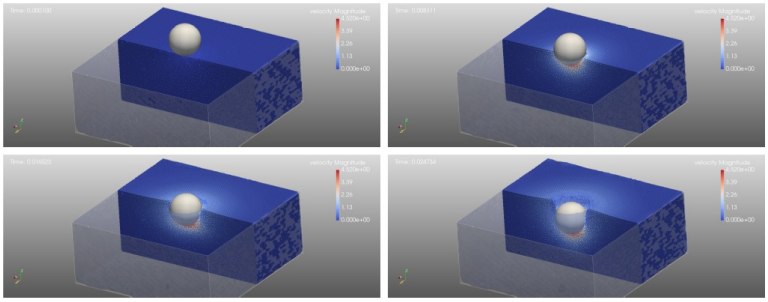


Figure 1: Evolution of the simulation of a sphere impacting a box filled with sand.

If the movement of the sphere should be that of a rigid body, the MOVE -statement has to be adapted accordingly (see %MOVE_rigid%).

DOWNLOAD COMPREHENSIVE EXAMPLE

MESHFREE · GettingStarted · SpecialCases · BasicPhysics · Sand · SandPileDeposition

SandPileDeposition

sand pile deposition

Sand is injected at an upwards moving inflow and hits a flat surface, see Figure 1. The sand collects on the surface in a growing pile according to the angle of repose which is determined by the coefficient $C_{\mathrm{DruckerPrager}}$ in DruckerPragerModel .

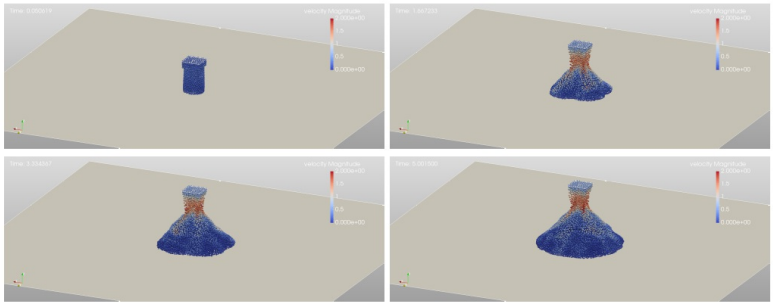


Figure 1: Evolution of sand pile deposition.

DOWNLOAD COMPREHENSIVE EXAMPLE

MESHFREE · GettingStarted · SpecialCases · BasicPhysics · TwoPhaseDarcy

TwoPhaseDarcy

water jet deflected by air

A water jet is deflected by moving air. In general, 2 phases are set up for water and air,
which are then coupled as follows.

Water to air via flow through porous medium

DarcyConstant(Air) = [min(1, projY(2,%ind_kob%))*1.0e5]

DarcyBasisVelocity(Air) = ([projY(2,%ind_v(1)%)], [projY(2,%ind_v(2)%)], [projY(2,%ind_v(3)%)])

Air to water via pressure boundary condition at free surface:

BC_p (0) = (%BND_free_implicit%, [equn($WaterInBox$)*(projY(1,%ind_p%)+projY(1,%ind_p_dyn%))])

Note: This is a completely different coupling approach from the one using BCON_CNTCT .

DOWNLOAD COMPREHENSIVE EXAMPLE

2.3.3. MultiPhaseCoupling

Solve selected test cases in the field of multi-phase simulations

Examples showing the capabilities of MESHFREE in applications where different phases are interacting with each other.

List of members:

ChannelWithFilter	One-Way coupling of droplets and air in channel with filter
PorousBlock	Local flow resistance due to block of porous material
PorousBlockAnisotropic	Local flow resistance due to block of anisotropic porous material
WaterSand	A jet of water and sand hits a plate

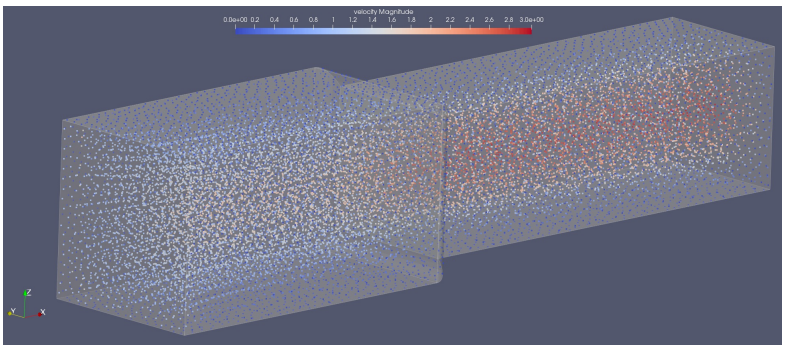
ChannelWithFilter

One-Way coupling of droplets and air in channel with filter

This example showcases some of the capabilities of the DROPLETPHASE solver in representing one-way coupled flow scenarios.

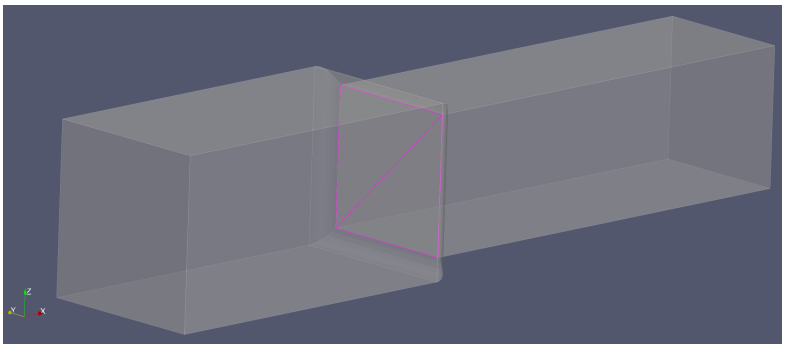
Starting point

As a starting point, we consider a simple flow of air through a channel which has a reduction of its cross-section half-way along the x-axis:



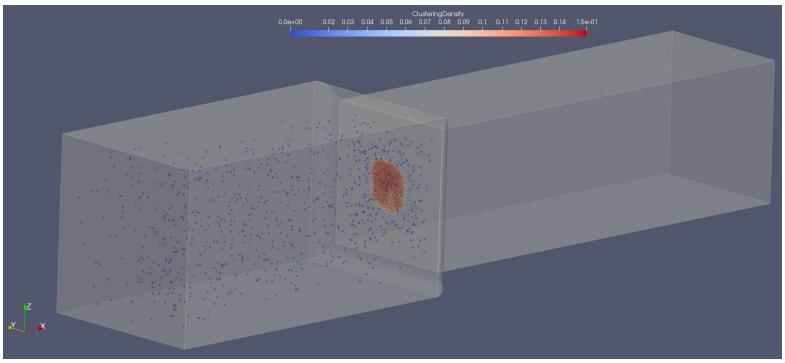
The DROPLETPHASE chamber

At the inlet of this channel, DROPLETPHASE points are added via the DropletSource command. These droplets move under the effect of drag until they hit a wall which is not visible by the fluid phase and can be thougt of as some kind of filter. This wall is depicted below



Simulation result

Running the simulation with the predefined settings will show that particles are gathering at the center of the "filter" wall, as is depicted in the following image:



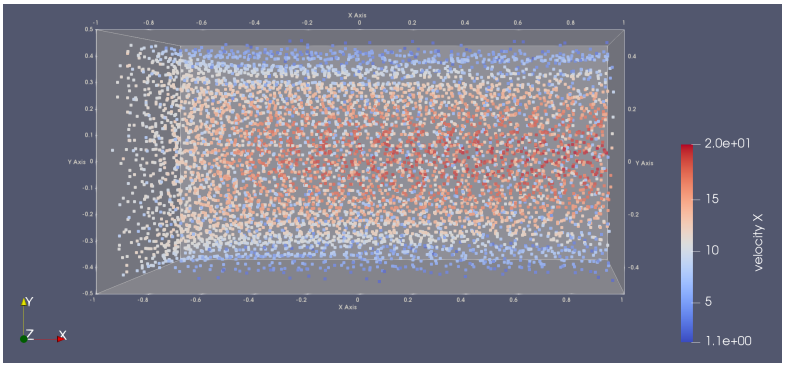
DOWNLOAD COMPREHENSIVE EXAMPLE

PorousBlock

Local flow resistance due to block of porous material

Starting point

As a starting point, let us consider the simple channel flow from tut3d_01 but with an extended channel. Clearly, this results in the following velocity field:



Introducing local flow resistance

In a wide range of applications the fluid is not moving as freely as in the above example. Local flow resistance may be caused by suspended particles of another phase or by a contiguous porous medium, such as filters. To understand how

we can introduce such a flow resistance into the above channel flow, let us assume that a single block of porous material is present within the flow geometry. One way to define this by means of UCV functionalities is through an indicator function:

```
begin_equation{ $BlockIndicator$ }
(Y%ind_x(1)%>-0.25)*(Y%ind_x(1)%<0.25)*(Y%ind_x(2)%>-0.35)*(Y%ind_x(2)%<0.35)*(Y%ind_x(3)%>-0.35)*
(Y%ind_x(3)%<0.35)
end_equation
```

Clearly, this equation will be equal to 1 if and only if points are within the porous material volume.

Since the block is stationary, we further want to prescribe zero velocity for all components of the porous material velocity. We may do this via the following command:

```
DarcyBasisVelocity( $MatUSER$ ) = ( 0.0, 0.0, 0.0)
```

Now that we have properly defined the position and velocity for our porous basis material, we further need to provide a measure of the resistance that the fluid phase experiences when passing through the porous block. To do this, we specify the DarcyConstant while using the above indicator function equation:

```
DarcyConstant( $MatUSER$ ) = ( [ &cDarcy& *equn{ $BlockIndicator$ }] )
```

If only DarcyConstant (for a straight-forward extension see ForchheimerConstant) is specified, this value manifests itself in a momentum sink

$$-\beta \cdot (\mathbf{v} - \mathbf{v}_\beta)$$

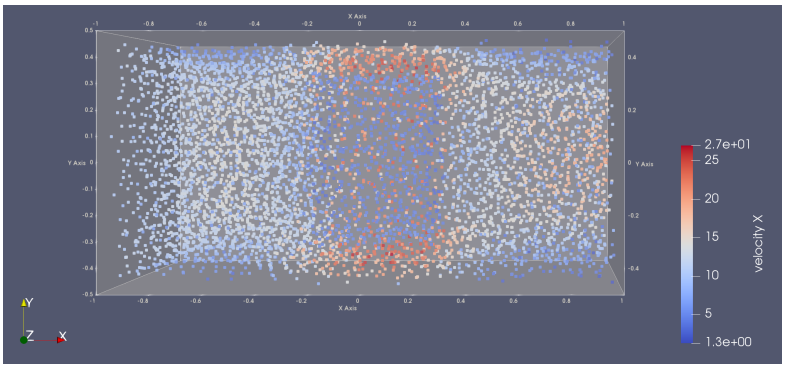
which is added to the momentum equation (see EquationsToSolve) for every point within the volume occupied by the porous material. In the above term, $\mathbf{v}_\beta$ denotes the velocity of the porous material, which we specified above as DarcyBasisVelocity . Note that DarcyConstant actually defines a constant $\tilde{\beta}$ which is related to β via $\beta = \tilde{\beta}\rho$. This steams from the classical formulation of Darcys law

$$-\nabla p = \tilde{\beta}\,(\mathbf{v} - \mathbf{v}_\beta)$$

and the fact that we have $\frac{1}{\rho}\nabla p$ on the left-hand side of the momentum equation in EquationsToSolve .

Simulation results

The image below shows the decrease in fluid velocity due to local flow resistance and further visualizes how the the fluid is accelerated towards the side walls in order to maintain the total mass flow rate:



Representing anisotropic materials

Refer to PorousBlockAnisotropic for the treatment of anisotropic materials.

DOWNLOAD COMPREHENSIVE EXAMPLE

PorousBlockAnisotropic

Local flow resistance due to block of anisotropic porous material

This tutorial is an extension of PorousBlock .

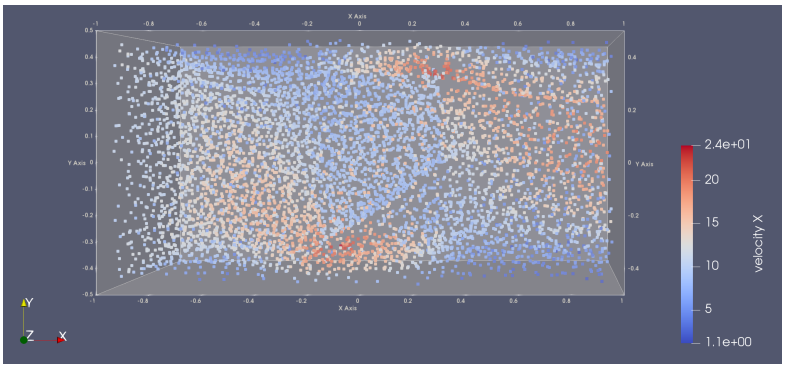
Representing anisotropic materials

In PorousBlock we specified a scalar value of flow resistance for the porous block. What this implies is, that the fluid will experience the same resistance independent of the angle at which it flows through the block. Thus, the block represents an isotropic material. To represent anisotropic materials, one can specify the resistance along three individual directions within the porous material. To visualize this, we rotate the above coordinate system by 45 degrees around the z-axis and prescribe a significantly decreased flow resistance along the x-axis of this rotated system, while the resistance along the other axes remains the same. In the UCV, this may achieved by setting

```
DarcyConstant( $MatUSER$ ) = ( [0.1* &cDarcy& *equn{ $BlockIndicator$ }], 1, 1, 0, ... # Darcy constant in direction of tilted x-axis
[ &cDarcy& *equn{ $BlockIndicator$ }], -1, 1, 0, ... # Darcy constant in direction of tilted y-axis
[ &cDarcy& *equn{ $BlockIndicator$ }], 0, 0, 1 ) # Darcy constant in direction of original z-axis
```

Simulation results

We expect that, due to the decreased flow resistance along the tilted x-axis, the fluid should take a diagonal path through the material. The simulation results below nicely visualize this aspect, with regions of high velocity at the lower left and upper right corner of the porous material:



DOWNLOAD COMPREHENSIVE EXAMPLE

WaterSand

A jet of water and sand hits a plate

A jet of a water-sand mixture is hitting a plate under a skewed angle. The two-phase mixture is modeled with a one-sided Darcy ansatz in the sense that the influence of the water on the sand is respected, but on the other hand the influence the sand has on the water is neglected. The interaction between the sand particles is also neglected.

Darcy ansatz

Water and sand are set to be two different materials. The one-sided interaction between them is ensured by defining a Darcy framework only for the sand phase. Defining two phases, for water the LIQUID solver is chosen, and the sand is modeled in a DROPLETPHASE :

```
KOP(1) = LIQUID LAGRANGE V:IMPLICIT vp- T:NONE # phase 1: water
KOP(2) = DROPLETPHASE # phase 2: sand
```

The velocity of the surrounding water phase is projected onto the Darcy basis velocity for the sand phase:

```
DarcyConstant( $Mat2$ ) = equn{ $DC_Sand$ }
DarcyBasisVelocity( $Mat2$ ) = ( [projY(1,%ind_v(1)%)], [projY(1,%ind_v(2)%)], [projY(1,%ind_v(3)%)] )
```

See EquationsToSolve to understand how the Darcy ansatz is integrated into the conservation of momentum. The DarcyBasisVelocity is the projected velocity of the water phase at the coordinates of a particular sand particle.

Monitor points

Monitor points are used to better track the behavior of the sand phase (see MONITORPOINTS). These points have no influence on the actual simulation and are only used for postprocessing (see Monitor file after running the simulation or look into the integration section of the USER_common_variables file). They are in general defined by user-defined conditions. In our case, a monitor point is created every time a sand particle hits the plate.

```
MONITORPOINTS_CREATION ( $Mat2$ ) = ( %MONITORPOINTS_CREATION_AtBoundary% , equn{ $IsReflected$ } ) # if point is pushed back from the boundary, create a monitor point
MONITORPOINTS_CREATION_FunctionEvaluation ( $Mat2$ ) = ( %ind_addvar(1)% , equn{ $vn$ }, %ind_addvar(2)% , equn{ $vt$ } ) # first index which is used for saving the following quantity
```

Because this quickly generates a lot of monitor points that slow down the simulation, there is a currently unused option at the end of the USER_common_variables file that can be switched on to erase the monitor points in the time step after their creation.

```
## deletion of monitor points in the time step directly after creation
#MONITORPOINTS_DELETION( $Mat2$ ) = (equn{ $mp_delete_in_next_ts$ })
#
## auxilliary equations
#begin_equation{ $mp_delete_in_next_ts$ }
# if ( real( %RealTimeSimulation% ) > Y %ind_st% ) :: 1.0
# else :: 0.0
# endif
#end_equation
```

Auxiliary adjustments

The movement of the sand phase is always disturbed to a small degree

```
COMP_DropletphaseWithDisturbance = 1 # small disturbance of all DROPLETPHASE points (for geometric disturbance shortly after the inflow see below)
# default: 0
```

and in particular directly after the inflow to achieve more realistic results

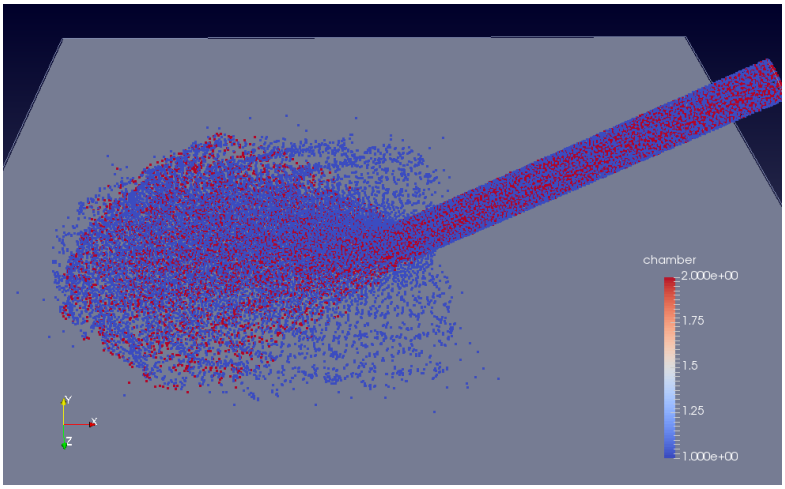
```
# geometric disturbance of the sand points shortly after the inflow
# REMARK: The geometric disturbance can be shut off by commenting the following event.
EVENT (6) = ( equn{ $event_trigger_move_sand_point$ }, %EVENT_FunctionManipulation% , %ind_x(1)% , equn{ $mv_x$ }, %ind_x(2)% , equn{ $mv_y$ }, %ind_x(3)% , equn{ $mv_z$ } )
```

The velocity is scaled for purely numerical reasons; it prevents isolated points from reducing the time step too much with high velocities.

```
# scaling of velocity:
# - water phase -> scaling of velocity only for isolated points (each velocity component is confined to the interval [- &sc_v_ref& * &v_ref& , &sc_v_ref& * &v_ref& ])
# - sand phase -> scaling of all points (each velocity component is confined to the interval [- &sc_v_ref& * &v_ref& , &sc_v_ref& * &v_ref& ])
# REMARK: The scaling can be shut off by commenting the following events or adapting sc_v_ref or v_ref, respectively.
EVENT (1) = ( [1.0], %EVENT_FunctionManipulation% , %indU_flagged_v1%, 0.0, %indU_flagged_v2%, 0.0, %indU_flagged_v3%, 0.0 )
EVENT (2) = ( equn{ $event_trigger_v1$ }, %EVENT_FunctionManipulation% , %ind_v(1)% , equn{ $scaled_v1$ }, %indU_flagged_v1%, 1.0 )
EVENT (3) = ( equn{ $event_trigger_v2$ }, %EVENT_FunctionManipulation% , %ind_v(2)% , equn{ $scaled_v2$ }, %indU_flagged_v2%, 1.0 )
EVENT (4) = ( equn{ $event_trigger_v3$ }, %EVENT_FunctionManipulation% , %ind_v(3)% , equn{ $scaled_v3$ }, %indU_flagged_v3%, 1.0 )
```

Simulation results

A look into some mid-simulation results shows clearly the different behavior of the two phases after hitting the plate. While the water phase (blue) is beginning to cover the plate in all directions, the sand phase (red) resists, due to its higher density, such a change in direction of its movement much stronger.



COMPREHENSIVE EXAMPLE

DOWNLOAD COMPREHENSIVE EXAMPLE

MESHFREE · GettingStarted · SpecialCases · SimulationSplittingWithMEMORIZE

2.3.4. SimulationSplittingWithMEMORIZE

usage of the MEMORIZE-feature to split a simulation

Simulation splitting based on MEMORIZE :

- A water cube falls due to gravity in z-direction. Points passing a certain z-limit are saved and deleted by MEMORIZE_Write .
- The saved points are read in during the second simulation by MEMORIZE_Read . In the end, the water cube is falling as a whole again in z-direction.

Note: With this procedure, different geometries can be analyzed.

DOWNLOAD COMPREHENSIVE EXAMPLE

MESHFREE · GettingStarted · SpecialCases · WaterCrossing

2.3.5. WaterCrossing

Solve selected test cases in the field of water crossing simulations

Examples showing the capabilities of MESHFREE in water crossing applications.

List of members:

SimpleBox	SimpleBox

SimpleBox

List of members:

Classical	simple box driving through a channel of water
FeederCutter	simple box driving through a channel of water
IncreasingNumberOfPoints	simple box driving through a channel of water, after a number of time cycles, the point cloud becomes denser
DifferentTypesOfPressureBoundaryConditions	DifferentTypesOfPressureBoundaryConditions

Classical

simple box driving through a channel of water

A box of 5 meters length, 2 meters width, and 1 m height is driven with constant velocity through a water channel, 20 meters long.

The water height in the channel is 1 meter, the box half-dived into the water.

By movement, it forms a breaking front wave.

For convenience, the two input files of this example are linked into this page in order to easily navigate to the functionalities used.

See especially:
%POINT_APPROXIMATE% as a means of retrieving function values at nodes points of the geometry.

DOWNLOAD COMPREHENSIVE EXAMPLE

DifferentTypesOfPressureBoundaryConditions

List of members:

NonQuasiStationary	various instances of simple box driving through a channel of water, apply differnet pressure BC at each instance
QuasiStationary	various instances of simple box driving through a channel of water, apply differnt pressure BC at each instance

NonQuasiStationary

various instances of simple box driving through a channel of water, apply differnet pressure BC at each instance

Non-quasistationary mode: BOX moves with user given velocity, POOL is at rest.
Besides this, the input files of this example and the ones of QuasiStationary are absolutely identical.

Start several instances of the classical box-in-channel example:

A box of 5 meters length, 2 meters width, and 1 m height is driven with constant velocity through a water channel, 20 meters long. The water height in the channel can be set by the user (default 1m), the box half-dived into the water. By movement, it forms a breaking front wave.

The problem is copied several times. Each copy runs in a different chamber. In each chamber, we apply a dedicated type of boundary condition for the dynamic pressure. The dynamic pressure %ind_p_dyn% is measured at monitor points at the front of the box and written to a .timestep file.

In common_variables, study the behavior of the boundary condition BoundaryConditions.BCON.%ind_p_dyn%.%BND_none% based on the choice of FLIQUID_ConsistentPressure_Version .

List of members:

USER_common_variables	simple box driving through a channel of water: USER_common_variables.dat
Ucv_SinglePoolWithBox	simple box driving through a channel of water: Ucv_SinglePoolWithBox.dat
common_variables	simple box driving through a channel of water: common_variables.dat

DOWNLOAD COMPREHENSIVE EXAMPLE

QuasiStationary

various instances of simple box driving through a channel of water, apply differnt pressure BC at each instance

Quasistationary mode: BOX remains at its original position, POOL moves with the user-given box speed.
Besides this, the input files of this example and the ones of NonQuasiStationary are absolutely identical.

Start several instances of the classical box-in-channel example:

A box of 5 meters length, 2 meters width, and 1 m height is driven with constant velocity through a water channel, 20 meters long. The water height in the channel can be set by the user (default 1m), the box half-dived into the water. By movement, it forms a breaking front wave.

The problem is copied several times. Each copy runs in a different chamber. In each chamber, we apply a dedicated type of boundary condition for the dynamic pressure. The dynamic pressure %ind_p_dyn% is measured at monitor points at the front of the box and written to a .timestep file.

UNlike in the NonQuasiStationary example, here we are allowed to set FLIQUID_ConsistentPressure_Version = 1127 (i.e. use a 1 in the second digit), and the pressure values at the front face of "box" still are in the right order of magnitude, even with %BND_none% .

List of members:

USER_common_variables	simple box driving through a channel of water: USER_common_variables.dat
Ucv_SinglePoolWithBox	simple box driving through a channel of water: Ucv_SinglePoolWithBox.dat
common_variables	simple box driving through a channel of water: common_variables.dat

DOWNLOAD COMPREHENSIVE EXAMPLE

MESHFREE · GettingStarted · SpecialCases · WaterCrossing · SimpleBox · FeederCutter

FeederCutter

simple box driving through a channel of water

The same case as SimpleBox .
However, in order to save computation time, we cut the long pool of
water in front of and behind the vehicle.

The feeder and cutter utilities are implemented in a general way, the can be
treated like functions or subroutines in a normal programming language, therefore see especially:
include_Ucv{ } , and its optional feature
parameters{ }

DOWNLOAD COMPREHENSIVE EXAMPLE

MESHFREE · GettingStarted · SpecialCases · WaterCrossing · SimpleBox · IncreasingNumberOfPoints

IncreasingNumberOfPoints

simple box driving through a channel of water, after a number of time cycles, the point cloud becomes denser

A box of 5 meters length, 2 meters width, and 1 m height is driven with constant velocity through a water channel, 20 meters long.

The water height in the channel is 1 meter, the box half-dived into the water.

By movement, it forms a breaking front wave.

after 200 time cycles, the number of MESHFREE points is subject to steady increase.
For convenience, the two input files of this example are linked into this documentation in order to
easily navigate to the functionalities used.

List of members:

USER_common_variables	simple box driving through a channel of water: USER_common_variables.dat

DOWNLOAD COMPREHENSIVE EXAMPLE

MESHFREE · GettingStarted · SpecialCases · WaterManagement

2.3.6. WaterManagement

Solve selected test cases in the field of water management simulations

Examples showing the capabilities of MESHFREE in water management applications.

List of members:

RainOnSimplePlate	simple rain source

RainOnSimplePlate

simple rain source

Study different aspects like volume control.

List of members:

SophisticatedVolumeControl	study a rain source with sophisticated volume control

SophisticatedVolumeControl

study a rain source with sophisticated volume control

The key point here ist to study the volume correction in detail. DropletSource geneerated droplets which fall on a plate. Then, the water slides down the plate and piles up at a sidewise wall, which acts as an obstacle for the water.

- After collision with the wall, burst into isolated MESHFREE points. Here, volume conservation is crucial.
- At the dam, again volume conservation becomes crucial, as the water collides with the wall initially as a very thin layer.
- The water flow is cut below the geometry by an EVENT statement, here another time volume conservation becomes crucial, because the volume packages of the MESHFREE points deleted are weak, but fully go into the computation of the target volume.

So, volume correction is essential in this example. We study four cases:

- SLIP condition with classical point cloud organization along the walls
- NOSLIP condition with classical point cloud organization along the walls
- SLIP condition with EXTENDED point cloud organization along the walls
- NOSLIP condition with EXTENDED point cloud organization along the walls

EXTENDED point cloud organization is currently experimental and is invoked in common_variables.dat by the line who_am_I = 'FLSLIP'

The volume correction is based an a Ucv-implementation. The main file is Ucv_VolumeCorrection .
This procedure will perform the volume correction in a similar way as the parameters VOLUME_correction and VOLUME_correction_FreeSurface would do.

The Ucv_VolumeCorrection uses another procedure Ucv_ComputeAdaptedTargetVolume , which limits the volume per time the can be deleted by EVENT or METAPLANES and recomputes the adapted target volume.

List of members:

Ucv_VolumeCorrection	implementation of the volume correction as a Ucv-procedure
Ucv_ComputeAdaptedTargetVolume	recompute the target volume due to a given maximum volume flux

DOWNLOAD COMPREHENSIVE EXAMPLE

2.4. Tutorial

simple, comprehensive examples in 3D

- Each tutorial covers several important features of MESHFREE .
- We suggest a Linux system. If working under Windows, please consider installing a virtual machine.
- To run MESHFREE , have a look at MESHFREE.InstallationGuide.Execute .
- The tutorials are ready to run. No preprocessing is necessary in the first place.
  Nevertheless, play around with the parameters given in the input files.
- See Download for archives of example setup suites.

List of members:

tut3d_00	TUTORIAL 0: Checking the geometry
tut3d_01	TUTORIAL 1: flow in a simple tube
tut3d_02	TUTORIAL 2: flow out of a tank
tut3d_03	TUTORIAL 3: flow in open channel with obstacle
tut3d_04a	TUTORIAL 4: flow around a cylinder with local refinement
tut3d_04b	TUTORIAL 4b: flow around a cylinder with local refinement (geometry-based)
tut3d_05	TUTORIAL 5: flow around a MOVING cylinder with MOVING local refinement
tut3d_06	TUTORIAL 6: flow around a periodically moving cylinder
tut3d_07	TUTORIAL 7: boiling flow in a bowl
tut3d_08	TUTORIAL 8: simple pressing process
tut3d_09	TUTORIAL 9: simple floating process
tut3d_10	TUTORIAL 10: simple rolling process

2.4.1. tut3d_00

TUTORIAL 0: Checking the geometry

Goals of this Unit

- Getting to know the requirements for geometry.

- Determination of the orientation of surfaces and lines as well as the definition of filling processes.
- Parameter SimCut in common_variables.dat.
- How to check the boundary normals.

The setting for this tutorial is found in the folder tut3d_00 .

Geometry files for MESHFREE

Usually, one of the first things to do in setting up a MESHFREE simulation is to check the geometry. In MESHFREE , the major available geometry formats are:

- stl
- obj
- msh
- fdneut

MESHFREE requires

- the geometry to be "watertight",
- the geometry to have consistently oriented normals,
- each part of the geometry to be uniquely labeled.

Exercises

In our example, we have the geometry file cube.msh containing a cube with the six faces labeled "top", "bottom", "in", "out", "back", and "front".

```
$MeshFormat
2.2 0 8
$ EndMeshFormat
$PhysicalNames
6
2 1 "top"
2 2 "bottom"
2 3 "in"
2 4 "out"
2 5 "back"
2 6 "front"
$ EndPhysicalNames
...
```

It is included into the simulation model in USER_common_variables.dat by:

```
include{ cube.msh}, scale{ 1, 1, 1}, offset{ -0.5,-0.5,0}
```

The geometry can be modified by GeometryManipulations such as scale{ or offset{ . What about the pointcloud and the generation of the point cloud? If we are not sure about the orientation of the boundary elements, we can use the option

```
SimCut = 4
```

in common_variables.dat, the initial point cloud generation stops after 4 cycles of the point filling procedure. The program is then stopped for checking the result of the initial filling. This might for example yield the configuration in Figure 1. If the orientation of some boundary partition is wrong (picture left), we see that the point cloud is generated on the wrong side.

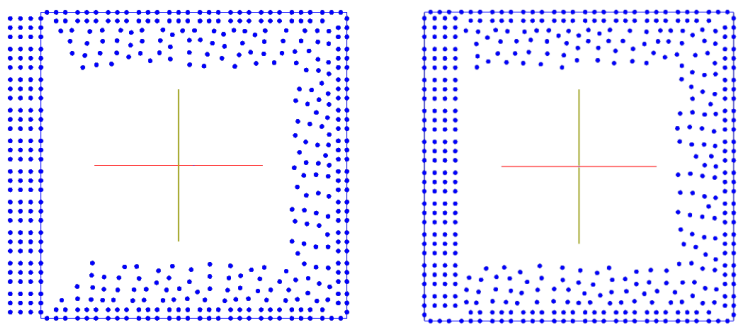


Figure 1: Wrong (left) and correct (right) point cloud generation with SimCut option turned on

Exercise 1 : Play around with the SimCut parameter. Execute the setting as it is and view the result in ParaView. Does the geometry fulfill the MESHFREE requirements?

We would expect the cube to be filled completely, but somehow points get filled outside the domain.

Exercise 2 : In MESHFREE , the interior points are filled in filling cycles startig from the boundary. In order to know in which direction to start, the orientation of the boundary normals is crucial. By convention, the boundary normals point into the flow domain.
Usually, we do not save them for memory reasons, but you can specify that they are written to the boundary elements result file by modifying the SAVE_format in USER_common_variables.dat to

```
SAVE_format (1) = 'ENSIGHT6 BINARY NN-T'
```

Rerun MESHFREE and check the BE_tut3d_0.case file in ParaView. What do you observe?

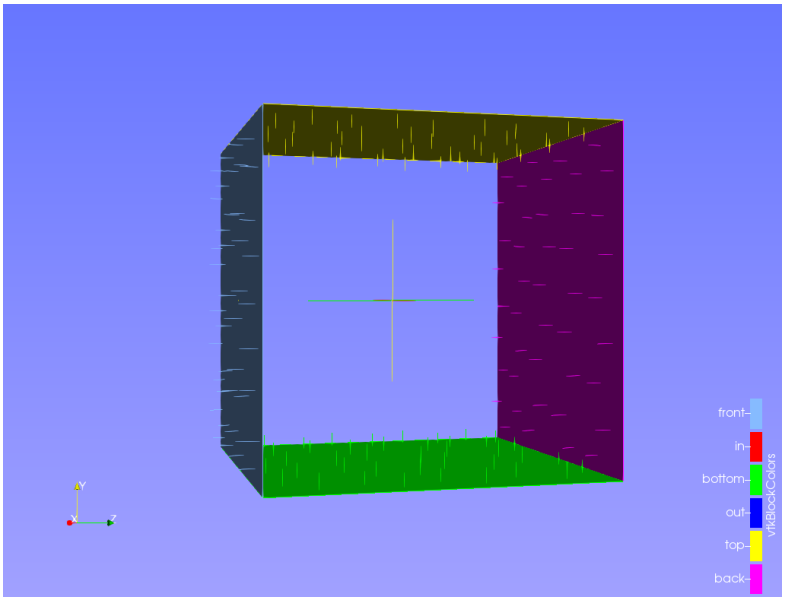


Figure 2: Boundary elements with normal information

The "front" face normal is oriented outwards, and all other face normals are oriented inwards.

Exercise 3 : How can you modify the example such that the filling of MESHFREE points will be correct? Check out the keyword REV_ORIENT in the documentation. Verify your guess by commenting the parameter SimCut : the simulation should then start normally.

Note: In order to reproduce Figure 2, load the state file tut00_figure2.pvsm in ParaView and choose 'Search files under specified directory'. Then, select the correct data directory (MESHFREE results folder).

DOWNLOAD COMPREHENSIVE EXAMPLE

2.4.2. tut3d_01

TUTORIAL 1: flow in a simple tube

Goals of this Unit:

- Setting up of a flow problem “simple channel flow”
- The most important parameters in the file common_variables.dat
- The parameters v-- , vp- and COEFF_dt_virt
- How to define boundaries and aliases in 3D examples

Formation of geometry:

The geometry for this tutorial can be seen in cube.geo in the folder tut3d_01 .

The fluid-mechanical problem:

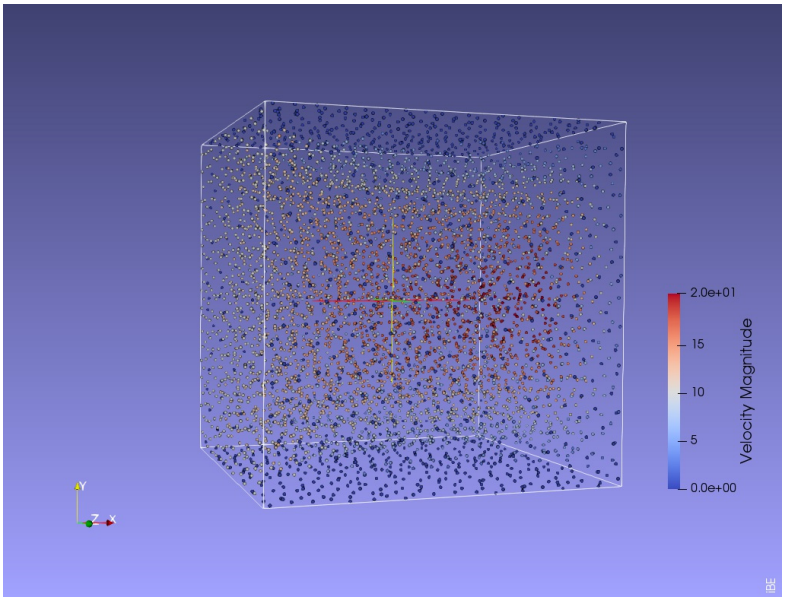


Figure 7: sketch of simulation

The first example is a simple channel flow. At the inlet on the left hand side we assume a constant velocity. There is no velocity at the walls (no-slip boundary condition at the bottom, top, back and front wall). Further there is no gravity present and the pressure at the outlet on the right hand side is zero.
Boundary conditions are defined in the following way at USER_common_variables.dat:

```
BC_T ( $wall$ ) = (%BND_ROBIN%, 10.0, 500.0, 0.3) # BC_T ( $xyz$ ) = (%BND_CAUCHY%, alpha, T0, inertialThickness), i.e. CAUCHY: lambda*dT/dn = alpha*(T-T0)
BC_T ( $in$ ) = ( %BND_DIRICH% , 1500.0) # BC_T ( $xyz$ ) = ( %BND_DIRICH% , T0 ) , i.e. fix the temperature at the boundary to a value of T0
BC_T ( $out$ ) = (%BND_ROBIN%, 0.0, 500.0) # Cauchy condition, see above. This condition mimics a pure insulatoin boundary
BC_T ( $wallt$ ) = (%BND_ROBIN%, 10.0, 500.0, 0.3) # Cauchy condition, see above
BC_p ( $wall$ ) = ( %BND_wall% ) # standard wall pressure condition
BC_p ( $in$ ) = ( %BND_wall% ) # for pressure BC , inflow and wall boundaries behave in the same way
BC_p ( $out$ ) = ( %BND_DIRICH% , 0.0 ) # fix the pressure to be 0 at the outlfow boundary
BC_p ( $wallt$ ) = ( %BND_wall% ) # standard wall pressure condition
BC_v ( $wall$ ) = ( %BND_wall_nosl% ) # standard noslip condition at lower wall
BC_v ( $in$ ) = ( %BND_inflow% , [ &v0& ], 0, 0 ) # inflow velocity prescribed
BC_v ( $out$ ) = ( %BND_NEUMANN% , 0,0,0 ) # standard Neumann condition at the outflow (i.e. keep the velocity free, but fix dv/dn=0)
BC_v ( $wallt$ ) = ( %BND_wall_nosl% ) #( %BND_slip% ) # classical noslip conditions
BCON ( $wall$ , %ind_p_dyn% ) = ( %BND_wall% ) # standard wall pressure condition
BCON ( $in$ , %ind_p_dyn% ) = ( %BND_AVERAGE% ) # for pressure BC , inflow and wall boundaries behave in the same way
BCON ( $out$ , %ind_p_dyn% ) = ( %BND_DIRICH% , 0.0 ) # fix the pressure to be 0 at the outlfow boundary
BCON ( $wallt$ , %ind_p_dyn% ) = ( %BND_wall% ) # standard wall pressure condition
```

In the Alias Section

```
begin_alias{ "BoundaryElements"}
"bottom" = " BC$wall$ ACTIVE$init_always$ IDENT%BND_wall_nosl% MAT$MatUSER$ TOUCH%TOUCH_always%
MOVE$NO_MOVE$ CHAMBER1 " #
"in" = " BC$in$ ACTIVE$init_always$ IDENT%BND_inflow% MAT$MatUSER$ TOUCH%TOUCH_always%
MOVE$NO_MOVE$ CHAMBER1 POSTPROCESS$PP_IN$ " #
"out" = " BC$out$ ACTIVE$init_always$ IDENT%BND_outflow% MAT$MatUSER$ TOUCH%TOUCH_always%
MOVE$NO_MOVE$ CHAMBER1 POSTPROCESS$PP_OUT$ " #
"top" = " BC$wallt$ ACTIVE$init_always$ IDENT%BND_wall_nosl% MAT$MatUSER$ TOUCH%TOUCH_always%
MOVE$NO_MOVE$ CHAMBER1 " #
"front" = " REV_ORIENT BC$wallt$ ACTIVE$init_always$ IDENT%BND_wall_nosl% MAT$MatUSER$
TOUCH%TOUCH_always% MOVE$NO_MOVE$ CHAMBER1 " #
"back" = " BC$wallt$ ACTIVE$init_always$ IDENT%BND_wall_nosl% MAT$MatUSER$ TOUCH%TOUCH_always%
MOVE$NO_MOVE$ CHAMBER1 " #
"dummyPoint"= " ACTIVE$init_always$ MOVE$NO_MOVE$ CHAMBER1 SMOOTH_LENGTH$P_0$ " #
"dummyPoint2"= " ACTIVE$init_always$ MOVE$NO_MOVE$ CHAMBER1 SMOOTH_LENGTH$P_0$ " #
end_alias
```

we have to define all parts of the geometry as read-in in the boundary element section.

The next picture exhibits the generation time of each particle after a certain number of simulation cycles have been completed.

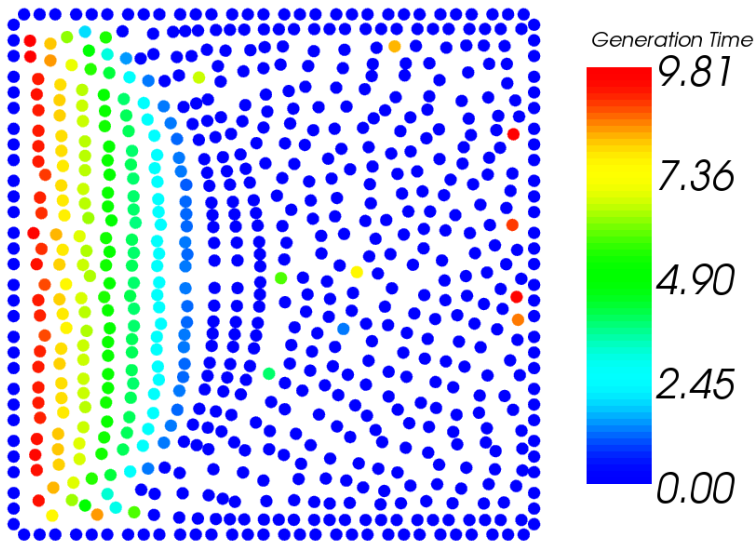


Figure 9: particle generation time after some simulation cycles elapsed

The computation was done using the Lagrange method which we have specified by writing the LAGRANGE flag in the first line

```
KOP(1) = LIQUID LAGRANGE IMPLICIT vp-
```

of “USER_common_variables.dat”. In this example the particles move with the fluid velocity. On the contrary the Euler method (specified by using the keyword EULERIMPL instead of LAGRANGE) leaves the particle cloud fixed. In general the Euler method works fine for stationary flows whereas the Lagrange method is more suitable for transient problems. The difference between these two methods can be seen by watching the animation in ParaView with the “Points of Surface” representation turned on (this shows the particles).

The option flags “IMPLICIT” and “vp-” specify the penalty scheme for the implicit formulation, see vp- . The coupling of the simultaneous computation of velocity and pressure is controlled by the COEFF_dt_virt value in “common_variables.dat”. COEFF_dt_virt represents the factor A in the scheme for the virtual time step size Δt_{virt} . The highest coupling is given for COEFF_dt_virt=0.0, because then we explicitly demand $\nabla^T \mathbf{v} = 0$, however the linear solver might not converge for such strong request. For values of COEFF_dt_virt bigger than zero, we penalize values of $\nabla^T \mathbf{v} \neq 0$ with a certain pressure. Higher values indicate less coupling (penalizing), which can be necessary if the linear solver does not converge

well. COEFF_dt_virt=0.1 is usually a good choice, already leading to very satisfactory results with invisible com

For Reynolds numbers of order 0.1 or greater we can also use the Chorins reprojection scheme. The corresponding flag is “v--”, see v-- . However the scheme v-- becomes unstable if COEFF_dt_virt is chosen too small, so in case of unstable results, this value should be increased.

The Reynolds number for this problem is in the order of magnitude of 1. Consequently the computation works fine with both methods.

Suggestions for exploring FPM:

- play around with the smoothing length (SMOOTH_LENGTH) -> use more or fewer MESHFREE points
- check vp- and v--
- especially check v-- for smaller and smaller Re-numbers (increase eta)
- in the boundary elements section, try to make the tube longer by scaling it, for example, in the x-direction

Advanced Example: flow in a Y_piece (recommended after successfull training according to the basic units)

Note: In order to reproduce Figure 7, load the state file tut01_figure7.pvsm in ParaView and choose 'Search files under specified directory'. Then, select the correct data directory (MESHFREE results folder).

List of members:

Y_piece	flow in a Y-piece

DOWNLOAD COMPREHENSIVE EXAMPLE

Y_piece

flow in a Y-piece

INTEGRATION -statements are introduced to measure the mass flows through the two inflows and the outflow. VOLUME_correction is switched on to reduce mass loss.

2.4.3. tut3d_02

TUTORIAL 2: flow out of a tank

Goals of this unit:

- free surfaces with boundary conditions in 3D,
- formation of a proper jet,
- controlling the jet (preventing infinite jet),
- introduction of the gravity vector and other material properties,
- activation and material specification in the alias-section.

The fluid-mechanical problem
This example shows a flow with free surface. The level of the fluid is decreasing in the draining tank because of a circular hole at the front face of the geometry, where a fluid jet will evolve. In Figure 10, geometry has been rotated such that the user can see the outlet at the bottom right side. The velocity and the flow rate of the jet depend on the depth of the fluid.

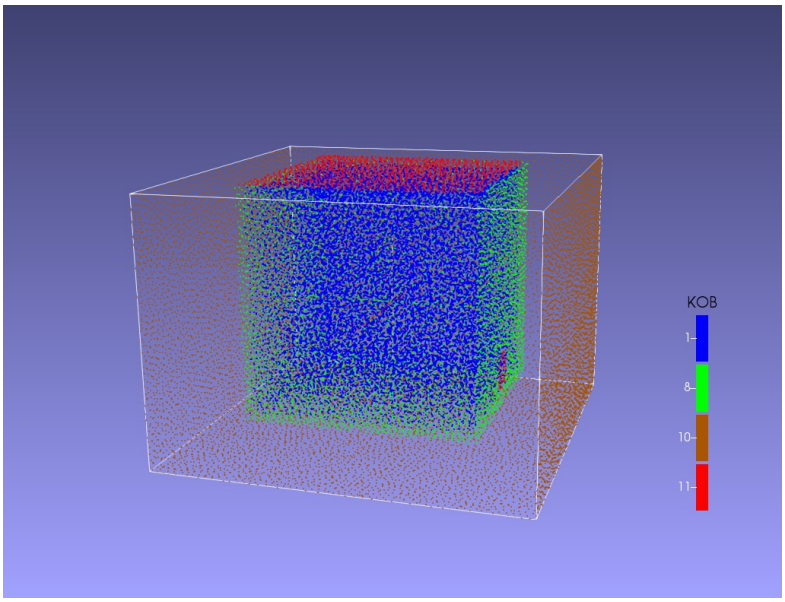


Figure 10: sketch of the simulation

In Figure 10, we observe the outer container (covered by brown points), that encloses the fluid geometry. This container was created to cut the jet, ejected from the orifice hole, and will prevent the formation of an infinite jet. The boundaries of this outer container have been defined in the alias-section in the following way:

```
"cut_side" = " BC$outflow$ ACTIVE$noinit_always$ IDENT%BND_outflow% MAT$MatUSER$
TOUCH%TOUCH_liquid% MOVE$NO_MOVE$ CHAMBER1 POSTPROCESS$PP_OUT$ "
"cut_bottom" = " REV_ORIENT BC$outflow$ ACTIVE$noinit_always$ IDENT%BND_outflow% MAT$MatUSER$
TOUCH%TOUCH_liquid% MOVE$NO_MOVE$ CHAMBER1 POSTPROCESS$PP_OUT$"
```

Free surface detection
As the setting contains free surfaces, we turn on the free surface detection by setting the parameter compute_FS to "YES" in either the USER_common_variables or the common_variables file:

```
# Set parameter compute_FS='YES' either in Ucv or cv to turn on detection of free surfaces:
compute_FS = 'YES'
```

Consistent geometry:
The MESHFREE points of the jet through the outlet orifice would be deleted, if they would see any geometry part from its back-side. So, to get rid of this situation, one must prepare the geometry in such a way that any point can uniquely determine its inside/outside status regarding the geometry model (boundary elements). Figure 11 shows the proper geometry modeling (inner AND outer skin of the tank).

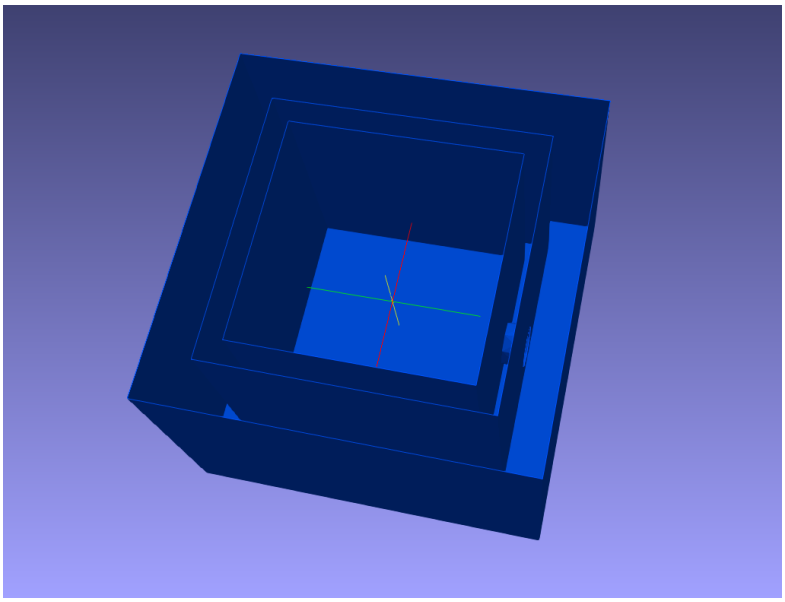


Figure 11: geometrical setup of the problem

The outer skin prevents MESHFREE points from being deleted once they pass through the orifice hole. The outer skin of the tank is defined by:

```
"plane1" = " BC$wall$ ACTIVE$noinit_always$ IDENT%BND_slip% MAT$MatUSER$ TOUCH%TOUCH_liquid%
MOVE$NO_MOVE$ CHAMBER1 "
"plane2" = " BC$wall$ ACTIVE$noinit_always$ IDENT%BND_slip% MAT$MatUSER$ TOUCH%TOUCH_liquid%
MOVE$NO_MOVE$ CHAMBER1 "
```

Flow and boundary conditions

In order to provoke the flow through the orifice hole (driven by hydrostatic pressure), we introduce the gravity vector:

```
gravity( $MatUSER$ ) = (0.0, -9.81, 0.0)
```

The gravity vector (body forces) is a physical property of the specified material $MatUSER$. It is listed in the “USER_common_variables.dat” together with the other material properties such as density, viscosity and initial temperatures.

The relevant boundary conditions are

```
BC_p (0) = ( %BND_free% ) # fallback for free surfaces
BC_p ( $free0$ ) = ( %BND_free% )
BC_p ( $out$ ) = ( %BND_DIRICH% , 0.0)
BC_p ( $wall$ ) = ( %BND_wall% )
BC_p ( $outflow$ ) = ( %BND_wall% )

#BC_v - velocity conditions
BC_v (0) = ( %BND_free% ,0,0,0,0.3) # fallback for free surfaces
BC_v ( $free0$ ) = ( %BND_free% ,0,0,0,0.3) # the last number 0.3 is the inertial thickness, i.e. incorporate inertial
forces into the free surface boundary conditions, see FPMDOCU
BC_v ( $out$ ) = ( %BND_outflow% )
BC_v ( $wall$ ) = ( %BND_slip% ,0,0.3)
BC_v ( $outflow$ ) = ( %BND_NEUMANN% , 0.0, 0.0, 0.0)

#BCON_pCorr - dynamic pressure conditions
BCON (0,%ind_p_dyn%) = ( %BND_free% ) # fallback for free surfaces
BCON ( $free0$ ,%ind_p_dyn%) = ( %BND_free% )
BCON ( $out$ ,%ind_p_dyn%) = ( %BND_DIRICH% , 0.0)
BCON ( $wall$ ,%ind_p_dyn%) = ( %BND_wall% )
BCON ( $outflow$ ,%ind_p_dyn%) = ( %BND_wall% )
```

The boundary index flag $free0$ defines the boundary conditions at the free surface. In the ALIAS section, the top wall is

specified by the flag ACTIVE $free_surface$ (see below), which means, that the border is active during pointfilling and preparation, after start-up it is switched off, turning all points belonging to "top" automatically into free surface points.

```
"top" = " BC$free0$ ACTIVE$free_surface$ MAT$MatUSER$ CHAMBER1 "
```

Typically there are at least the following three ACTIVE statements present:

```
ACTIVE ( $init_always$ ) = ( %ACTIVE_init% , %ACTIVE_always% )
ACTIVE ( $free_surface$ ) = ( %ACTIVE_init% )
ACTIVE ( $noinit_always$ ) = ( %ACTIVE_noinit% , %ACTIVE_always% )
```

The ACTIVE ($init_always$) flag is used for walls which are initially filled and are active throughout the computation. For walls which are not active initially but might come into contact with the fluid (and thus become active) the ACTIVE ($noinit_always$) flag is defined. Finally ACTIVE ($free_surface$) specifies surfaces which are initially filled with points and then immediately switched to the free surface boundary condition.

Use temperature to colorize the material

We use the temperature to simply colorize the material (choosing very small heat conductivity) and isolation boundary conditions:

```
BC_T (0) = (%BND_ROBIN%, 0.0, 0.0, 0.3) # fallback for free surfaces
BC_T ( $free0$ ) = (%BND_ROBIN%, 0.0, 0.0, 0.3)
BC_T ( $out$ ) = (%BND_ROBIN%, 0.0, 0.0, 0.3)
BC_T ( $wall$ ) = (%BND_ROBIN%, 0.0, 0.0, 0.3)
BC_T ( $outflow$ ) = (%BND_ROBIN%, 0.0, 0.0, 0.3)
```

The temperature is initialized due to the y-component of their initial positions:

```
INITDATA ( $MatUSER$ ,%ind_T%) = [Y %ind_x(2)% ] # colorize/initialize temperature by y-values
```

Output files

In the result folder, MESHFREE will generate two kinds of files. The result file starting with BE_... contains the boundary elements. With this, the user has a feedback, how FPM interpreted the geometry from the input files given in the begin_boundary_elements{ } environment. The other result file contains the pointcloud together with the result items defined in the SAVE_ITEM section.

The user can check the “free surface particles” by observing the pointcloud result file with (item "KOB"), as shown in Figure 10, there red particles are free surface particles.

For this tutorial we have chosen special output such that deactivated particles can be seen in ParaView. The activation status can be checked using the item “Activation” which is 0 if the particle is deactivated or it shows the number of time cycles it has been activated without interruption.

Use outer boundary as wall

As an option, the user can switch the fluid behavior at the outflow-box by changing the boundary conditions from $outflow$ to $wall$. In this case, the jet becomes reflected as if the outer box was a wall, the liquid will flow down along the wall due to the given gravity. See the commented lines:

```
#"cut_side" = " BC$wall$ ACTIVE$noinit_always$ IDENT%BND_slip% MAT$MatUSER$ TOUCH%TOUCH_liquid%
MOVE$NO_MOVE$ CHAMBER1 "
#"cut_bottom" = " REV_ORIENT BC$wall$ ACTIVE$noinit_always$ IDENT%BND_slip% MAT$MatUSER$
TOUCH%TOUCH_liquid% MOVE$NO_MOVE$ CHAMBER1 "
```

Suggestions to explore MESHFREE :

- play with the interaction radius SMOOTH_LENGTH
- switch the boundary conditions of the out bounds from outflow to solid wall conditions

Note: In order to reproduce Figures 10 and 11, load the state files tut02_figure10.pvsm and tut02_figure11.pvsm in ParaView and choose 'Search files under specified directory'. Then, select the correct data directory (MESHFREE results folder).

2.4.4. tut3d_03

TUTORIAL 3: flow in open channel with obstacle

Goals of this Unit:

- Discussion of “open Edges”.
- Understanding the normals and volume relation (while making geometry with GMSH).

The fluid-mechanical problem:
In this example the fluid flows around a cylinder and generates a small hump at the free surface. Now we have to take into account that the height of the fluid at the outflow wall is not fixed and might vary in time. In particular it might overflow the original box. In order to avoid that the fluid flows over an edge of thickness zero we have to extend the geometrical model (which is called roof in the formation of the geometry). We briefly have a look at the changes needed to be done in USER_common_variable.dat.

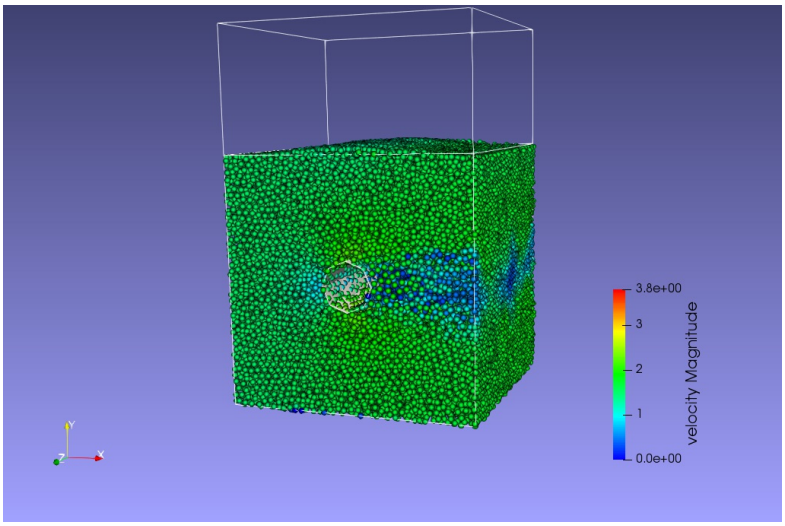


Figure 12: sketch of the problem

It can be easily observed that the roof above the cube is necessary to provide proper closing of the geometry in order to avoid that the fluid flows over the wall.

Healing wrong orientation of geometry items:
While defining aliases in USER_common_variable.dat, boundaries whose orientation is wrong, need to be equipped with the flag REV_ORIENT . If you are working with GMSH, the boundary orientation can be easily seen if displaying the normals of the geometry:

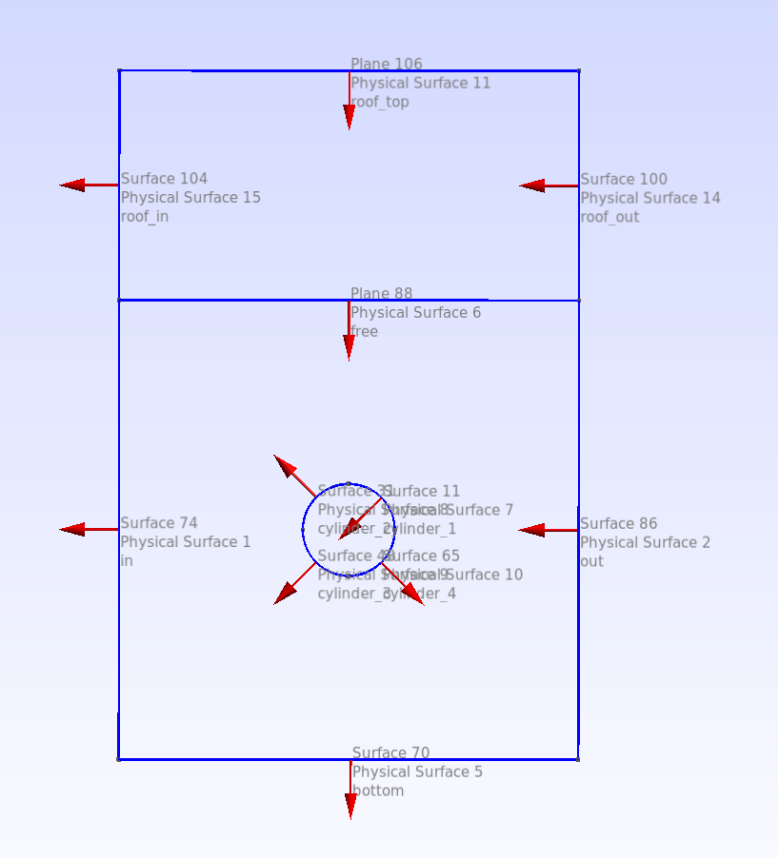


Figure 13: showing orientations and directions of the normals

If working with different preprocessing tools, usually there is a way to display boundary orientations in most of the systems, sometimes however not easy to find. Figure 13 shows the front look of the 3D geometry of this tutorial and also the normals of the surface of the cylinder (please observe the inconsistent formation of the boundary normals, **the normals always how to point to the interior of the flow domain, however GMSH displays the normals the other way around**). Thus, for the appropriate face, we turn around the orientation by REV_ORIENT :

```
"cylinder_1" = " BC$wallCyl$ ACTIVE$init_always$ IDENT%BND_slip% MAT$MatUSER$ TOUCH%TOUCH_always%
MOVE$NO_MOVE$ CHAMBER1 "
"cylinder_2" = " REV_ORIENT BC$wallCyl$ ACTIVE$init_always$ IDENT%BND_slip% MAT$MatUSER$
TOUCH%TOUCH_always% MOVE$NO_MOVE$ CHAMBER1 "
"cylinder_3" = " REV_ORIENT BC$wallCyl$ ACTIVE$init_always$ IDENT%BND_slip% MAT$MatUSER$
TOUCH%TOUCH_always% MOVE$NO_MOVE$ CHAMBER1 "
"cylinder_4" = " REV_ORIENT BC$wallCyl$ ACTIVE$init_always$ IDENT%BND_slip% MAT$MatUSER$
TOUCH%TOUCH_always% MOVE$NO_MOVE$ CHAMBER1 "
```

Closing the geometry on the top:
The “roof” should not contribute in the formation of the point cloud, therefore, the aliases of these walls should be for example defined as follows :

```
"roof_in" = " BC$free0$ ACTIVE$noinit_always$ IDENT%BND_outflow% MAT$MatUSER$
TOUCH%TOUCH_geometrical% MOVE$NO_MOVE$ CHAMBER1 "
"roof_out" = " REV_ORIENT BC$free0$ ACTIVE$noinit_always$ IDENT%BND_outflow% MAT$MatUSER$
TOUCH%TOUCH_geometrical% MOVE$NO_MOVE$ CHAMBER1 "
"roof_back" = " REV_ORIENT BC$free0$ ACTIVE$noinit_always$ IDENT%BND_outflow% MAT$MatUSER$
TOUCH%TOUCH_geometrical% MOVE$NO_MOVE$ CHAMBER1 "
"roof_front" = " BC$free0$ ACTIVE$noinit_always$ IDENT%BND_outflow% MAT$MatUSER$
TOUCH%TOUCH_geometrical% MOVE$NO_MOVE$ CHAMBER1 "
"roof_top" = " REV_ORIENT BC$free0$ ACTIVE$noinit_always$ IDENT%BND_outflow% MAT$MatUSER$
TOUCH%TOUCH_geometrical% MOVE$NO_MOVE$ CHAMBER1 "
```

The ACTIVE statement in the alias definition is ACTIVE $noinit_always$ which tells MESHFREE that this boundary shall not be active during MESHFREE initialization/startup, but has to be active during time integration/simulation.

Suggestions for exploring MESHFREE :

- work with more or less MESHFREE points by adapting the smoothing length
- work with different speeds of the liquid

Advanced Example: FormationFreeJet (recommended after successfull training according to the basic units)

Note: In order to reproduce Figure 12, load the state file tut03_figure12.pvsm in ParaView and choose 'Search files under specified directory'. Then, select the correct data directory (MESHFREE results folder).

List of members:

FormationFreeJet	formation of a free jet

DOWNLOAD COMPREHENSIVE EXAMPLE

MESHFREE · GettingStarted · Tutorial · tut3d_03 · FormationFreeJet

FormationFreeJet

formation of a free jet

A flow through a pipe forms a free jet at the end of the pipe. The free jet hits an inclined plate. The usage of the Selection -feature to control the simulation setup is demonstrated.

MESHFREE · GettingStarted · Tutorial · tut3d_04a

2.4.5. tut3d_04a

TUTORIAL 4: flow around a cylinder with local refinement

Goals of this Unit:

- Problem Specific Variation of the Smoothing Length (and thus the Particle Density)

The fluid-mechanical problem
The fluid mechanical problem and the geometrical setting remains the same as in Tutorial tut3d_03 . However, it might be desirable to have a denser particle cloud around the obstacle in the center of the flow in 3D.

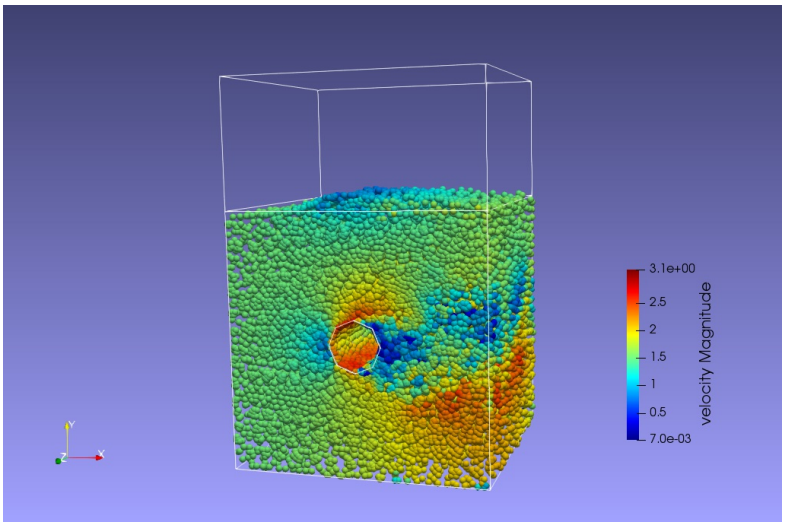


Figure 14: Local pointcloud refinement around the cylinder

In order to use a variable, locally refined smoothing length (which determines the particle density) the keyword 'DSCR' is needed.
In this example, the smoothing length is of a cylindrical distributed density around a line/axis running through a given point. The point is defined by (especially check the SMOOTH_LENGTH -flag):

```
BND_point &hPoint& 0.5 0.5 0.0 # create a point in the middle of the cylinder
...
"hPoint" = "SMOOTH_LENGTH$P_0$ ACTIVE$init_always$ MOVE$NO_MOVE$ CHAMBER1 "
```

The smoothing length about the flagged point is defined by:

```
USER_h_funct = 'DSCR'
SMOOTH_LENGTH ( $P_0$ ) = ( %H_radial% , 0.07, 0.1, 0,0,1, 0.2, 0.3 )
```

Have a look in the SMOOTH_LENGTH documentation to see the full spectrum of defining locally refined smoothing length (interaction radius).
In our special case here, we use %H_radius%, allowing to refine around a given axis.

Here the minimum smoothing length at the cylinder is the first parameter, which is kept at this value in a close neighborhood around the axis (second parameter). The axis of the cylinder is the line going through the point P_0 with direction vector (0,0,1) (third to fifth parameter). Outside this cylinder, the smoothing length increases with the given increase rate up to the maximum allowed smoothing length (last two parameters).

Suggestions to explore FPM

- play around with the parameters in the smoothing length definition,
- use additional sources of refinements (i.e. generate additional BND_point and define a refinement about it),
- go on to example tut3d_04b in order to see how to attach refinement around existing geometry (for example the cylinder).

Note: In order to reproduce Figure 14, load the state file tut04_figure14.pvsm in ParaView and choose 'Search files under specified directory'. Then, select the correct data directory (MESHFREE results folder).

DOWNLOAD COMPREHENSIVE EXAMPLE

2.4.6. tut3d_04b

TUTORIAL 4b: flow around a cylinder with local refinement (geometry-based)

Goals of this Unit:

- attach local refinement to existing geometry items (e.g. the cylinder)

The fluid-mechanical problem

The fluid mechanical problem and the geometrical setting remains the same as in Tutorial tut3d_03 and tut3d_04a . However, it might be desirable to have a denser particle cloud around the obstacle in the center of the flow in 3D and save computation time by thinning out the point cloud far away from the cylinder. In contrast to tut3d_04a , the local refinement of the pointcloud is not prescribed by a virtual axis, but the smoothing length is attached to existing geometrical entities.

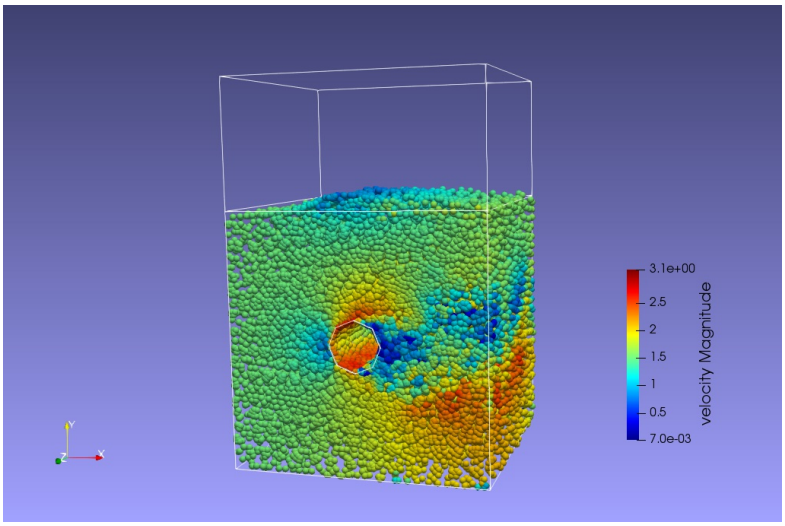


Figure 14: Local refinement of the pointcloud around the cylinder

In order to use a variable, locally refined smoothing length the keyword 'DSCR' is needed.
In this example, the local refinement is attached to the "cylinder"-geometry items given by the geometry. For this, the appropriate elements have to be flagged with the SMOOTH_LENGTH flag:

```
"cylinder_1" = " SMOOTH_LENGTH$P_0$ BC$wallCyl$ ACTIVE$init_always$ IDENT%BND_slip% MAT$MatUSER$
TOUCH%TOUCH_always% MOVE$NO_MOVE$ LAYER0 CHAMBER1 "
"cylinder_2" = " SMOOTH_LENGTH$P_0$ REV_ORIENT BC$wallCyl$ ACTIVE$init_always$ IDENT%BND_slip%
MAT$MatUSER$ TOUCH%TOUCH_always% MOVE$NO_MOVE$ LAYER0 CHAMBER1 "
"cylinder_3" = " SMOOTH_LENGTH$P_0$ REV_ORIENT BC$wallCyl$ ACTIVE$init_always$ IDENT%BND_slip%
MAT$MatUSER$ TOUCH%TOUCH_always% MOVE$NO_MOVE$ LAYER0 CHAMBER1 "
"cylinder_4" = " SMOOTH_LENGTH$P_0$ REV_ORIENT BC$wallCyl$ ACTIVE$init_always$ IDENT%BND_slip%
MAT$MatUSER$ TOUCH%TOUCH_always% MOVE$NO_MOVE$ LAYER0 CHAMBER1 "
```

For the boundary elements, flagged with the SMOOTH_LENGTH flag, we define the local refinement by

```
USER_h_funct = 'DSCR'
SMOOTH_LENGTH ( $P_0$ ) = ( %H_spherical% , 0.07, 0.1, 0.2, 0.3 )
```

Have a look in the SMOOTH_LENGTH documentation in order to have the full spectrum of defining locally refined smoothing length (interaction radius).
In our special case here, we use %H_spherical% , allowing to refine around a point, axis, or geometry.

Here the minimum smoothing length at the cylinder is the first parameter, which is kept at this value in a close neighborhood around the axis (second parameter). Outside this close neighborhood, the smoothing length increases with the given increase rate up to the maximum allowed smoothing length (last two parameters).

Suggestions to explore FPM

- play around with the parameters in the smoothing length definition
- try to attach the smoothing length to other boundary items

Note: In order to reproduce Figure 14, load the state file tut04_figure14.pvsm in ParaView and choose 'Search files under specified directory'. Then, select the correct data directory (MESHFREE results folder).

DOWNLOAD COMPREHENSIVE EXAMPLE

2.4.7. tut3d_05

TUTORIAL 5: flow around a MOVING cylinder with MOVING local refinement

Goals of this Unit:

- Movement of Walls and associated movement of local refinement

The fluid-mechanical problem

Again the fluid mechanical setting remains the same as in the two previous examples. The only difference will be the movement of the cylinder in the center of the channel.

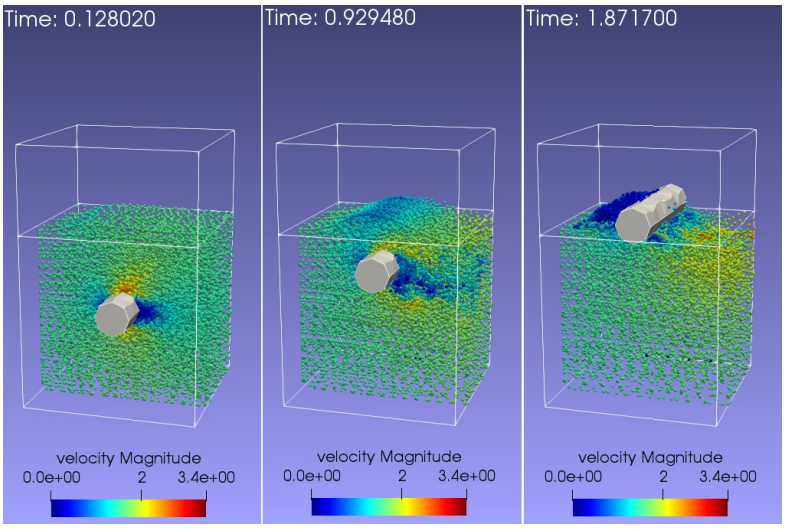


Figure 15: Moving Cylinder perturbing the Fluid Flow

The main tool to move walls, bodies and other geometry elements such as points for smoothing length definitions is the MOVE flag to be given in the alias definition. If we want to move the cylinder in vertical direction, we include the following MOVE statement:

```
MOVE ( $MOVE_circle$ ) = ( %MOVE_velocity% , 0.0, 0.3, 0)
```

Instead if we want to move the cylinder in the x-y-plane with the velocity 0.9 in each direction (x and y) then we may use the MOVE statement in the following way

```
MOVE ( $MOVE_circle$ ) = ( %MOVE_velocity% , 0.9, 0.9, 0)
```

In order to associate the movement with a geometrical entity we have to modify the alias-section, i.e. assign the boundary elements concerned with the appropriate MOVE -flag:

```
"cylinder" = " BC$wallCyl$ ACTIVE$init_always$ IDENT%BND_slip% MAT$MatUSER$
TOUCH%TOUCH_geometrical% MOVE$MOVE_circle$ LAYER0 CHAMBER1 SYMMETRYFACE2 "
```

The higher particle density around the cylinder now will have to move in time, as the cylinder also moves. Thus, we attach the MOVE -flag also to the point around which the smoothing length is defined:

```
"hPoint" = "SMOOTH_LENGTH$P_0$ ACTIVE$init_always$ MOVE$MOVE_circle$ CHAMBER1 "
```

In this example the cylinder is not subdivided into different parts of the hull, only the side faces are separated.

Note: In order to reproduce Figure 15, load the state file tut05_figure15.pvsm in ParaView and choose 'Search files under specified directory'. Then, select the correct data directory (MESHFREE results folder).

DOWNLOAD COMPREHENSIVE EXAMPLE

2.4.8. tut3d_06

TUTORIAL 6: flow around a periodically moving cylinder

Goals of this Unit:

- user-defined functions, see especially Equations

The Fluid Mechanical Problem
Once again we keep our setting and only change the movement of the cylinder.

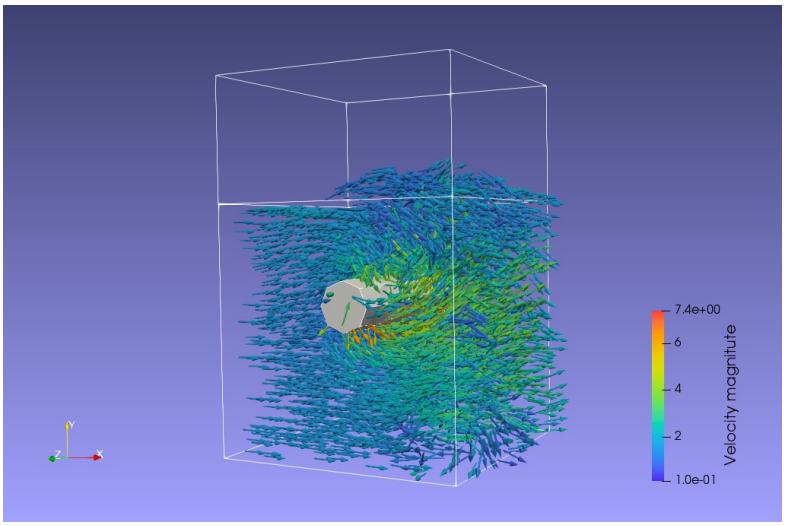


Figure 16: Fluid Flow with periodically moving cylinder

Instead of constant movement, we now want to move it periodically according our own equation:

```
MOVE ( $MOVE_circle$ ) = ( %MOVE_position% , 0, [0.3*sin(15.0*Y %ind_time% +0.0)], 0)
```

Here, as you see, we use the index %ind_time% which stores the current simultion time.

All the other settings are similar to tut3d_05 and tut3d_04a .

temperature as material coloring
Again, we use the temperature as colorizing functionality of the material, in order to visualize the mixing effect of the periodically moved cylinder. For that purpose, we give an extremely small heat conductivity as well as isolating boundary conditions for the temperature.

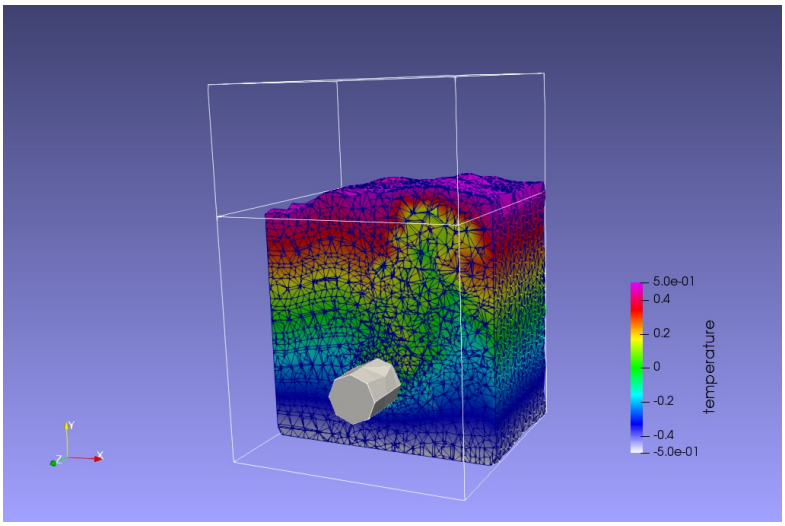


Figure 16b: temperature colorizing the material and thus visualizing the mixing effect of the moving cylinder

Note: In order to reproduce Figures 16 and 16b, load the state files tut06_figure16.pvsm and tut06_figure16b.pvsm in ParaView and choose 'Search files under specified directory'. Then, select the correct data directory (MESHFREE results folder).

DOWNLOAD COMPREHENSIVE EXAMPLE

2.4.9. tut3d_07

TUTORIAL 7: boiling flow in a bowl

Goals of this Unit:

- Further Example for User-defined Functions and Constants
- density (other material items) based on simulation result (such as temperature)

The Fluid Mechanical Problem

A bowl filled with a liquid slowly heats at the bottom and cools at the free surface by radiation and convection. The density of the liquid depends on the temperature. By gravity, the fluid starts to circulate.

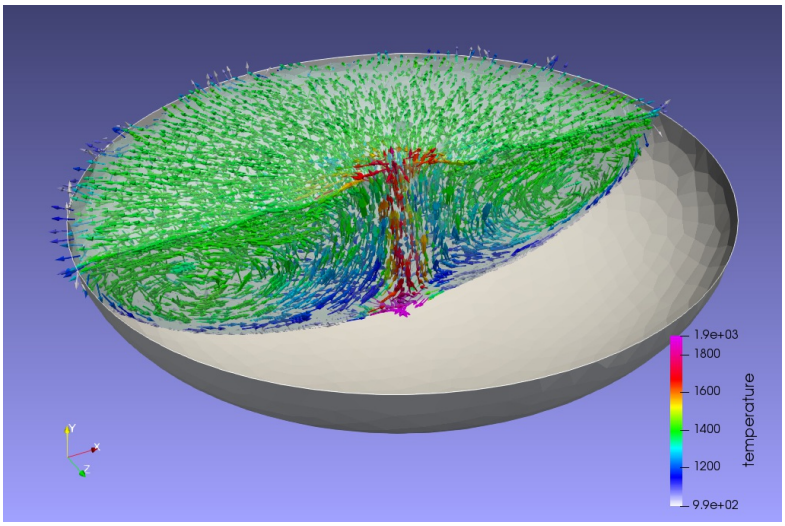


Figure 17: Flow profile and temperature distribution in the bowl at a selected simulation state

Density depending on temperature

The density of the material “XYZ” is not a constant value anymore as in previous tutorials. It is dependent on the temperature and defined by a curve as follows:

```
density( $XYZ$ ) = curve{ $density_XYZ$ }depvar{ %ind_T% } # curve $density_XYZ$ is dependent (leftmost column in the curve definition) on the FPM-simulation item %ind_T% (i.e. temperature)
...
begin_curve{ "density_XYZ"}, nb_functions {1} # curve defining the density based on the temperature
950.0 1000.0
1200.0 970.0
1400.0 870.0
1800.0 760.0
2000.0 730.0
end_curve
```

By “curve{$density_XYZ$} depvar{ %ind_T% }” we tell MESHFREE that the density depends on the variable %ind_T% (the temperature) by the curve:

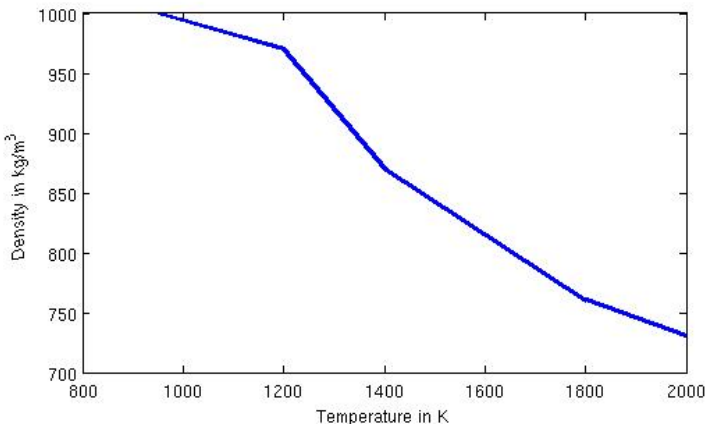


Figure 18: Density depending on the Temperature

The first column in the curve represents the temperature, the second column represents the corresponding density. For temperatures not listed the density is obtained by linear interpolation.

Temperatur boundary conditions dependent on geometrical position

At the free surface, we assume radiation and heat energy convection:

```
BC_T ( $free0$ ) = ( %BND_ROBIN%, equn{ $Radi_Con$ }, &T_ref& )
```

The first parameter for this Cauchy boundary condition is a formula which we put separately into an equation named “Radi_Con”

```
begin_equation{ "Radi_Con"}
&sigma& * &epsilon& *(Y %ind_T% ^3+Y %ind_T% ^2* &T_ref& +Y %ind_T% * &T_ref& ^2+ &T_ref& ^3)+ &convect&
end_equation
```

Inside an equation we have access to all the usual variables. Further, it is advisable to define necessary parameters also in a dedicated alias block:

```
begin_alias{ }
"Spec1" = "%indU_matColor1%" # set up a user-defined index (alsways to be of the form indU_xyz
"sigma" = "5.67E-8"
"epsilon" = "0.3"
"T_ref" = "1000.0"
"convect" = "30"
end_alias
```

The temperature boundary condition for the bottom of the bowl is, again, given by a curve

```
BC_T ( $wall$ ) = ( %BND_ROBIN%, 50000, curve{ $bc_temp$ }depvar{equn{ $x-z-radius$ }} ) # make the curve given in $bc_temp$ dependent from the radius with respect to the x-z-plane
...
begin_curve{ "bc_temp"}, nb_functions {1} # curve defining the enivronment temperature for the temperature-BC base don the x-z-radius of the bowl
0.00 1900.0
0.30 1900.0
0.30 1400.0
0.50 1000.0
10.0 1000.0
end_curve
```

At the center of the bottom we want to have 1900K. Far a way from the center, we have colder temperatures.

Note: In order to reproduce Figure 17, load the state file tut07_figure17.pvsm in ParaView and choose 'Search files under specified directory'. Then, select the correct data directory (MESHFREE results folder).

DOWNLOAD COMPREHENSIVE EXAMPLE

2.4.10. tut3d_08

TUTORIAL 8: simple pressing process

Goals of this Unit:

- Transport Equations for additional Species
- user defined indices
- user defined coloring indices

The Fluid Mechanical Problem
In this tutorial we dip a plunger into a tank filled with a viscous fluid. As shown in the series of images below, the plunger will force the fluid upwards in between the plunger and the side walls.

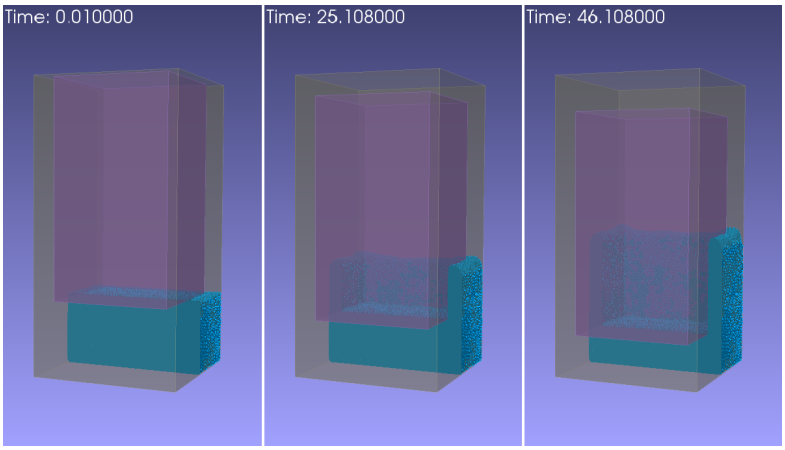


Figure 20a: Fluid at selected simulation states

Initialize and save the color items

In order to see how the fluid interfuses, we define several color species which are assigned to points depending on their initial position. These species are then transported with the point cloud and visualize how the fluid is mixed during the motion enforced by the plunger.

We consider two rather similar ways of saving such a coloring. On the one hand we store the species in the UserDefinedIndices %indU_spec1% and %indU_spec2% which are written out due to the lines

```
SAVE_ITEM = ( %SAVE_scalar%,[Y%indU_spec1%], "spec1")
SAVE_ITEM = ( %SAVE_scalar%,[Y%indU_spec2%], "spec2")
```

in the UCV. This provides us with an access to the values from ParaView. On the other hand we write out discrete UserDefinedIndices for coloring %indC_spec1% and %indC_spec2% via

```
SAVE_ITEM = ( %SAVE_scalar%,[Y%indC_spec1%], "spec1_C")
SAVE_ITEM = ( %SAVE_scalar%,[Y%indC_spec2%], "spec2_C")
```

The difference between these two options will be discussed shortly. In both cases, the initialization of our colorized species is given in the INITDATA -block:

```
INITDATA ( $GLASS$ ,%indU_spec1%) = [equn{ $equn_xBinIdx$ }]
INITDATA ( $GLASS$ ,%indU_spec2%) = [equn{ $equn_yBinIdx$ }]
INITDATA ( $GLASS$ ,%indC_spec1%) = [equn{ $equn_xBinIdx$ }]
INITDATA ( $GLASS$ ,%indC_spec2%) = [equn{ $equn_yBinIdx$ }]
```

where the equations

```
begin_equation{ $equn_xBinIdx$ }
int(Y %ind_x(1)% /1.0*( &nBinX& -1))
end_equation
begin_equation{ $equn_yBinIdx$ }
int(Y %ind_x(2)% /0.5*( &nBinY& -1))
end_equation
```

simply represent a partitioning of the initial pointcloud along the x- and y-direction into the number of bins specified via

```
begin_alias{ }
"nBinX" = "5" #Number of discrete values along x-direction (similar to histogram bins)
"nBinY" = "5" #Number of discrete values along y-direction (similar to histogram bins)
end_alias
```

User defined material index

In FPM, the user is able to define additional indices in order to solve additional simulations tasks, see UserDefinedIndices . They work in the same way as the classical indices, so the user can initialize them, and on top, solve PDE of convection-diffusion-type.

In this tutorial, we used these UserDefinedIndices in order to set up the coloring we discussed above. Taking the vertical coloring stored in %indU_spec2% as an example, the above settings lead to the following simulation snapshots

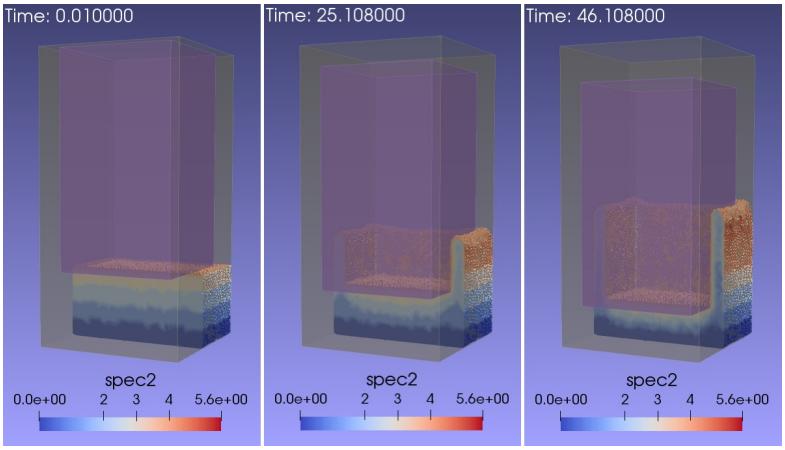


Figure 20b: Fluid colouring via indU_ at selected simulation states

User defined coloring index

While the UserDefinedIndices provide a visually informative representation of mixing, we also observe that the range of values shifts over time. This is due to the fact that these indices are subject to all interpolations that would be applied to other physical variables.

This behavior can be circumvented by the subclass of UserDefinedColorIndices, which always inherit values from parent points instead of employing interpolation procedures. In this way, the original number of discrete values is maintained throughout the simulation.

Consequently, considering the identical snapshots for %indC_spec2% shows an unchanged range of values:

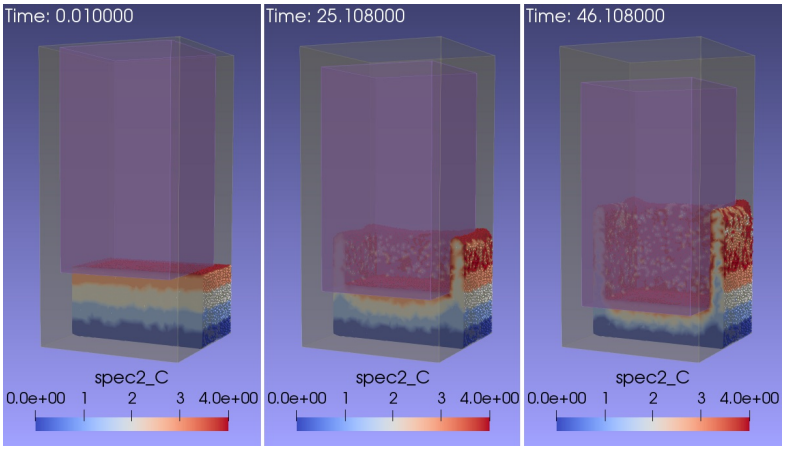


Figure 20c: Fluid colouring via indC_ at selected simulation states

The smearing of initial values when using UserDefinedIndices can also be seen when considering histograms of %indU_spec2% and %indC_spec2% values at the time of the final snapshot:

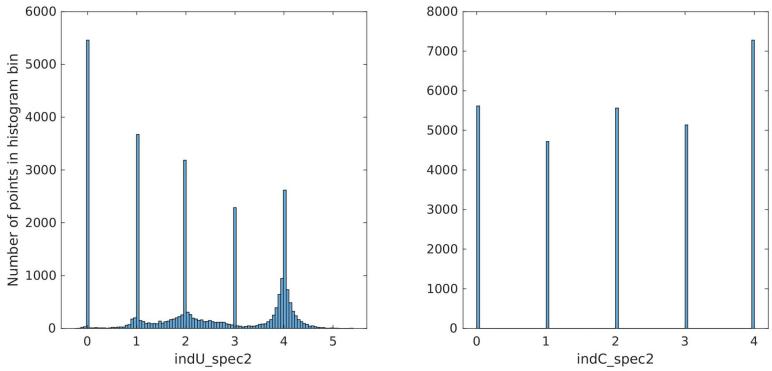


Figure 21: Histograms of species 2 values at time of final snapshot

Note: In order to reproduce Figures 20a, 20b, and 20c, load the state files tut08_figure20a.pvsm, tut08_figure20b.pvsm, and tut08_figure20c.pvsm in ParaView and choose 'Search files under specified directory'. Then, select the correct data directory (MESHFREE results folder).

DOWNLOAD COMPREHENSIVE EXAMPLE

2.4.11. tut3d_09

TUTORIAL 9: simple floating process

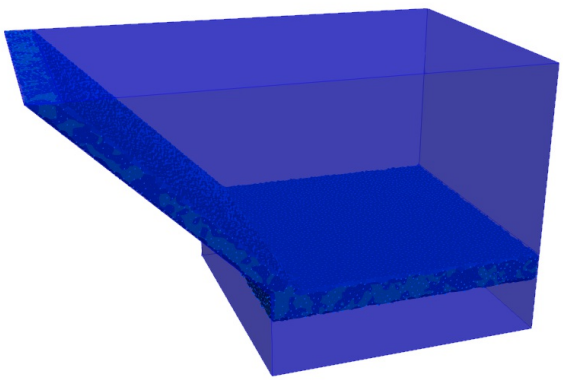
Goals of this Unit:

- Several Free Surfaces
- Symmetrical Model

The Fluid Mechanical Problem

Molten material flows down a ramp onto a bath of liquid support material whose density is bigger than the one of the melt. Thgus, the melt swims on the support bath. The idea of this tutorial stems from the float glass production process, where the melt material is liquid glass, and the support bath is liquid tin. This process is indeed meaningful for many more production processes in industry.

Figure 21: Start Configuration: Glass flows from the upper left to the lower right side



Subdividing the free surface into top and bottom parts
The tin bath on which the glass floats (but which we do not want to compute explicitly) is contained in the empty box at the bottom shown in Figure 21. We only include the buoyant forces acting upon the lower surface so that the glass can dip under the tin level (which we assume to be constant). To this end we assume a free boundary condition for velocity, and a Dirichlet condition for the pressure:

```
BC_T ( $free_bottom$ ) = (%BND_ROBIN%, 2000.0, 1400.0)
BC_p ( $free_bottom$ ) = (%BND_free_implicit%, equn{ $hpressureTin$ }) # the outer pressure is governed by the
diving depth into the support bath
BC_v ( $free_bottom$ ) = ( %BND_free% , 0,0,0, 0.3)

...

begin_equation{ "hpressureTin"}
&gravity& * &Tdensity& *( &Theight& -Y %ind_x(2)% )
end_equation
```

In order to distinguish the lower free surface from the upper one we have given the boundary condition identifiers explicit names (rather than the default “0”). The conditions for the upper free surface are as usual:

```
BC_T ( $free_top$ ) = (%BND_ROBIN%, 100.0, 1400.0)
BC_p ( $free_top$ ) = (%BND_free_implicit%, 0)
BC_v ( $free_top$ ) = ( %BND_free% , 0,0,0, 0.3)
```

The listing of the corresponding geometric entities in the alias section now looks like:

```
"gtop" = " REV_ORIENT BC$free_top$ ACTIVE$free_surface$ MAT$GLASS$ LAYER0 CHAMBER1 "
"gdown" = " BC$free_top$ ACTIVE$free_surface$ MAT$GLASS$ LAYER0 CHAMBER1 "
"gbottom" = " REV_ORIENT BC$free_bottom$ ACTIVE$free_surface$ MAT$GLASS$ LAYER0 CHAMBER1 "
```

where “gtop” and “gdown” are the two rectangular upper faces, "gbottom" is the free surface at the interface to the support bath.

Define symmetry plane
By using a symmetry plane, one can reduce the simulation time, as one considers only a half or a part of the geometrical model. However, it is necessary to provide proper boundary conditions at the symmetry boundary. In our case, the box shown in Figure 21 is, in reality, twice as wide as shown, the back side (the right side when seen from the outflow wall) is the symmetry plane. The boundary conditions are:

```
BC_T ( $sym$ ) = (%BND_ROBIN%, 0, 1400)
BC_p ( $sym$ ) = ( %BND_NEUMANN% , 0.0)
BC_v ( $sym$ ) = ( %BND_NEUMANN% , 0, 0, 0)
```

Here %BND_NEUMANN% defines a pure symmetry condition, as it imposes du/dn=0, that is the normal derivative of the function vanishes. The geometry items belonging to the symmetry-plane are listed here:

```
"gside3" = " BC$sym$ ACTIVE$init_always$ IDENT%BND_slip% MAT$GLASS$ TOUCH%TOUCH_geometrical%
MOVE$NO_MOVE$ LAYER0 CHAMBER1 "
"bwall3" = " BC$sym$ ACTIVE$noinit_always$ IDENT%BND_slip% MAT$GLASS$ TOUCH%TOUCH_geometrical%
MOVE$NO_MOVE$ LAYER0 CHAMBER1 "
"gside6" = " REV_ORIENT BC$sym$ ACTIVE$init_always$ IDENT%BND_slip% MAT$GLASS$
TOUCH%TOUCH_geometrical% MOVE$NO_MOVE$ LAYER0 CHAMBER1 "
"wall3" = " REV_ORIENT BC$sym$ ACTIVE$noinit_always$ IDENT%BND_slip% MAT$GLASS$
TOUCH%TOUCH_geometrical% MOVE$NO_MOVE$ LAYER0 CHAMBER1 "
```

In the beginning, the interface to the support bath swings up and down until finding the equilibrium. After 50s of simulation time, the stationary solution is reached:

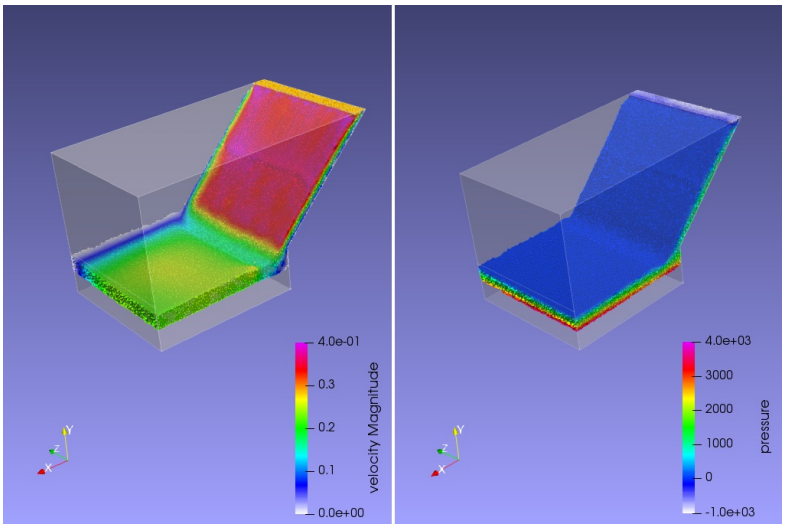


Figure 22: Stationary Solution with a Glance at the Symmetry Plane of the Model

Note: In order to reproduce Figures 21 and 22, load the state files tut09_figure21.pvsm and tut09_figure22.pvsm in ParaView and choose 'Search files under specified directory'. Then, select the correct data directory (MESHFREE results folder).

DOWNLOAD COMPREHENSIVE EXAMPLE

2.4.12. tut3d_10

TUTORIAL 10: simple rolling process

Goals of this Unit:

- several materials and chambers in 3D
- smoothing length definition for chambers, respectively
- tear-off criterion

The Fluid Mechanical Problem

A fluid coming out of a feeder is rolled into a film by two rolls. The rolls are filled with high-viscosity fluids, such that they practically perform a rigid rotation. The rolls are cooled at the inside.

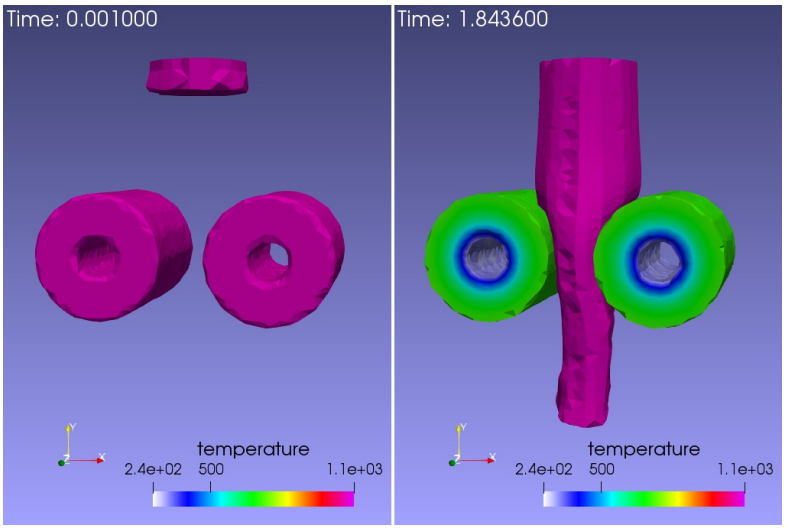


Figure 25: (a) Initial Stage of the Point Cloud; (b) Stage when Jet has been cut

Setting up the problem

Altogether we have three different materials. In order to handle the model with FPM we introduce three chambers, one for each material:

```
KOP(1) = LIQUID V:IMPLICIT T:EXPIMP(1.0) LAGRANGE vp-
KOP(2) = LIQUID V:IMPLICIT T:EXPIMP(1.0) LAGRANGE vp-
KOP(3) = LIQUID V:IMPLICIT T:EXPIMP(1.0) LAGRANGE vp-
```

For each chamber we need to define a smoothing length, so we define three additional points in the Boundary Elements section:

```
BND_point &Point_H_Curve1& 0.0 0.0 0.0
BND_point &Point_P_100& 0.0 0.0 0.0
BND_point &Point_P_200& 0.0 0.0 0.0
```

The rolls do not require a dense particle cloud. In contrast we should use a small smoothing length close to where the two rolls almost touch:

```
USER_h_funct = 'DSCR'
SMOOTH_LENGTH ( $H_CURVE1$ ) = ( %H_constant% , 0.4 )
SMOOTH_LENGTH ( $P_100$ ) = ( %H_constant% , 0.3 )
SMOOTH_LENGTH ( $P_200$ ) = ( %H_constant% , 0.3 )
```

In the alias section we now have to specify the chamber to which the geometric entities belong.

For chamber 1 (the liquid melt), we define

```
"in" = " REV_ORIENT BC$BC_inflow$ ACTIVE$noinit_always$ IDENT%BND_inflow% MAT$GLASS$
TOUCH%TOUCH_always% MOVE$NO_MOVE$ CHAMBER1 "
"out_left" = " BC$BC_outflow$ ACTIVE$noinit_always$ IDENT%BND_outflow% MAT$GLASS$
TOUCH%TOUCH_liquid% MOVE$NO_MOVE$ CHAMBER1 "
"out_right" = " REV_ORIENT BC$BC_outflow$ ACTIVE$noinit_always$ IDENT%BND_outflow% MAT$GLASS$
TOUCH%TOUCH_liquid% MOVE$NO_MOVE$ CHAMBER1 "
"out_back" = " BC$BC_outflow$ ACTIVE$noinit_always$ IDENT%BND_outflow% MAT$GLASS$
TOUCH%TOUCH_liquid% MOVE$NO_MOVE$ CHAMBER1 "
"out_front" = " REV_ORIENT BC$BC_outflow$ ACTIVE$noinit_always$ IDENT%BND_outflow% MAT$GLASS$
TOUCH%TOUCH_liquid% MOVE$NO_MOVE$ CHAMBER1 "
"out_bottom" = " BC$BC_outflow$ ACTIVE$noinit_always$ IDENT%BND_outflow% MAT$GLASS$
TOUCH%TOUCH_liquid% MOVE$NO_MOVE$ CHAMBER1 "
"roll_left_out" = " REV_ORIENT BC$BC_left_out2$ ACTIVE$noinit_always$ IDENT%BND_wall_nosl% MAT$GLASS$
TOUCH%TOUCH_liquid% MOVE$MOVE_RLEFT$ CHAMBER1 "
"roll_right_out" = " REV_ORIENT BC$BC_right_out2$ ACTIVE$noinit_always$ IDENT%BND_wall_nosl%
MAT$GLASS$ TOUCH%TOUCH_liquid% MOVE$MOVE_RRIGHT$ CHAMBER1 "
```

For the two other chambers (rolls), we define

```
"roll_left_front" = " BC$BC_roll_side$ ACTIVE$init_always$ IDENT%BND_slip% MAT$MAT_RLEFT$
TOUCH%TOUCH_always% MOVE$MOVE_RLEFT$ CHAMBER2 "
"roll_left_back" = " BC$BC_roll_side$ ACTIVE$init_always$ IDENT%BND_slip% MAT$MAT_RLEFT$
TOUCH%TOUCH_always% MOVE$MOVE_RLEFT$ CHAMBER2 "
"roll_left_in" = " BC$BC_roll_in$ ACTIVE$init_always$ IDENT%BND_wall_nosl% MAT$MAT_RLEFT$
TOUCH%TOUCH_always% MOVE$MOVE_RLEFT$ CHAMBER2 "
"roll_left_out" = " BC$BC_left_out1$ ACTIVE$init_always$ IDENT%BND_wall_nosl% MAT$MAT_RLEFT$
TOUCH%TOUCH_always% MOVE$MOVE_RLEFT$ CHAMBER2 "
...
"roll_right_front" = " BC$BC_roll_side$ ACTIVE$init_always$ IDENT%BND_slip% MAT$MAT_RRIGHT$
TOUCH%TOUCH_always% MOVE$MOVE_RRIGHT$ CHAMBER3 "
"roll_right_back" = " BC$BC_roll_side$ ACTIVE$init_always$ IDENT%BND_slip% MAT$MAT_RRIGHT$
TOUCH%TOUCH_always% MOVE$MOVE_RRIGHT$ CHAMBER3 "
"roll_right_in" = " BC$BC_roll_in$ ACTIVE$init_always$ IDENT%BND_wall_nosl% MAT$MAT_RRIGHT$
TOUCH%TOUCH_always% MOVE$MOVE_RRIGHT$ CHAMBER3 "
"roll_right_out" = " BC$BC_right_out1$ ACTIVE$init_always$ IDENT%BND_wall_nosl% MAT$MAT_RRIGHT$
TOUCH%TOUCH_always% MOVE$MOVE_RRIGHT$ CHAMBER3 "
#ALIAS_points
"Point_H_Curve1" = " ACTIVE$init_always$ SMOOTH_LENGTH$H_CURVE1$ MOVE$NO_MOVE$ CHAMBER1 "
"Point_P_100" = " ACTIVE$init_always$ SMOOTH_LENGTH$P_100$ MOVE$NO_MOVE$ CHAMBER2 "
"Point_P_200" = " ACTIVE$init_always$ SMOOTH_LENGTH$P_200$ MOVE$NO_MOVE$ CHAMBER3 "
```

Please observe, that "roll_left_out" and "roll_right_out" (the outer skins of the rolls) are defined twice, as they are part of the rolls as well as of the liquid melt.

Especially have a look at the temperature boundary conditions for the contact between the melt and the rolls, where we prescribe a big heat transfer coefficient:

```
BCON_CNTCT ( $BC_left_out1$ ,%ind_T%) = ( %BND_ROBIN%, 200000, 0, 0 ) # almost perfect heat contact
BCON_CNTCT ( $BC_left_out2$ ,%ind_T%) = ( %BND_ROBIN%, 200000, 0, 0 ) # almost perfect heat contact
BCON_CNTCT ( $BC_right_out1$ ,%ind_T%) = ( %BND_ROBIN%, 200000, 0, 0 ) # almost perfect heat contact
BCON_CNTCT ( $BC_right_out2$ ,%ind_T%) = ( %BND_ROBIN%, 200000, 0, 0 ) # almost perfect heat contact
```

In order to release the liquid melt from the rolls, we have to provide tear-off criteria

```
BC_TearOffCriterion ( $BC_left_out2$ ) = equn{ $TearOff$ }
BC_TearOffCriterion ( $BC_right_out2$ ) = equn{ $TearOff$ }
```

Note: In order to reproduce Figure 25, load the state file tut10_figure25.pvsm in ParaView and choose 'Search files under specified directory'. Then, select the correct data directory (MESHFREE results folder).

DOWNLOAD COMPREHENSIVE EXAMPLE

3. InputFiles

Input files used for steering MESHFREE

MESHFREE is mainly steered by two Input files: USER_common_variables.dat and common_variables.dat. In order to start a simulation, these two files need to be present in your project folder.

List of members:

common_variables	input file for development and debugging purposes
USER_common_variables	defines the simulation model: geometry, boundary conditions, material parameters, etc.

3.1. USER_common_variables

defines the simulation model: geometry, boundary conditions, material parameters, etc.

The file USER_common_variables.dat contains the definition of the simulation model, i.e. all physical and geometrical behavior of the fluid phases, boundary conditions, and parts. The file is structured in sections and there is no certain ordering of these sections required.

Necessary sections in USER_common_variables.dat

A simulation model consists at least of the following sections:

- **Solver** section: KindOfProblem , selection of the solver to be used for each simulation chamber (chamber = phase).
- **Physical Properties** Section: PhysicalProperties , to define material properties.
- **Boundary Elements** section: BoundaryElements , to include geometry data into the simulation model.
- **Active** Section: ACTIVE , to define the active/visibility flags for the boundaries. (Active always or only in the beginning? Shall point cloud filling happen from this boundary element?)
- **Move** Section: MOVE , to define movement of the boundary elements.
- **Boundary conditions** Section: BoundaryConditions , to define boundary conditions for quantities of computation, usually velocity, pressure and temperature.
- **Initial conditions** Section: INITDATA , to define initial conditions.
- **Time Step Control** Section: TimeControl , to define simulation time parameters (start/end time, time step sizes).
- **Smoothing Length** Section: SmoothingLength , for specifying the level of discretization for the simulation.
- **Alias** Section: the AliasForGeometryItems section combines the definitions of the previous sections and attaches them to the boundary elements.
- **Saving** Section: the SAVE section specifies the format for the simulation results and which quantities shall be stored.

Syntax of USER_common_variables.dat

The file USER_common_variables.dat (UCV) has its own scripting syntax to define the simulation model. An overview over this syntax can be found in __GeneralRemarks__ .
A general overview over all supported keywords is found in __overview_of_syntax_elements__ .
A quick reference to all predefined variables and constants can be found in Indices and __Constants__ , respectively.
A reference to all parameters that can be defined in both common_varables (CV) as well as UCV can be found in __Parameters__ .

Equations and Curves

A powerful feature of MESHFREE is that the user is very flexible in including measurement data and dependencies between quantities into the simulation model:

- Curves - tabular value depending on one or two variables, ideal for measurement data.
- Equations - ideal for physical relations.

These can for example be flexibly included in evaluating a RightHandSideExpression .

Postprocessing

MESHFREE offers some features for immediate postprocessing of computation results:

- INTEGRATION offers features to calculate integrals over the simulation domain and the boundaries at the end of the timestep.
- UserDefinedIndices allow the user to define additional MESHFREE internal variables.

List of members:	
__DEFAULT_configuration_file__	allows to provide Ucv_DEFAULT.dat as a generalistic/default definition
__GeneralRemarks__	general remarks upon the syntax within UCV files
__overview_of_syntax_elements__	shows all possible syntax in USER_common_variables
__Parameters__	CV-parameters that can also be set in UCV
AbaqusInterpolation	abaqus mesh interpolation
ACTIVE	sets active flags for boundary aliases
ALIAS	alias definitions within a begin_alias-end_alias-block
BoundaryConditions	definition of physical boundary conditions for boundary elements
BoundaryElements	definition of the boundary elements to be used during simulation
BUBBLES	BUBBLES
CODI	solve additional COnvection-DIffusion-problems (CODI)
ConsistencyChecksAtStartup	check the physical/mathematical consistency for user-given input data
COUPLING	couple the running MESHFREE simulation to another, currently running simulation
Curves	define curves in the input file
DropletSource	generate a sequence of spherical droplets
Equations	define functions, equations, and algebraic expressions
EVENT	events defined for the point cloud
include_Ucv{	include a file in UCV-format
INITDATA	prescribe initial data conditions
INTEGRATION	integration of the simulation results
KindOfProblem	Solver Selection for a simulation chamber
Loops	loop over a block of lines in the input file
MEMORIZE	memorize functionality
MONITORPOINTS	monitor points due to user-defined conditions
MOVE	move parts of the boundary by an explicit statement
NumericalControl	numerical control options
ODE	solver for ordinary differential equations (ODE)
PhysicalProperties	define physical properties of a material
PointCloudQualityCheck	check the quality of a read in point cloud
PointCloudReduction	select/mark MESHFREE points by reducing the point cloud

ReadInPointCloud	read in an already existing point cloud from file
RepeatCurrentTimeStep	repeat the current time step with different parameters or reduced pointcloud
RESTART	control the restart functionality
SAVE	save computational results in different formats
Selection	Switch/Case-type selection statement
SmoothingLength	define the smoothing length by a set of commands
TimeControl	time control options

3.1.1. ACTIVE

sets active flags for boundary aliases

The ACTIVE statement specifies when boundary elements are considered. It needs to be specified for the initial filling phase (%ACTIVE_init% , %ACTIVE_noinit% , %ACTIVE_nofill%) and the actual simulation (%ACTIVE_always%). The specifier for the filling phase is mandatory, whereas there is only %ACTIVE_always% for the simulation phase. Leaving off %ACTIVE_always% will deactivate the boundary during simulation.

The ACTIVE statements are then later referenced in the AliasForGeometryItems .

Common combinations:

```
ACTIVE ( $init_always$ ) = ( %ACTIVE_init% , %ACTIVE_always% ) # normal wall, inflow, or outflow
ACTIVE ( $noinit_always$ ) = ( %ACTIVE_noinit% , %ACTIVE_always% ) # wall if simulation starts at a nozzle
ACTIVE ( $nofill_always$ ) = ( %ACTIVE_nofill% , %ACTIVE_always% ) # complex geometry which would try too many seeding points in filling
ACTIVE ( $init_never$ ) = ( %ACTIVE_init% ) # free surface: fills initially, but is not kept for simulation
```

The behavior of %ACTIVE_init% can be different when using ORGANIZE_ReducedFillingOfWalls .

Intervals of activity:
If the boundary has to be switched on/off after certain times, or if there are activity intervals, the keyword %ACTIVE_always%
has to be replaced by PAIRS of numbers.

```
ACTIVE ( $init_temporal$ ) = ( %ACTIVE_init% , t_on_1, t_off_1 ) # boundary is active for all times t fulfilling (t_on_1 <= t <= t_off_1)
ACTIVE ( $noinit_temporal2$ ) = ( %ACTIVE_noinit% , t_on_1, t_off_1, t_on_2, t_off_2 ) # boundary is active for all times t fulfilling (t_on_1 <= t <= t_off_1) OR (t_on_2 <= t <= t_off_2)
ACTIVE ( $noinit_temporalN$ ) = ( %ACTIVE_noinit% , t_on_1, t_off_1, ..., t_on_N, t_off_N ) # N time intervals.
```

- The intervals have to be given in increasing order.
- There is no limit to the number of time intervals.

Good to know:

- See also BC_PASSON , IDENT_PASSON , MOVE_PASSON .
- In order to detect free surfaces, the parameter compute_FS must be set to 'YES' for the corresponding chamber.

List of members:

%ACTIVE_init%	active during initial filling
%ACTIVE_nofill%	only visible during initial filling
%ACTIVE_noinit%	not active during initial filling
%ACTIVE_always%	active during simulation

MESHFREE · InputFiles · USER_common_variables · ACTIVE · %ACTIVE_always%

%ACTIVE_always%

active during simulation

%ACTIVE_always% determines if the boundary is active during the actual simulation. It does not imply that the boundary will be active in the initial filling phase. In order to leave out a boundary in the simulation just leave off the %ACTIVE_always% keyword.

MESHFREE · InputFiles · USER_common_variables · ACTIVE · %ACTIVE_init%

%ACTIVE_init%

active during initial filling

Activates the boundary during initial filling. MESHFREE starts by filling points on these boundaries first and then filling into the interior.

The actual filling behavior depends on IDENT and ORGANIZE_ReducedFillingOfWalls as well.

For boundary elements to be visible in the filling phase but not to fill points themselves choose %ACTIVE_nofill% instead. This can be helpful if there is complex geometry, e.g. a fully detailed car. However, one filling boundary part is mandatory.

MESHFREE · InputFiles · USER_common_variables · ACTIVE · %ACTIVE_nofill%

%ACTIVE_nofill%

only visible during initial filling

Boundary parts marked with %ACTIVE_nofill% are visible during the initial filling phase. However, they do not fill points to the inside, but only restrict the filling domain.

MESHFREE · InputFiles · USER_common_variables · ACTIVE · %ACTIVE_noinit%

%ACTIVE_noinit%

not active during initial filling

Boundary parts marked with %ACTIVE_noinit% are not visible in the initial filling phase.

MESHFREE · InputFiles · USER_common_variables · ALIAS

3.1.2. ALIAS

alias definitions within a begin_alias-end_alias-block

Note: All names of geometry parts need to be assigned to special aliases (see AliasForGeometryItems).

An alias block contains replacement definitions, i.e. what a certain string occurring in USER_common_variables will be replaced with.

```
begin_alias{ }
"alias1" = " String to replace &alias1& "
...
"aliasN" = " String to replace &aliasN& "
end_alias
```

I f MESHFREE encounters one of the text strings given in the alias block on the left hand side during read-in of USER_common_variables ,
then they will by replaced by the string given on the right hand side.

In order to exclude misinterpretations, text strings to be replaced have to be put in between &...& icons.

Note:

- Definition and referencing of vectorial aliases is also possible. The entries have to be of the same type, i.e. string or number.
- Referencing aliases on the left hand side of another alias definition is also possible.
- See also ConstructClause .
- See also Variables .

Example 1:

```
begin_alias{ }
"EqunForBC" = "[ Y%ind_x(1)%/Y%ind_h% * &Param& ]"
"Param" = "23.452444 * &Scaling& "
"Scaling" = "0.001 "
end_alias
```

During read-in of USER_common_variables the line

```
BC_T ( $TemCond$ ) = &EqunForBC&
```

will be replaced by: BC_T ($TemCond$) = [Y %ind_x(1)% /Y %ind_h% * 23.452444 * 0.001]

Example 2:

```
begin_alias{ }
"Class" = "inflow, wall, outflow" # definition of geometry class
...
"&Class(1)&" = " BC$BC_in$ ..." # definition of inflow alias
"&Class(2)&" = " BC$BC_wall$ ..." # definition of wall alias
"&Class(3)&" = " BC$BC_out$ ..." # definition of outflow alias
end_alias
```

During read-in of USER_common_variables &Class(i)& will be replaced by the respective entry of Class.

List of members:

AliasForGeometryItems	alias definitions for geometry parts

AliasForGeometryItems

alias definitions for geometry parts

All names of geometry parts need to be assigned to special aliases.

There are multiple options to deal with the name of a geometry part:

1.) Explicit assignment of properties to full names of geometry parts

- Declare full properties as explained below.

  Example:

```
begin_alias{ }
"car" = " BC$BC_box$ ACTIVE$noinit_always$ IDENT%BND_slip% MAT$Mat1$
TOUCH%TOUCH_liquid% MOVE$MOVE_car$ LAYER0 CHAMBER1 " # full description
end_alias
```

- Assign properties of another alias.

  Example:

```
begin_alias{ }
"windshield" = "&car&" # reference to alias car
end_alias
```

2.) Automatical choice of properties based on patterns

- Similar to 1, but matching multiple names with a wildcard. The wildcard-option is tried ONLY, if no direct match with the given aliasses
  can be stablished. In this case, each of the matched names is available in postprocessing.

  Example:
  Names in geometry file: WheelFrontLeft, WheelFrontRight, WheelBackLeft, WheelBackRight

```
begin_alias{ }
"Wheel*" = "&car&" # reference to alias car
"*" = " BC$BC_box$ ACTIVE$noinit_always$ IDENT%BND_slip% MAT$Mat1$ TOUCH%TOUCH_liquid%
MOVE$NO_MOVE$ LAYER0 CHAMBER1 " # full description
end_alias
```

- Using the DEFAULT properties for any group of names not specified via the options above. The alias-name-definition has to
  contain "_DEFAULT" at the end. See also __DEFAULT_configuration_file__ .
  Example:

```
begin_alias{ }
"in*_DEFAULT" = "&car&" # reference to previously defined alias car
end_alias
```

  For all those matching a DEFAULT-item, MESHFREE attaches "_DEFAULT" as a suffix
  to the given name from the geometry input, such that it can be recognized easily as DEFAULT-defined. The geomtry item "inflow1", matching the
  alias "in*_DEFAULT", will be named "inflow1_DEFAULT" for postprocessing.

- DEPRECATED: Using default properties for any names not specified via the options above. All these names will be replaced by 'default'.

Example:

```
begin_alias{ }
"default" = "&car&" # reference to alias car
end_alias
```

Instead of "default", use "*_DEFAULT" in order to have a general default definition, that matches ALL geoemtry.

List of members:

ACTIVE	(required) define the activation behavior of the boundary elements of this part
BC	(required) define flag for boundary conditions
BC_PASSON	(optional) for deactivated/disappearing boundary elements: give BC-flag to released MESHFREE points
BOUNDARYFILLING	(optional) possibility to request reduced filling behavior for MESHFREE points for parts of the boundary
CHAMBER	(required) define the chamber index for the geometry entities
COORDTRANS	(experimental) define coordinate transformation to mathematically transform long thin geometries into short thick ones
IDENT	(required) how to handle the geometry part during point cloud organization
IDENT_PASSON	(optional) for deactivated/disappearing boundary elements: give IDENT-information to released MESHFREE points
IGNORE	(optional) ignore this geometry item when reading from geometry file
LAYER	(optional) define layer index
MAT	(required) define the material flag to be used, when the geometry part fills new points (mostly for initial filling)
METAPLANE	(optional) define a cutting plane for MESHFREE points
MOVE	(required) provide a flag for the definition of boundary movement
MOVE_PASSON	(optional) for deactivated/disappearing boundary elements: give MOVE-flag to released MESHFREE points
MPCCI	(optional) define mpcci index
POSTPROCESS	(optional) define flag for postprocessing/integration
REV_ORIENT	(optional) flip around orientation of boundary parts upon read-in of geometry files
SMOOTH_LENGTH	(optional) define flag for smoothing length definition
SMOOTH_N	(experimental) invoke smoothing of the boundary
SYMMETRYFACE	(optional) definition of the geometry part as symmetryface (influences distance computation)
TOUCH	(required) define the wetting/activation behavior of MESHFREE points along the given boundary part
TWOSIDED	(experimental) copy the boundary entity re-orient it, and give other attributes to it

ACTIVE

(required) define the activation behavior of the boundary elements of this part

The ACTIVE flag in the alias definition of a boundary element references an ACTIVE statement. The ACTIVE statement defines whether a boundary is active during the initial filling and/or the remaining simulation. Additionally, it specifies if the boundary fills points to the inside in the initialization phase of the simulation.

Example:

```
ACTIVE ( $init_always$ ) = ( %ACTIVE_init% , %ACTIVE_always% ) # Definition of the Active statement
...
begin_alias{ }
"car" = " BC$...$ ACTIVE$init_always$ IDENT%...% MAT$...$ TOUCH%...% MOVE$...$ CHAMBER1 " # Referencing
the Active statement
end_alias
```

MESHFREE · InputFiles · USER_common_variables · ALIAS · AliasForGeometryItems · BC

BC

(required) define flag for boundary conditions

In the alias setion, the BC flag attaches boundary conditions to boundary elements. Several AliasForGeometryItems might share the same boundary conditions.

Example:

```
# definition of boundary conditions $BC_wall$ for velocity, pressure and dynamic pressure
BC_p ( $BC_wall$ ) = ( %BND_wall% )
BC_v ( $BC_wall$ ) = ( %BND_wall% )
BCON ( $BC_wall$ ,%ind_p_dyn%) = ( %BND_wall% )
...
begin_alias{ }
#referencing the definition of the boundary conditions $BC_wall$
"wall" = " BC$BC_wall$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE$...$ CHAMBER1 "
end_alias
```

Referencing the definition of the boundary conditions BC_wall in the alias section applies the boundary conditions to all boundary elements with that alias.

See also: BoundaryConditions

MESHFREE · InputFiles · USER_common_variables · ALIAS · AliasForGeometryItems · BOUNDARYFILLING

BOUNDARYFILLING

(optional) possibility to request reduced filling behavior for MESHFREE points for parts of the boundary

Experimental!
Reduce filling on certain boundary elements.

List of members:

BOUNDARYFILLING_OnlyInActiveNeighborhood	only if active points in the neighborhood
BOUNDARYFILLING_OnlyIfActiveItself	only if BE is active
BOUNDARYFILLING_Always	always fill

MESHFREE · InputFiles · USER_common_variables · ALIAS · AliasForGeometryItems · BOUNDARYFILLING · BOUNDARYFILLING_Always

BOUNDARYFILLING_Always

always fill

Default behavior.

CHAMBER

(required) define the chamber index for the geometry entities

In MESHFREE , a simulation chamber generally means a phase that takes part in the simulation. In the alias section, the CHAMBER index selects for which KindOfProblem the boundary has an influence and provides thus a link between the solver choice in KOP and the boundary conditions BC in the alias definition.

Example 1: In a one-phase example KOP is defined for **CHAMBER 1**

```
KOP(1) = LIQUID LAGRANGE IMPLICIT v-- TURBULENCE:k-epsilon
...
begin_alias{ }
"car" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE$...$ CHAMBER1 "
end_alias
```

and in the alias section the flag **CHAMBER1** links to **KOP(1)** for the boundary element "car" .

If the same boundary needs to be visible to several chambers, they need to be defined once for each chamber, possibly with different settings.

Example 2: in this two-phase example, KOP selects solvers for both simulation chambers 1 and 2.

```
KOP(1) = LIQUID IMPLICIT LAGRANGE vp- T:NONE # chamber 1: air phase
KOP(2) = LIQUID IMPLICIT LAGRANGE vp- T:NONE # chamber 2: water phase
...
begin_alias{ }
"wall" = " BC$BC_wall_air$ ACTIVE$init_always$ IDENT%BND_slip% MAT$AIR$ TOUCH%TOUCH_always%
MOVE$NO_MOVE$ CHAMBER1 "
"wall" = " BC$BC_wall_water$ ACTIVE$noinit_always$ IDENT%BND_wall% MAT$WATER$ TOUCH%TOUCH_liquid%
MOVE$NO_MOVE$ CHAMBER2 "
end_alias
```

The boundary elements with the alias "wall" are used in both chambers as there is one line for CHAMBER1 and one line for CHAMBER2. In most cases it makes sense that both chambers share the same MOVE statement as the movement of the boundary elements will be identical. Everything else might be set different. Of course, it depends on the use case if the geometry should be visible for both chambers.

A further example with different geometries for both phases can be found here .

COORDTRANS

(experimental) define coordinate transformation to mathematically transform long thin geometries into short thick ones

EXPERIMENTAL only.

List of members:

COORDTRANS_linear

COORDTRANS_radial

COORDTRANS_spherical

COORDTRANS_ring

COORDTRANS_cone

MESHFREE · InputFiles · USER_common_variables · ALIAS · AliasForGeometryItems · COORDTRANS · COORDTRANS_cone

COORDTRANS_cone

EXPERIMENTAL only.

MESHFREE · InputFiles · USER_common_variables · ALIAS · AliasForGeometryItems · COORDTRANS · COORDTRANS_linear

COORDTRANS_linear

EXPERIMENTAL only.

MESHFREE · InputFiles · USER_common_variables · ALIAS · AliasForGeometryItems · COORDTRANS · COORDTRANS_radial

COORDTRANS_radial

EXPERIMENTAL only.

MESHFREE · InputFiles · USER_common_variables · ALIAS · AliasForGeometryItems · COORDTRANS · COORDTRANS_ring

COORDTRANS_ring

EXPERIMENTAL only.

MESHFREE · InputFiles · USER_common_variables · ALIAS · AliasForGeometryItems · COORDTRANS · COORDTRANS_spherical

COORDTRANS_spherical

EXPERIMENTAL only.

MESHFREE · InputFiles · USER_common_variables · ALIAS · AliasForGeometryItems · IDENT

IDENT

(required) how to handle the geometry part during point cloud organization

In the alias section, the IDENT identifier defines how boundary elements are treated during point cloud organization and in distance computation. IDENT is used in AliasForGeometryItems statements.

Example:

```
begin_alias{ }
"car" = " BC$...$ ACTIVE$...$ IDENT%BND_wall% MAT$...$ TOUCH%...% MOVE$...$ LAYER0 CHAMBER1 "
end_alias
```

The most important identifiers for regular geometries are defined as:

```
IDENT %BND_wall%
IDENT %BND_slip%
IDENT %BND_wall_NoLayerThickness%
IDENT %BND_free%
IDENT %BND_inflow%
IDENT %BND_outflow%
```

IDENT %BND_free% is used in initial filling to define the initial free surface. IDENT %BND_wall% will be default if nothing is set, except for boundary elements of type BND_plane which will be %BND_free% by default.

A second set of identifiers is provided for invisible boundary elements in integration statements:

```
IDENT %BND_void%
IDENT %BND_BlindAndEmpty%
```

IDENT %BND_BlindAndEmpty% is perfect for flux integrations, e.g. %INTEGRATION_FLUX% , and monitor point creation with %MONITORPOINTS_CREATION_PenetrationOfBlindAndEmptyBoundary% .

A list of all possible IDENT identifiers is found below.

Note: The type of boundary associated to a MESHFREE point is found in %ind_kob% .

List of members:

%BND_void%	invisible precision measurement BE
%BND_BlindAndEmpty%	invisible flux measurement BE
%BND_wall%	non-moving wall points
%BND_slip%	movable wall points
%BND_inflow%	inflow BE
%BND_outflow%	outflow BE
%BND_free%	free surface BE
%BND_wall_NoLayerThickness%	non-moving wall points
%BND_cut%	cut-off points at metaplanes

%BND_BlindAndEmpty%

invisible flux measurement BE

Like %BND_void% , %BND_BlindAndEmpty% does not participate in computations of the distance of points to the boundary. It also does not have any boundary points on it.

The main use is in flux integrations using e.g. %INTEGRATION_FLUX% , %INTEGRATION_ABSFLUX% , or %INTEGRATION_FLUX_DROPLETPHASE% .
Similarly, it is used in the cross() -function.
Furthermore, monitor points can be created at the intersection with this boundary using the %MONITORPOINTS_CREATION_PenetrationOfBlindAndEmptyBoundary% flag.

%BND_cut%

cut-off points at metaplanes

%BND_cut% is used to cut off points crossing this boundary. Most commonly it is used with metaplanes (see BND_plane). Other than this it does not participate in any computations.

%BND_free%

free surface BE

%BND_free% is usually used as IDENT flag in the initial filling phase to specify the initial free surface. Points filled on this boundary will mark all its boundary points as free surface. The ACTIVE flag should be set to be only active during the initial filling phase.

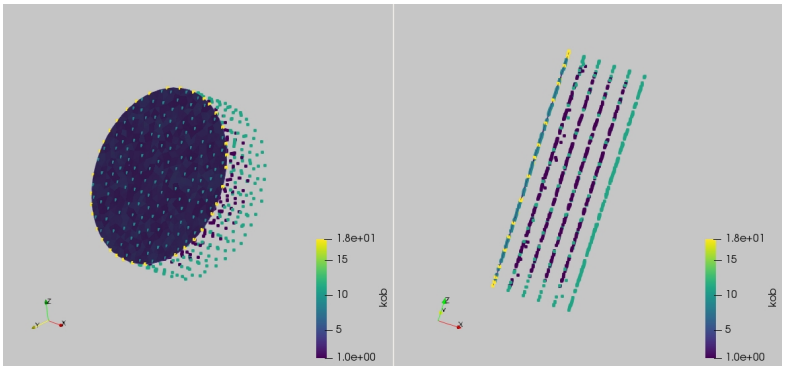
%BND_free% does not participate in point cloud organization if ORGANIZE_ReducedFillingOfWalls is turned on.

%BND_inflow%

inflow BE

%BND_inflow% is an identifier for a boundary geometry with a special filling algorithm. It does not trigger filling to the entire interior domain. Instead, it will fill several additional layers in front of the inflow (in normal direction). This will lead to stability in case of a free (no connection to other boundary elements) inlet.

Example: The following picture shows the initial filling for a free (no connection to other boundary elements), round inlet with IDENT %BND_inflow% .



Parameter COMP_FillEdges = 1 improves the quality of the pointcloud by placing points on the edge of the inflow shape (visible with %ind_kob% = 18) and parameter Nb_InflowLayers = 5 sets the number of initially filled layers.

%BND_inflow% is also filled when ORGANIZE_ReducedFillingOfWalls is turned on.

%BND_outflow%

outflow BE

%BND_outflow% is a special kind of boundary geometry. It is specifically useful for outflow but also some inflow boundaries. If the inflow is adjacent to an entirely filled interior domain it can have the %BND_outflow% flag. Compare for differences to %BND_inflow% . Because of typical outflow boundary conditions boundary points on %BND_outflow% are not fixed on the boundary, but are able to move with the flow velocity.

%BND_outflow% does not participate in point cloud organization if ORGANIZE_ReducedFillingOfWalls is turned on.

MESHFREE · InputFiles · USER_common_variables · ALIAS · AliasForGeometryItems · IDENT · %BND_slip%

%BND_slip%

movable wall points

%BND_slip% in an IDENT statement is used for walls with slip velocity boundary conditions. It will fill points according to the ACTIVE statement of the same AliasForGeometryItems . Compared to %BND_wall% points are marked to be movable and hence will be moved with the according velocity.

%BND_slip% is filled when ORGANIZE_ReducedFillingOfWalls is turned on.

MESHFREE · InputFiles · USER_common_variables · ALIAS · AliasForGeometryItems · IDENT · %BND_void%

%BND_void%

invisible precision measurement BE

%BND_void% is usually not visible when computing the distance of a numerical point to the boundary. In contrast to %BND_BlindAndEmpty% , %BND_void% will however participate in filling. This is useful for measurements of properties on this boundary element. Because of this points will be densly filled on boundary elements marked with %BND_void% .

%BND_void% will still be filled if ORGANIZE_ReducedFillingOfWalls is turned on.

MESHFREE · InputFiles · USER_common_variables · ALIAS · AliasForGeometryItems · IDENT · %BND_wall%

%BND_wall%

non-moving wall points

%BND_wall% in an IDENT statement is treated as regular wall. This means that it will fill points based on ACTIVE . In contrast to %BND_slip% boundary points are marked as non-moving.

%BND_wall% is filled when ORGANIZE_ReducedFillingOfWalls is turned on.

MESHFREE · InputFiles · USER_common_variables · ALIAS · AliasForGeometryItems · IDENT · %BND_wall_NoLayerThickness%

%BND_wall_NoLayerThickness%

non-moving wall points

Behaves mostly identical to %BND_wall% . However, dist_LayerThickness does not have an effect on free surface points close to boundaries marked with %BND_wall_NoLayerThickness% .

IGNORE

(optional) ignore this geometry item when reading from geometry file

In the alias definition, if a boundary element is tagged with the IGNORE flag this boundary item will be ignored when reading from the geometry file. This is useful when there are parts in the geometry, that shall not take part in the simulation. Instead of removing them from the geometry file, they can be ignored by name - also by using wildcards.

Example 1:

```
begin_alias{ }
"wheel" = " IGNORE " # ignore all boundary elements "wheel"
end_alias {}
```

Example 2: Using wildcards:

```
begin_alias{ }
"wheel*" = " IGNORE " # ignore all parts which names start with 'wheel'
end_alias {}
```

LAYER

(optional) define layer index

The LAYER functionality offers a method to filter neighbors in the stencils that would otherwise be considered through thin geometries. This prevents influencing through thin layers of geometries, e.g.in stirring applications.

A newer algorithm for performing the stencil filtering is steered by NEIGHBOR_FilterMethod . Please consider using these methods before utilizing the LAYER functionality.

Layer based neighbor filtering

Different LAYER numbers tell MESHFREE to treat points with different numbers to not be visible to each other. Neighbor points with a different LAYER number are not taken into account for the points stencil. This helps with certain kinds of problems related to small and/or thin boundary.

By default all LAYERS have the index 0 - thus all neighbors would be visible to each other.

Example 1: In this example points facing one part of the geometry should not be considered neighbors of the other part of the geometry despite being in the h-environment.

```
begin_alias{ }
"OnePartOfThinGeometry" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE$...$ LAYER1
CHAMBER1 "
"OppositePartOfThinGeometry" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE$...$ LAYER2
CHAMBER1 "
end_alias
```

Advanced Layer based neighbor filtering

The default value of compute_LAYER is 0. !$FPMDOCU To enable the advanced mode of the layer based filtering compute_LAYER can be set to a positive integer. Points can only be neighbors if the LAYER numbers of two points differ by less or equal than compute_LAYER .

Example 2: In the common_variables file set

```
compute_LAYER = 2
```

In the USER_common_variables the LAYER keyword is attached to the aliasses of different geometry parts.

```
begin_alias{ }
"OnePartOfThinGeometry" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE$...$ LAYER1
CHAMBER1 "
"OppositePartOfThinGeometry" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE$...$ LAYER4
CHAMBER1 "
"AnotherPartOfThinGeometry" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE$...$ LAYER3
CHAMBER1 "
end_alias
```

Points with LAYER1 and LAYER4 can't be neighbors of each other, but they can both be neighbor to a point with LAYER3.

MAT

(required) define the material flag to be used, when the geometry part fills new points (mostly for initial filling)

The MAT tag in the alias definition associates the boundary elements with the PhysicalProperties of a given material. These PhysicalProperties will be inherited to the points filled by this boundary.

Example:

```
# Definition of Physical properties for material $MAT1$
density( $MAT1$ ) = 2500.0 # density in kg/(m³)
cv( $MAT1$ ) = 1500.0 # heat capacity in Nm/()Kg*K))
lambda( $MAT1$ ) = 2.0 # heat conductivity in W/(mK))
eta( $MAT1$ ) = 1.0e6 # viscosity in Pa*s
mue( $MAT1$ ) = 0.0 # shear modulus Pa
sigma( $MAT1$ ) = 0.3 # surface tension in N/m
heatsource( $MAT1$ ) = 0 # heat source W/(m³)
gravity( $MAT1$ ) = (0.0, 0.0, 0.0) # gravity in m/s²
...
begin_alias{ }
"car" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$MAT1$ TOUCH%...% MOVE$...$ CHAMBER1 " #referencing the
physical properties.
end_alias
```

The use of **MAT$MAT1$** in the alias definition establishes the link between material and the boundary element "car" .

METAPLANE

(optional) define a cutting plane for MESHFREE points

Points outside the METAPLANE will be cut off. IDENT should be preferrably set to %BND_cut% .

The METAPLANE flag takes a number as parameter. METAPLANES with the same number only reject/delete points if it is outside all METAPLANES with the same number. The number has to be >=1.

Example:

```
begin_alias{ }
"plane" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE$...$ LAYER0 CHAMBER1 METAPLANE1
"
end_alias
```

See BND_plane for additional information.

Visualization of a METAPLANE can be turned on for ENSIGHT6 with the additional flag 'P'.

MOVE

(required) provide a flag for the definition of boundary movement

In the alias section, the MOVE flag selects a MOVE statement for the boundary elements.

Example 1 : defining a MOVE statement and referencing it in the alias section

```
MOVE ( $MOVE_in_x_direction$ ) = ( %MOVE_velocity% , 1.0, 0.0, 0.0 ) #definition of MOVE
...
begin_alias{ }
"wall" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE$MOVE_in_x_direction$ CHAMBER1 " #
referencing the definition of the MOVE statement
end_alias
```

The corresponding MOVE statement is selected through the variable name **$MOVE_in_x_direction$** .

If no movement of geometry is involved in the simulation model, there is also the shorthand writing MOVE -1 for this without having to define a MOVE statement first.

Example 2 : no movement by **MOVE-1**

```
begin_alias{ }
"wall" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE-1 CHAMBER1 "
end_alias
```

MPCCI

(optional) define mpcci index

The MPCCI numbers tell MESHFREE to couple this geometry part with the MpCCI interface.

By default all MPCCIs have the index -1, which means no coupling with MpCCI is done.

Example:

```
begin_alias{ }
"car" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE$...$ MPCCI1 CHAMBER1 "
end_alias
```

POSTPROCESS

(optional) define flag for postprocessing/integration

POSTPROCESS defines a name that can be used in integrations over the boundary, e.g. for %INTEGRATION_BND% . The postprocessing tag associates for an INTEGRATION statement to which boundary it belongs.

This flag is optional and needs only be supplied if the boundary should be used by an integration statement.

Example:

```
INTEGRATION ( $INTpressure$ ) = ( %INTEGRATION_BND% , [Y %ind_p% +Y %ind_p_dyn% ], [Y%ind_p+Y
%ind_p_dyn% ], [Y %ind_p% +Y %ind_p_dyn% ], $PPwall$ , %INTEGRATION_Header%, "pressure" )
...
begin_alias{ }
"car" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE$...$ LAYER0 CHAMBER1
POSTPROCESS$PPwall$ "
end_alias
```

REV_ORIENT

(optional) flip around orientation of boundary parts upon read-in of geometry files

Invert the orientation of all boundary elements of this alias.

Example :

```
begin_alias{ }
"WronglyOrientedPart" = "REV_ORIENT BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE$...$
CHAMBER1 "
end_alias
```

An very similar functionality is provided in revOrient{ }.

SMOOTH_LENGTH

(optional) define flag for smoothing length definition

In the alias section, the SMOOTH_LENGTH flag references a smoothing length for the boundary element it is attached to.

Prerequisite : this method of assigning and defining the smoothing length only works if the UCV parameter USER_h_funct is either set to

- DSCR or
- ADDS ,

else the statements do not have an effect.

Depending on the type of boundary element, the behaviour is different:

- If SMOOTH_LENGTH is attached to a point BND_point , then the condition defined in the SMOOTH_LENGTH will be evaluated with respect to that point.
- If SMOOTH_LENGTH is used on triangulated boundaries (BND_tria), MESHFREE will sample several positions on this boundary element and the condition defined in SMOOTH LENGTH will be defined with respect to these positions.

The smoothing length applies to all points within the CHAMBER . If multiple smoothing lengths per chamber are defined and attached, then for each point in the chamber all smoothing lengths are evaluated and the final smoothing length is the minimum over all smoothing lengths.

This functionality can for example be used to refine locally around boundary elements.

Warning : Applying a smoothing length to a large geometry is computational very expensive and thus will significantly slow down MESHFREE in its pointcloud organization step. So it is good practice to avoid assigning a SMOOTH_LENGTH to large boundary elements.

Example 1: Constant smoothing length attached to a point

```
USER_h_funct = 'DSCR'
USER_h_min = "0.1"
USER_h_max = "2.0"
SMOOTH_LENGTH ( $SL1$ ) = ( %H_constant% , 0.1 ) #definition of a (constant) smoothing length $SL1$
...
begin_boundary_elements{ }
BND_point &dummyPointSmooth& 0 0 0 #defines a point in the origin (0,0,0)
end_boundary_elements {}
...
begin_alias{ }
"dummyPointSmooth" = " SMOOTH_LENGTH$SL1$ CHAMBER1 " # establishes the link between the smoothing length
$SL1$ and the chamber.
end_alias
```

Example 2: Local spherical refinement around boundary alias "RefineAroundThisBE"

```
USER_h_funct = 'DSCR'
USER_h_min = "0.1"
USER_h_max = "2.0"
SMOOTH_LENGTH ( $SL2$ ) = ( %H_spherical% , 0.1, 0.5, 0.1, 1.0 ) #definition of a (spherical refined) smoothing
length $SL2$
...
begin_alias{ }
"RefineAroundThisBE" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE$...$
SMOOTH_LENGTH$SL2$ CHAMBER1 " # attach smoothing length $SL2$ to boundary element
end_alias
```

See SmoothingLength for more information.

SMOOTH_N

(experimental) invoke smoothing of the boundary

EXPERIMENTAL only.

- Each node point i establishes its local boundary normal by

$$\tilde{\boldsymbol{n}}_i = \frac{\sum_{k=\mathrm{AllTrianglesAttachedToPoint}} (\boldsymbol{p}_{k,2} - \boldsymbol{p}_{k,1}) \times (\boldsymbol{p}_{k,3} - \boldsymbol{p}_{k,1})}{\|...\|_2}$$

  where $\boldsymbol{p}_{k,i}, i = 1...N_p$ are the node point coordinates of the shape (in most cases triangles N_p=3, sometimes quads, N_p=4)
- The boundary normal of the MESHFREE point with index i which is situated inside of the triangle with index k is computed by its shape functions, i.e.

$$\boldsymbol{n}_i = \frac{\sum_{j=1...N_p} s_j \cdot \tilde{\boldsymbol{n}}_{k,j}}{\|...\|_2}$$

  where N_p is the number of nodes of the given boundary element.
- The shape functions are computed for each MESHFREE point in a standard way. The MESHFREE point with index i situated on the triangle with index k has the shape functions

$$\boldsymbol{x}_i = \sum_{j=1\ldots N_p} s_j \cdot \boldsymbol{p}_{k,j}$$

with the requirement $\sum_{j=1\ldots N_p} s_j = 1$

SYMMETRYFACE

(optional) definition of the geometry part as symmetryface (influences distance computation)

MESHFREE computes distances for points of the pointcloud to all SYMMETRYFACEs in proximity. The point is only considered to be inside if this it is inside regarding all different SYMMETRYFACEs.

Example:

```
begin_alias{ }
"car" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE$...$ CHAMBER1 SYMMETRYFACE1"
"box" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE$...$ CHAMBER1 SYMMETRYFACE2"
end_alias
```

TOUCH

(required) define the wetting/activation behavior of MESHFREE points along the given boundary part

The TOUCH flag defines when to activate boundary points on these boundary elements.

TOUCH is used in AliasForGeometryItems statements

```
begin_alias{ }
"car" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%TOUCH_liquid% MOVE$...$ LAYER0 CHAMBER1 "
end_alias
```

There are two flags for general activation/deactivation and three flags for wetting:

```
TOUCH %TOUCH_always%
TOUCH %TOUCH_never%
TOUCH %TOUCH_liquid%
TOUCH %TOUCH_solid%
TOUCH %TOUCH_geometrical%
```

%TOUCH_liquid% and %TOUCH_solid% will behave quite similarly. The only difference is in detachment of points from the boundary. %TOUCH_solid% has an additional criterion how this might occur. The difference to %TOUCH_geometrical% is in the initial filling: Both for %TOUCH_liquid% and %TOUCH_solid% interior points very close to the boundary will be projected back to the boundary element in the initial filling phase. This does not occur for %TOUCH_geometrical% .

The default value if no TOUCH flag is provided is %TOUCH_geometrical% .

Activation can be further controlled with ORGANIZE_ForceTouchCheckAtWalls .

There is one special flag for reflective boundaries:

```
TOUCH %TOUCH_reflection%
```

Reflected points will set index %ind_Organize% in Y to %ORGANIZE_WasPushedBackFromBoundary% .

List of members:

%TOUCH_always%	boundary points always active
%TOUCH_never%	boundary points never active
%TOUCH_liquid%	boundary points activated by flow (non-geometrical criterion)
%TOUCH_solid%	boundary points activated by flow (non-geometrical criterion plus special tear off)
%TOUCH_geometrical%	boundary points activated by flow (geometrical criterion)
%TOUCH_reflection%	points reflected at boundary

%TOUCH_always%

boundary points always active

Boundary points on boundary elements marked with %TOUCH_always% will always be active.

%TOUCH_geometrical%

boundary points activated by flow (geometrical criterion)

Simplest form of activating boundary points by the flow. Based on ORGANIZE_ForceTouchCheckAtWalls either only free surface points or both free surface points and interior points will activate boundary points if they are in their proximity.

%TOUCH_liquid%

boundary points activated by flow (non-geometrical criterion)

Boundary points on these boundary elements are activated by free surface and interior points (controlled by ORGANIZE_ForceTouchCheckAtWalls).

In the initial filling phase interior points very close to the boundary will be projected to the boundary. To avoid this use %TOUCH_geometrical% instead. For additional tear-off criteria use %TOUCH_solid% .

%TOUCH_never%

boundary points never active

Boundary points on boundary elements marked with %TOUCH_never% will never be activated.

TOUCH · %TOUCH_reflection%

%TOUCH_reflection%

points reflected at boundary

This will reflect oncoming interior or free surface points from the boundary according to the local boundary normal. The %TOUCH_reflection% flag can also be set for free surfaces.

Points which have been reflected set their index %ind_Organize% in Y to %ORGANIZE_WasPushedBackFromBoundary% .

MESHFREE · InputFiles · USER_common_variables · ALIAS · AliasForGeometryItems · TOUCH · %TOUCH_solid%

%TOUCH_solid%

boundary points activated by flow (non-geometrical criterion plus special tear off)

Same activation behavior as %TOUCH_liquid% . Also here in the initial filling phase interior points close to the boundary are projected onto the boundary elements.

The difference to %TOUCH_liquid% is an additional tear-off criterion. Free surface points will also tear off if

1.) tension forces are pulling the point, and
2.) the velocity in normal direction is non-zero, and
3.) the point was on the boundary for at least one time step.

Additionally, a user-defined tear-off criterion can be specified using BC_TearOffCriterion . A point will tear off if either the list or the user-defined criterion is fulfilled.

MESHFREE · InputFiles · USER_common_variables · ALIAS · AliasForGeometryItems · TWOSIDED

TWOSIDED

(experimental) copy the boundary entity re-orient it, and give other attributes to it

EXPERIMENTAL only.
Example: The alias "box" refers to a boundary element for CHAMBER1 and at the same time re-orientated to a boundary element for CHAMBER2

```
begin_alias{ }
"box" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE$...$ CHAMBER1 TWOSIDED
BC$...$ ACTIVE$...$ IDENT%...% MAT$...$ TOUCH%...% MOVE$...$ CHAMBER2 "
end_alias
```

MESHFREE · InputFiles · USER_common_variables · AbaqusInterpolation

3.1.3. AbaqusInterpolation

abaqus mesh interpolation

MESHFREE provides several ways of interpolation of pressure data to abaqus meshes.
Supported mesh elements are: STRI65, S8R, S3, S4

```
AbaqusInterpolation (1)=(%ABAQUS_IntplMidpoint%,3)
```

directly interpolates (Intpl) pressure data onto mesh element midpoints (MidPoint)

```
AbaqusInterpolation (1)=(%ABAQUS_AVMidpointShpdNode%,1)
```

is a two step mapping of data onto mesh element midpoints (MidPoint).
First, the pressure is interpolated on the mesh element nodes (Node) by
a weighted average based on the distance between MESHFREE nodes and Abaqus nodes (shepard interpolation, Shpd).
Then, the data is getting averaged (AV) and this value is set to be the value at the mesh element midpoint.

```
AbaqusInterpolation (1)=(%ABAQUS_AVMidpointIntplNode%,3)
```

does essentially the same as ABAQUS_AVMidpointShpdNode, but shepard interpolation is
replaced by a second order polynomial fpm interpolation (Intpl)
on the mesh nodes (Node)

```
AbaqusInterpolation (1)=(%ABAQUS_ShpdMidpoint%,1)
)
```

directly performs a shepard interpolation (Shpd) onto mesh midpoints (Midpoint)

```
AbaqusInterpolation (1)=(%ABAQUS_IntplNode%,3)
```

directly performs a second order polynomial fpm interpolation (Intpl) onto mesh nodes (Node)

3.1.4. BUBBLES

List of members:

BUBBLE_forbidden	let MESHFREE know, in what regions bubbles cannot be accepted

BUBBLE_forbidden

let MESHFREE know, in what regions bubbles cannot be accepted

```
BUBBLE_forbidden ($Material$) = ( MathematicalEquation )
```

MathematicalEquation : is a typical right hand side expression.
If MathematicalEquation is positive for at least ONE surface point of the bubble (active as well as inactive points), then it is rejected as a regular
bubble. i.e.

- its pressure is set to zero, i.e. Y%ind_pBubble%==0
- its volume is: Y %ind_volBubble% = -(trueBubbleVolume)

Example:

```
BUBBLE_forbidden ( $Material$ ) = ( [ -Y %ind_x(1)% - 1.0 ] ) # this expression becomes positive for x
# # with members x
```


3.1.5. BoundaryConditions

definition of physical boundary conditions for boundary elements

Boundary conditions are an essential ingredient for the simulation model. They are defined and then attached to boundary elements in the alias section. They must be provided according to the solver choice.

General Syntax

Boundary conditions for all relevant variables can be defined by:

```
BC_p ( $BCindex$ ) = RightHandSideExpression # hydrostatic pressure
BC_v ( $BCindex$ ) = RightHandSideExpression # velocity
BCON ( $BCindex$ ,%ind_Var%) = RightHandSideExpression # BCON is a more general keyword to define boundary
conditions
```

Here, BC_p and BC_v are specialized keywords for pressure and velocity, respectively, and BCON is a more general keyword to define boundary conditions for arbitrary variables.
These boundary conditions are then related to boundary elements in the alias section (see AliasForGeometryItems) with the BC -flag:

```
begin_alias{ }
"BoundaryName" = " ...MOVE$MOVEindex$ ... BC$BCindex$ ... SMOOTH_LENGTH$Hindex$ ... "
end_alias
```

List of members:

DROPLETPHASE__BC__	Boundary Conditions for Dropletphase
LIQUID__BC__	definition of physical boundary conditions for LIQUID solver

DROPLETPHASE__BC__

Boundary Conditions for Dropletphase

Set boundary conditions for the Dropletphase solver.

List of members:

BC_v	Velocity boundary Conditions for Dropletphase

BC_v

Velocity boundary Conditions for Dropletphase

Set velocity boundary conditions for the Dropletphase solver.

List of members:

%BND_COLLISION%	velocity boundary condition to represent collisions

%BND_COLLISION%

velocity boundary condition to represent collisions

BC_v ($BC1$) = (%BND_COLLISION% , k_n, e_n, E_a, R_a, mu, SplitFactor, theta)

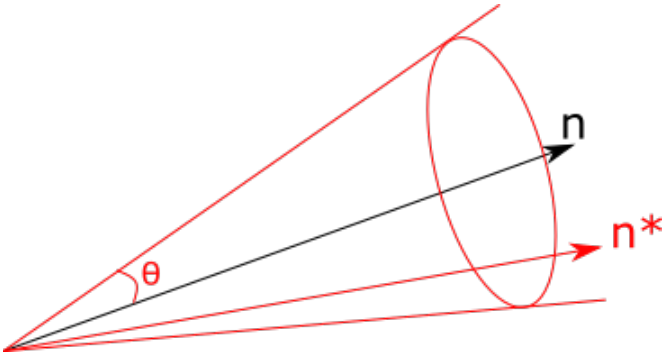
BND_COLLISION is a velocity boundary condition for particles within a DROPLETPHASE chamber. Particle dynamic when colliding with a boundary element with this boundary condition is modeled as a mass spring damper model. Additionally, an adhesive force can be applied, friction can be incorporated, energy dissipated at the boundary element can be modeled and a model for the roughness of the boundary element can be employed.

The adhesion/collision model is determined by the first five parameters k_n, e_n, E_a, R_a, mu (see DropletCollisions).

Parameter	Meaning	Possible Values	Default
k_n	Spring Constant for particle interaction	k_n >= 0.0	0.0 (no collision modeling)
e_n	if 0 <= e_n <= 1 Coefficient of Restitution (0 ideal plastic, 1.0 ideal elastic), if e_n < 0, negative value of the damping coefficient	between 0 and 1 or negative	0.0
E_a	Adhesive potential difference relative to the particle mass	non-negative	0.0 (no adhesion)
R_a	Broadness of zone of attraction relative to d30	non-negative	1.0
mu	Friction Coefficient	non-negative	0.0 (off)
SplitFactor	Fraction of total dissipated energy in collision dissipated at wall	0.0 <= SplitFactor <= 1.0	0.0 (no energy dissipated at wall)
theta	Roughness: maximum angle of random perturbance of normals	0.0 <= theta <= pi/2	0.0 (no roughness)

Additionally for particle boundary interaction, the two parameters SplitFactor and theta may be specified:

- SplitFactor determines the fraction of energy dissipated by the wall: The energy calculated within the collision is split up between particle and wall in the given ratio.
- theta is given, the boundary normals will be randomly perturbed in order to model surface roughness. The value of theta, $\theta \in [0, \frac{\pi}{2}]$, determines the maximum angle between the modified normal vector $\boldsymbol{n}^*$ and the original one $\boldsymbol{n}$:



Example:

```
BC_v ( $BC1$ ) = ( %BND_COLLISION% , 1.1 , .1 , 1e-3, 1.0, 0.8, 0.5 , 0.02)
```


LIQUID__BC__

definition of physical boundary conditions for LIQUID solver

Required boundary conditions for LIQUID

For all participating geometry items mandatory boundary conditions must be defined depending on the choices for the solver.
v--

If the scheme v-- is chosen, then boundary conditions for the velocity and the hydrostatic pressure must be specified by the user.
vp-

If the scheme vp- is chosen, then boundary conditions for the velocity, the hydrostatic pressure and the dynamic pressure must be given by the user.
The boundary condition for hydrostatic pressure and dynamic pressure must be chosen suitable to each other.
Temperature

If temperature is also included in the simulation, then also boundary conditions for the temperature must be defined.
Turbulence

If the k-epsilon turbulence model is included, then boundary conditions for k and epsilon must be defined.

List of members:

BC_CNTFORCE	force contact between phases
BC_eps	turbulence-epsilon boundary conditions
BC_k	turbulence-k boundary conditions
BC_p	pressure boundary conditions
BC_S	stress tensor boundary conditions
BC_T	temperature boundary conditions
BC_TearOffCriterion	establish a tear-off criterion for release from walls
BC_v	velocity boundary conditions
BC_WettingAngle	define the contact angle between free surface
BCON	general setting of boundary conditions
BCON_CNTCT	general setting of contact boundary conditions

BCON

general setting of boundary conditions

Set boundary conditions for any variable (see Indices) for which a partial differential equation (PDE) has to be solved.
The general syntax is

```
BCON ($BCflag$,%ind_Variable%) = RightHandSideExpression
```

This is especially important in the framework of CODI . For all variables used in a CODI -environment, this boundary condition
feature is important and completes the setup of the PDE.

Example:

```
CODI_D ( $MAT$ ,%indU_userdefined%) = 10000
CODI_Q ( $MAT$ ,%indU_userdefined%) = 1
BCON ( $BND1$ ,%indU_userdefined%) = ( %BND_DIRICH% , 0 )
BCON ( $BND2$ ,%indU_userdefined%) = ( %BND_NEUMANN% , 0 )
```

However, this is a general function. The convenience functions BC_... are shortcuts to BCON :
BC_v (BND) -> BCON (BND,%ind_v(1)%)
BC_p (BND) -> BCON (BND,%ind_p%)
BC_T (BND) -> BCON (BND,%ind_T%)
BC_k (BND) -> BCON (BND,%ind_k%)
BC_eps (BND) -> BCON (BND,%ind_eps%)
BC_S (BND) -> BCON (BND,%ind_Sxx%)

List of members:

%ind_c%	correction pressure boundary conditions
%ind_p_dyn%	dynamic pressure boundary conditions

%ind_p_dyn%

dynamic pressure boundary conditions

```
BCON ( $BCindex$ ,%ind_p_dyn%) = ( %BND_slip% )
BCON ( $BCindex$ ,%ind_p_dyn%) = ( %BND_wall% , OPTIONAL:RegularizationParameter ,
OPTIONAL:LimitationOfAccelerationOfBoundary )
BCON ( $BCindex$ ,%ind_p_dyn%) = ( %BND_wall_nosl% , OPTIONAL:RegularizationParameter ,
OPTIONAL:LimitationOfAccelerationOfBoundary )
BCON ( $BCindex$ ,%ind_p_dyn%) = ( %BND_NEUMANN% , Value )
BCON ( $BCindex$ ,%ind_p_dyn%) = ( %BND_VONNEU% , Value ) # legacy only
BCON ( $BCindex$ ,%ind_p_dyn%) = ( %BND_DIRICH% , Value )
BCON ( $BCindex$ ,%ind_p_dyn%) = ( %BND_free_implicit% )
BCON ( $BCindex$ ,%ind_p_dyn%) = ( %BND_free% )
BCON ( $BCindex$ ,%ind_p_dyn%) = ( %BND_none% )
BCON ( $BCindex$ ,%ind_p_dyn%) = ( %BND_AVERAGE% )
```

List of members:

%BND_wall%	quasi-stationary dynamic pressure boundary condition
%BND_wall_nosl%	quasi-stationary dynamic pressure boundary condition
%BND_inflow%	quasi-stationary dynamic pressure boundary condition
%BND_slip%	direct dynamic pressure boundary conditions
%BND_AVERAGE%	weighted average of the pressure values in the neigborhood of the boundary point
%BND_VONNEU%	Neumann boundary conditions for the pressure (require a dedicated slope of the function in normal direction)
%BND_NEUMANN%	Neumann boundary conditions for the pressure (require a dedicated slope of the function in normal direction)
%BND_DIRICH%	classical Dirichlet condition (prescribe the function value at the boundary)
%BND_outflow%	relaxed Dirichlet conditions, penalize differences between the current and the requested boundary values
%BND_none%	treat the boundary point as if it would be an interior point
%BND_free%	direct dynamic pressure boundary conditions at free surface
%BND_free_implicit%	direct dynamic pressure boundary conditions at free surface
%BND_free_implicit_InContact_explicit%	direct dynamic pressure boundary conditions at phase boundary

%BND_AVERAGE%

weighted average of the pressure values in the neigborhood of the boundary point

```
BCON ( $Material$ ,%ind_p_dyn%) = ( %BND_AVERAGE% )
```

We define the average value of the pressure in the neighborhod of boundary point and assign it to the boundary point

$$(p_{dyn})_i = \frac{\sum_{j \in N_i, i \neq j} W_{ij} \, (p_{dyn})_j}{\sum_{j \in N_i, i \neq j} W_{ij}}$$

%BND_DIRICH%

classical Dirichlet condition (prescribe the function value at the boundary)

```
BCON ( $Material$ ,%ind_p_dyn%) = ( %BND_DIRICH% , p_0 )
```

$$(p_{dyn})_i = p_0$$

%BND_NEUMANN%

Neumann boundary conditions for the pressure (require a dedicated slope of the function in normal direction)

```
BCON ( $Material$ ,%ind_p_dyn%) = ( %BND_NEUMANN% , slope ) # this is the correction version, this type of boundary conditions goes back to Carl Gottfreid Neumann,
BCON ( $Material$ ,%ind_p_dyn%) = ( %BND_VONNEU% , slope ) # originally we wrongly assumed the boundary condition goes back to JOhn von Neumann (famous for his stability analysis of PDE)
```

$$\left(\frac{\partial p_{dyn}}{\partial n} \right)_i = \text{slope}$$

The user has to provide a useful value for the slope.
The boundary conditions %BND_wall% and %BND_slip% are also of Neumann type. Here, MESHFREE computes the value of the slope by itself.

%BND_VONNEU%

Neumann boundary conditions for the pressure (require a dedicated slope of the function in normal direction)

```
BCON ( $Material$ ,%ind_p_dyn%) = ( %BND_NEUMANN% , slope ) # this is the correction version, this type of boundary conditions goes back to Carl Gottfreid Neumann,
BCON ( $Material$ ,%ind_p_dyn%) = ( %BND_VONNEU% , slope ) # originally we wrongly assumed the boundary condition goes back to JOhn von Neumann (famous for his stability analysis of PDE)
```

$$\left(\frac{\partial p_{dyn}}{\partial n}\right)_i = \text{slope}$$

The user has to provide a useful value for the slope.
The boundary conditions %BND_wall% and %BND_slip% are also of Neumann type. Here, MESHFREE computes the value of the slope by itself.

%BND_free%

direct dynamic pressure boundary conditions at free surface

```
BCON ($BCindex$,%ind_p_dyn%) = ( %BND_free% )
BCON ($BCindex$,%ind_p_dyn%) = ( %BND_free% , OuterDynamicPressure )
BCON ($BCindex$,%ind_p_dyn%) = ( %BND_free% , OuterDynamicPressure, RelaxationFactor )
```

The pressure at the free surface is given by the dynamic, viscous stretch of the free surface. The general condition is:

$$p_{\text{hyd}} + p_{\text{dyn}} = \boldsymbol{n}^T \boldsymbol{S}_{\text{visc}} + S_{\text{body}} \boldsymbol{n} + p_{\text{dyn}}^{outer} + p_{\text{hyd}}^{outer}$$

Again, the hydrostatic part is already taken care of such that the remaining part for the dynamic pressure is (**OuterDynamicPressure** = p_{dyn}^{outer}):

$$p_{\text{dyn}} = \boldsymbol{n}^T \boldsymbol{S}_{\text{visc}} \boldsymbol{n} + p_{\text{dyn}}^{outer}$$

If the **RelaxationFactor** is used, we have the constraint

$$p_{\text{dyn}}^{n+1} = \text{RelaxationFactor} \cdot \boldsymbol{n}^T \boldsymbol{S}_{\text{visc}} \boldsymbol{n} + p_{\text{dyn}}^{outer} + (1 - \text{RelaxationFactor}) \cdot p_{\text{dyn}}^{n}.$$

This will only be applied, if the v-- solver is active for the present boundary point.

In order to detect free surfaces, the parameter compute_FS must be set to 'YES' for the chamber, where the boundary condition shall be applied.

%BND_free_implicit%

direct dynamic pressure boundary conditions at free surface

```
BCON ($BCindex$,%ind_p_dyn%) = ( %BND_free_implicit% )
BCON ($BCindex$,%ind_p_dyn%) = ( %BND_free_implicit% , OuterDynamicPressure )
```

The pressure at the free surface is given by the dynamic, viscous stretch of the free surface. The general conditions is:

$$p_{\text{hyd}} + p_{\text{dyn}} = \boldsymbol{n}^T \boldsymbol{S}_{\text{visc}} + S_{\text{body}} \boldsymbol{n} + p_{\text{dyn}}^{outer} + p_{\text{hyd}}^{outer}$$

Again, the hydrostatic part is already taken care of such that the remaining part for the dynamic pressure is (**OuterDynamicPressure** = p_{dyn}^{outer}):

$$p_{\mathrm{dyn}} = \boldsymbol{n}^T \boldsymbol{S}_{\mathrm{visc}} \boldsymbol{n} + p_{\mathrm{dyn}}^{outer}$$

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BCON · %ind_p_dyn% · %BND_free_implicit_InContact_explicit%

%BND_free_implicit_InContact_explicit%

direct dynamic pressure boundary conditions at phase boundary

BCON ($BCindex$,%ind_p_dyn%) = (%BND_free_implicit_InContact_explicit%)

The pressure condition on phase boundaries is:

$$p_{\mathrm{dyn}}^{n+1} = \boldsymbol{n}^T \cdot \boldsymbol{S}_{\mathrm{visc}}^{n+1} \cdot \boldsymbol{n} - \boldsymbol{n}^T \cdot \left(\boldsymbol{S}_{\mathrm{visc}}^{n+1}\right)_{\mathrm{opp}} \cdot \boldsymbol{n} + \left(p_{\mathrm{dyn}}^{n}\right)_{\mathrm{opp}}$$

In order to detect free surfaces, the parameter compute_FS must be set to 'YES' for the chamber, where the boundary condition shall be applied.

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BCON · %ind_p_dyn% · %BND_inflow%

%BND_inflow%

quasi-stationary dynamic pressure boundary condition

BCON ($BCindex$,%ind_p_dyn%) = (%BND_inflow%)
BCON ($BCindex$,%ind_p_dyn%) = (%BND_inflow% , OPTIONAL:RegularizationParameter)

From the momentum equation

$$\frac{d\boldsymbol{v}}{dt} + \frac{1}{\rho} \cdot \nabla p_{\mathrm{hyd}} + \frac{1}{\rho} \cdot \nabla p_{\mathrm{dyn}} = \frac{1}{\rho} \nabla^T \boldsymbol{S} + \boldsymbol{g}$$

the boundary conditions can be derived by multiplying the boundary normal from left and ignoring the terms connected to the hydrostatic pressure, i.e.

$$\rho \boldsymbol{n}^T \cdot \frac{d\boldsymbol{v}}{dt} + \frac{\partial p_{\mathrm{dyn}}}{\partial n} = \boldsymbol{n}^T \cdot \left(\nabla^T \boldsymbol{S}_{\mathrm{visc}}\right)^T.$$

As the boundary points are not necessarily moving with fliud velocity $\boldsymbol{v}$, we use

$$\frac{d\boldsymbol{v}}{dt} = \frac{d\boldsymbol{v}_p}{dt} + \left(\boldsymbol{v}^T - \boldsymbol{v}_p^T\right) \cdot \left(\boldsymbol{v}^T \cdot \nabla\right) \boldsymbol{v},$$

where $\boldsymbol{v}_p$ is the velocity the boundary point is actually moving with and $\frac{d\boldsymbol{v}_p}{dt} \approx \frac{\boldsymbol{v}^{n+1} - \boldsymbol{v}^n}{\Delta t}$ is an easy, first order time difference
in order to approximate the velocity change of a MESHFREE point moving with $\boldsymbol{v}_p$.

Regularization: The RegularizationParameter is a small value, something like 1.0e-4. In order to regularize the boundary condition,
the classical Neumann-type condition

$$\frac{\partial p_{\mathrm{dyn}}}{\partial n} = A$$

is turned into a Nusselt-type condition

$$\frac{\partial p_{\mathrm{dyn}}}{\partial n} = A + \frac{\mathrm{RegularizationParameter}}{h} p_{\mathrm{dyn}},$$

where h is the local smoothing length at the boundary point.

%BND_none%

treat the boundary point as if it would be an interior point

ATTENTION: this type of boundary condition is, theoretically, not valid in general if used with FLIQUID_ConsistentPressure_Version =?1?? , i.e. a 1 at the second digit. Here, the accelarations are considered to be quasistationary,
hence %BND_none% will make a mistake if used in a non-quasistationary setting.

```
BCON ( $Material$ ,%ind_p_dyn%) = ( %BND_none% , OPTIONAL: AllowBoundaryAcceleration ...
OPTIONAL: WeightKernel )
```

With the same ansatz as in AlternativeDPA , we solve for the boundary point i the equation

$$\sum_{j=1}^{N(i)} W_{ij} \left(\frac{1}{2} \left(\mathbf{x}_j - \mathbf{x}_i \right)^T \cdot \frac{1}{\rho_j} \nabla p_j^{\mathrm{target}} + \frac{1}{2} \left(\mathbf{x}_j - \mathbf{x}_i \right)^T \cdot \frac{1}{\rho_i} \nabla p_i^{\mathrm{target}} \right) = \sum_{j=1}^{N(i)} W_{ij} \left(\frac{1}{2} \frac{\rho_i + \rho_j}{\rho_i \rho_j} \left(p_j - p_i \right) \right)$$

Remember that $\frac{1}{\rho_j} \nabla p_j^{\mathrm{target}}$ might contain only the stationary part of the substantial derivative $\frac{d\mathbf{v}}{dt} \approx \left(\mathbf{v}^T \cdot \nabla \right) \mathbf{v}$, so we provide the option:
AllowBoundaryAcceleration: has to be bigger than zero.
If the optional parameter is given, we enhance the equation to

$$\sum_{j=1}^{N(i)} W_{ij} \left(\frac{1}{2} \left(\mathbf{x}_j - \mathbf{x}_i \right)^T \cdot \frac{1}{\rho_j} \nabla p_j^{\mathrm{target}} + \frac{1}{2} \left(\mathbf{x}_j - \mathbf{x}_i \right)^T \cdot \left(\frac{1}{\rho_i} \nabla p_i^{\mathrm{target}} - \frac{d\mathbf{v}_i^{BND}}{dt} \right) \right) = \sum_{j=1}^{N(i)} W_{ij} \left(\frac{1}{2} \frac{\rho_i + \rho_j}{\rho_i \rho_j} \left(p_j - p_i \right) \right)$$

where we restrict the magnitude of the acceleration of the boundary to the optional value given, i.e. $\left\| \frac{d\mathbf{v}_i^{BND}}{dt} \right\| \leq \mathrm{AllowBoundaryAcceleration}$
In order to allow the true acceleration, set the value hugh enough. DEFAULT: 0

WeightKernel: the %BND_none% conditions can be put into practice ONLY treating the boundary point acording to AlternativeDPA .
This requires a weight kernel, which is defined by this optional parameter. If set to 0, the classical Neumann ctencil is used as a weight. Otherwise, $W_{ij} = exp\left(-\alpha \cdot r_{ij}^2\right)$.
The value of **WeightKernel:** then defines the parameter α . DEFAULT: 0

%BND_outflow%

relaxed Dirichlet conditions, penalize differences between the current and the requested boundary values

BCON ($Material$,%ind_p_dyn%) = (%BND_outflow% , p_0, alpha)

$$(p_{dyn})_i + \alpha h_i \cdot \left(\frac{\partial p_{dyn}}{\partial n}\right)_i = p_0$$

Rewriting this equation gives

$$\frac{\partial p_{dyn}}{\partial n}_i = \frac{1}{\alpha h} p_0 - \frac{1}{\alpha h} p_{dyni}$$

that means we prescribe the slope of the pressure based on the difference between the current and requested function values.
Thus, it reveals the penalty character of this type of boundary condition, as a big alpha alpha emphasizes the function slope,
whereas a small alpha forces the boundray value to assume p_0 .

%BND_slip%

direct dynamic pressure boundary conditions

From the momentum equation

$$\frac{d\boldsymbol{v}}{dt} + \frac{1}{\rho} \cdot \nabla p_{\mathrm{hyd}} + \frac{1}{\rho} \cdot \nabla p_{\mathrm{dyn}} = \frac{1}{\rho} \nabla^T \boldsymbol{S} + \boldsymbol{g}$$

the boundary conditions can be derived by multiplying from left with the boundary normal

$$\boldsymbol{n}^T \cdot \frac{d\boldsymbol{v}}{dt} + \frac{1}{\rho} \frac{\partial p_{\mathrm{hyd}}}{\partial n} + \frac{1}{\rho} \frac{\partial p_{\mathrm{dyn}}}{\partial n} = \frac{1}{\rho} \boldsymbol{n}^T \cdot \left(\nabla^T \boldsymbol{S}\right)^T + \boldsymbol{n}^T \cdot \boldsymbol{g}.$$

Since the BC for the hydrostatic part is already taken care of, the remaining equation for dynamic pressure is

$$\boldsymbol{n}^T \cdot \frac{d\boldsymbol{v}}{dt} + \frac{1}{\rho} \frac{\partial p_{\mathrm{dyn}}}{\partial n} = \frac{1}{\rho} \boldsymbol{n}^T \cdot \left(\nabla^T \boldsymbol{S}_{\mathrm{visc}}\right)^T,$$

which finally leads to

$$\frac{\partial p_{\mathrm{dyn}}}{\partial n} = -\rho \left(\boldsymbol{n}^T \cdot \frac{d\boldsymbol{v}}{dt}\right) + \boldsymbol{n}^T \cdot \left(\nabla^T \boldsymbol{S}_{\mathrm{visc}}\right)^T.$$

For this type of boundary condition, the acceleration term is numerically approximated by $\boldsymbol{n}^T \cdot \frac{d\boldsymbol{v}}{dt} \approx \boldsymbol{n}^T \cdot \frac{\boldsymbol{v}^{n+1} - \boldsymbol{v}^n}{\Delta t}$.

%BND_wall%

quasi-stationary dynamic pressure boundary condition

BCON ($BCindex$,%ind_p_dyn%) = (%BND_wall%)
BCON ($BCindex$,%ind_p_dyn%) = (%BND_wall% , OPTIONAL:RegularizationParameter , OPTIONAL:LimitationOfAccelerationOfBoundary)

From the momentum equation

$$\frac{d\boldsymbol{v}}{dt} + \frac{1}{\rho} \cdot \nabla p_{\mathrm{hyd}} + \frac{1}{\rho} \cdot \nabla p_{\mathrm{dyn}} = \frac{1}{\rho} \nabla^T \boldsymbol{S} + \boldsymbol{g}$$

the boundary conditions can be derived by multiplying the boundary normal from left:

$$\boldsymbol{n}^T \cdot \frac{d\boldsymbol{v}}{dt} + \frac{1}{\rho} \frac{\partial p_{\mathrm{hyd}}}{\partial n} + \frac{1}{\rho} \frac{\partial p_{\mathrm{dyn}}}{\partial n} = \frac{1}{\rho} \boldsymbol{n}^T \cdot \left(\nabla^T \boldsymbol{S} \right)^T + \boldsymbol{n}^T \cdot \boldsymbol{g}$$

Let us extract the part for the dynamic pressure, which is:

$$\frac{1}{\rho} \frac{\partial p_{\mathrm{dyn}}}{\partial n} = -\boldsymbol{n}^T \cdot \frac{d\boldsymbol{v}}{dt} + \frac{1}{\rho} \boldsymbol{n}^T \cdot \left(\nabla^T \boldsymbol{S}_{visc} \right)^T$$

In order to bring in the acceleration of the boundary, we choose an observation point that travels with the moving boundary, so we have (zero addition)

$$\boldsymbol{v} = (\boldsymbol{v} - \boldsymbol{v}_w) + \boldsymbol{v}_w$$

where $\boldsymbol{v}_w$ is the travelling velocity of the observation system. The total time derivative of this term yields

$$\frac{d\boldsymbol{v}}{dt} = \frac{d}{dt} (\boldsymbol{v} - \boldsymbol{v}_w) + \frac{d\boldsymbol{v}_w}{dt}$$

We can rewrite the first term as

$$\frac{d}{dt} (\boldsymbol{v} - \boldsymbol{v}_w) = \frac{\partial}{\partial t} (\boldsymbol{v} - \boldsymbol{v}_w) + \left((\boldsymbol{v} - \boldsymbol{v}_w)^T \cdot \nabla \right) (\boldsymbol{v} - \boldsymbol{v}_w)$$

and under the assumption of quasistationary flow in the travelling observation system, we have

$$\frac{d}{dt} (\boldsymbol{v} - \boldsymbol{v}_w) = \left((\boldsymbol{v} - \boldsymbol{v}_w)^T \cdot \nabla \right) (\boldsymbol{v} - \boldsymbol{v}_w)$$

Finally, the Neumann condition imposed on the dynamic pressure is

$$\frac{1}{\rho} \frac{\partial p_{\mathrm{dyn}}}{\partial n} = -\boldsymbol{n}^T \cdot \left(\left((\boldsymbol{v} - \boldsymbol{v}_w)^T \cdot \nabla \right) (\boldsymbol{v} - \boldsymbol{v}_w) + \frac{d}{dt} \boldsymbol{v}_w \right) + \frac{1}{\rho} \boldsymbol{n}^T \cdot \left(\nabla^T \boldsymbol{S}_{visc} \right)^T := \frac{A}{\rho}$$

The quasistationary term can be rewritten in terms of the n-, a-, and b- directions, i.e. the normal (n) and the two tangential directions (a,b) of the wall, which form a perpendicular system:

$$\begin{aligned} \boldsymbol{n}^T \cdot \left((\boldsymbol{v} - \boldsymbol{v}_w)^T \cdot \nabla \right) (\boldsymbol{v} - \boldsymbol{v}_w) = & \boldsymbol{n}^T (\boldsymbol{v} - \boldsymbol{v}_w) \cdot \tfrac{\partial}{\partial n} (\boldsymbol{n}^T (\boldsymbol{v} - \boldsymbol{v}_w)) + \\ & \boldsymbol{a}^T (\boldsymbol{v} - \boldsymbol{v}_w) \cdot \tfrac{\partial}{\partial a} (\boldsymbol{n}^T (\boldsymbol{v} - \boldsymbol{v}_w)) + \\ & \boldsymbol{b}^T (\boldsymbol{v} - \boldsymbol{v}_w) \cdot \tfrac{\partial}{\partial b} (\boldsymbol{n}^T (\boldsymbol{v} - \boldsymbol{v}_w)) \end{aligned}$$

The first term is usually zero, if there is no penetration through the wall.
The other two terms are nonzero, if there is tangential slip. In this case, they represent the centrifugal forces, if sliding along a curved boundary.

Regularization: The RegularizationParameter is a small value, something like 1.0e-4. In order to regularize the boundary condition,
the classical Neumann-type condition

$$\frac{\partial p_{\mathrm{dyn}}}{\partial n} = A$$

is turned into a Nusselt-type condition

$$\frac{\partial p_{\mathrm{dyn}}}{\partial n} = A + \frac{\mathrm{RegularizationParameter}}{h} p_{\mathrm{dyn}},$$

where h is the local smoothing length at the boundary point.

LimitationOfAccelerationOfBoundary: if set > 0.0, MESHFREE limits/cuts the current (i.e. measured) acceleration of the boundary elements down to this magnitude.

IMPORTANT:

- In the case of %BND_wall% -> $\boldsymbol{v}_w = \boldsymbol{v}_{\mathrm{trueWallVelocity}}$
- In the case of %BND_wall_nosl% -> $\boldsymbol{v}_w = \boldsymbol{v}eqlwhichprovideseql\boldsymbol{n}^T \cdot \left((\boldsymbol{v}-\boldsymbol{v}_w)^T \cdot \nabla\right)(\boldsymbol{v}-\boldsymbol{v}_w) = 0$ #

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BCON · %ind_p_dyn% · %BND_wall_nosl%

%BND_wall_nosl%

quasi-stationary dynamic pressure boundary condition

BCON ($BCindex$,%ind_p_dyn%) = (%BND_wall_nosl%)
BCON ($BCindex$,%ind_p_dyn%) = (%BND_wall_nosl% , OPTIONAL:RegularizationParameter , OPTIONAL:LimitationOfAccelerationOfBoundary)

Same as %BND_wall% .

From the momentum equation

$$\frac{d\boldsymbol{v}}{dt} + \frac{1}{\rho}\cdot\nabla p_{\mathrm{hyd}} + \frac{1}{\rho}\cdot\nabla p_{\mathrm{dyn}} = \frac{1}{\rho}\nabla^T\boldsymbol{S} + \boldsymbol{g}$$

the boundary conditions can be derived by multiplying the boundary normal from left:

$$\boldsymbol{n}^T\cdot\frac{d\boldsymbol{v}}{dt} + \frac{1}{\rho}\frac{\partial p_{\mathrm{hyd}}}{\partial n} + \frac{1}{\rho}\frac{\partial p_{\mathrm{dyn}}}{\partial n} = \frac{1}{\rho}\boldsymbol{n}^T\cdot\left(\nabla^T\boldsymbol{S}\right)^T + \boldsymbol{n}^T\cdot\boldsymbol{g}$$

Let us extract the part for the dynamic pressure, which is:

$$\frac{1}{\rho}\frac{\partial p_{\mathrm{dyn}}}{\partial n} = -\boldsymbol{n}^T\cdot\frac{d\boldsymbol{v}}{dt} + \frac{1}{\rho}\boldsymbol{n}^T\cdot\left(\nabla^T\boldsymbol{S}_{visc}\right)^T$$

In order to bring in the acceleration of the boundary, we choose an observation point that travels with the moving boundary, so we have (zero addition)

$$\boldsymbol{v} = (\boldsymbol{v}-\boldsymbol{v}_w) + \boldsymbol{v}_w$$

where $\boldsymbol{v}_w$ is the travelling velocity of the observation system. The total time derivative of this term yields

$$\frac{d\boldsymbol{v}}{dt} = \frac{d}{dt}(\boldsymbol{v}-\boldsymbol{v}_w) + \frac{d\boldsymbol{v}_w}{dt}$$

We can rewrite the first term as

$$\frac{d}{dt}(\boldsymbol{v}-\boldsymbol{v}_w) = \frac{\partial}{\partial t}(\boldsymbol{v}-\boldsymbol{v}_w) + \left((\boldsymbol{v}-\boldsymbol{v}_w)^T\cdot\nabla\right)(\boldsymbol{v}-\boldsymbol{v}_w)$$

and under the assumption of quasistationary flow in the travelling observation system, we have

$$\frac{d}{dt}(\boldsymbol{v}-\boldsymbol{v}_w) = \left((\boldsymbol{v}-\boldsymbol{v}_w)^T\cdot\nabla\right)(\boldsymbol{v}-\boldsymbol{v}_w)$$

Finally, the Neumann condition imposed on the dynamic pressure is

$$\frac{1}{\rho}\frac{\partial p_{\mathrm{dyn}}}{\partial n} = -\boldsymbol{n}^T\cdot\left(\left((\boldsymbol{v}-\boldsymbol{v}_w)^T\cdot\nabla\right)(\boldsymbol{v}-\boldsymbol{v}_w) + \frac{d}{dt}\boldsymbol{v}_w\right) + \frac{1}{\rho}\boldsymbol{n}^T\cdot\left(\nabla^T\boldsymbol{S}_{visc}\right)^T := \frac{A}{\rho}$$

The quasistationary term can be rewritten in terms of the n-, a-, and b- directions, i.e. the normal (n) and the two tangential directions (a,b) of the wall, which form a perpendicular system:

$$\begin{aligned}\boldsymbol{n}^T\cdot\left((\boldsymbol{v}-\boldsymbol{v}_w)^T\cdot\nabla\right)(\boldsymbol{v}-\boldsymbol{v}_w) =& \boldsymbol{n}^T(\boldsymbol{v}-\boldsymbol{v}_w)\cdot\tfrac{\partial}{\partial n}(\boldsymbol{n}^T(\boldsymbol{v}-\boldsymbol{v}_w)) + \\ & \boldsymbol{a}^T(\boldsymbol{v}-\boldsymbol{v}_w)\cdot\tfrac{\partial}{\partial a}(\boldsymbol{n}^T(\boldsymbol{v}-\boldsymbol{v}_w)) + \\ & \boldsymbol{b}^T(\boldsymbol{v}-\boldsymbol{v}_w)\cdot\tfrac{\partial}{\partial b}(\boldsymbol{n}^T(\boldsymbol{v}-\boldsymbol{v}_w))\end{aligned}$$

The first term is usually zero, if there is no penetration through the wall.
The other two terms are nonzero, if there is tangential slip. In this case, they represent the centrifugal forces, if sliding along a curved boundary.

Regularization: The RegularizationParameter is a small value, something like 1.0e-4. In order to regularize the boundary condition,
the classical Neumann-type condition

$$\frac{\partial p_{\mathrm{dyn}}}{\partial n} = A$$

is turned into a Nusselt-type condition

$$\frac{\partial p_{\mathrm{dyn}}}{\partial n} = A + \frac{\mathrm{RegularizationParameter}}{h} p_{\mathrm{dyn}},$$

where h is the local smoothing length at the boundary point.

LimitationOfAccelerationOfBoundary: if set > 0.0, MESHFREE limits/cuts the current (i.e. measured) acceleration of the boundary elements down to this magnitude.

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BCON_CNTCT

BCON_CNTCT

general setting of contact boundary conditions

This feature rules the contact conditions (interphase conditions) between contacting phases.
The general syntax is

```
BCON_CNTCT ($BCflag$,%ind_Variable%) = RightHandSideExpression
```

Contact can occur between regular boundaries of two chambers or free surface points of two chambers.
Contact cannot appear between a free surface point of one chamber with a regular boundary point of another chamber.

Note: For most cases, we recommend using the Darcy approach, see TwoPhaseDarcy , instead of explicit interphase conditions.

List of members:

%ind_T%	temperature boundary conditions at interfaces
%ind_v(1)%	velocity boundary conditions at interfaces

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BCON_CNTCT · %ind_T%

%ind_T%

temperature boundary conditions at interfaces

This feature rules the contact conditions (interphase conditions) for the temperature between contacting phases.
It must be added to the BC_T condition of the corresponding alias. There are two possible conditions that can be specified here. Suppose we have two contact boundaries contact_boundary1 and contact_boundary2.
The syntax for modeling ideal heat transition (ideal contact) is

```
BC_T ( $contact_boundary1$ ,%ind_T%) = ( %BND_NEUMANN% , 0)
BCON_CNTCT ( $contact_boundary1$ ,%ind_T%) = (%BND_contact%)

BC_T ( $contact_boundary2$ ,%ind_T%) = ( %BND_NEUMANN% , 0)
BCON_CNTCT ( $contact_boundary2$ ,%ind_T%) = (%BND_contact%)
```

The software automatically takes care of which phase the Dirichlet condition

$$T = T_{\text{opp}}$$

and the heat flux condition

$$\lambda \frac{\partial T}{\partial n} = \lambda_{\text{opp}} \frac{\partial T_{\text{opp}}}{\partial n},$$

are set to have complementary conditions at the contact point. This is decided based on the two heat conductivities (lambda).

The syntax for modeling heat transition with a user given heat transfer coefficient α is

```
BC_T ( $contact_boundary1$ ,%ind_T%) = (%BND_ROBIN%, , [Yopp( %ind_T% )])
BCON_CNTCT ( $contact_boundary1$ ,%ind_T%) = (%BND_ROBIN%, )

BC_T ( $contact_boundary2$ ,%ind_T%) = (%BND_ROBIN%, , [Yopp( %ind_T% )])
BCON_CNTCT ( $contact_boundary2$ ,%ind_T%) = (%BND_ROBIN%, )
```

In this case the Dirichlet condition (see ideal contact) is replaced by

$$\lambda \frac{\partial T}{\partial n} = \alpha (T - T_{\text{opp}}).$$

Note that BCON_CNTCT is an additional condition that is only active when the boundary points in their neighbourhood detect
points of the other phase. Otherwise, this condition falls back to BC_T .

%ind_v(1)%

velocity boundary conditions at interfaces

List of members:

%BND_slip_InContact%	velocity boundary conditions at interfaces, implicit
%BND_slip_InContact_Explicit%	velocity boundary conditions at interfaces, explicit

%BND_slip_InContact%

velocity boundary conditions at interfaces, implicit

The opposite phase is seen as (moving) wall, along which a slip condition is realized. In all other aspects, this boundary condition is very similar to %BND_slip% .

BCON_CNTCT ($BCindex$,%ind_v(1)%) = (%BND_slip_InContact% , FrictionCoefficient, ControlThicknessMomentum)

FrictionCoefficient: Viscous friction in the sense

$$\boldsymbol{S}\left(\boldsymbol{v}^{n+1}\right) \cdot \boldsymbol{a} = \alpha \cdot \left(\boldsymbol{v}^{n+1} - \boldsymbol{v}_{opp}^{n+1}\right) \cdot \boldsymbol{a}$$

Here, α is the FrictionCoefficient, $\alpha = 0$ would lead to pure slip, $\alpha \to \infty$ would lead to pure no-slip.
If the turbulence model is in action, the effective friction coefficient is given by $\alpha_{eff} = \alpha_{turb} + \alpha$, where $\alpha_{turb} = \alpha_{turb}(k, \epsilon)$.
$\boldsymbol{v}_{opp}^{n+1}$ is the local velocity of the opposite phase (contact phase) at the next time level.

ControlThicknessMomentum: Incorporation of the momentum balance into the boundary condition, especially important for big Re-numbers.
The thickness of the momentum control cell is ControlThicknessMomentum*H (smoothing length).

Make sure to set the ControlThicknessMomentum to ZERO if using EULER , EULERIMPL or EULEREXPL!

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BCON_CNTCT · %ind_v(1)% · %BND_slip_InContact_Explicit%

%BND_slip_InContact_Explicit%

velocity boundary conditions at interfaces, explicit

The opposite phase is seen as (moving) wall, along which a slip condition is realized.

BCON_CNTCT ($BCindex$,%ind_v(1)%) = (%BND_slip_InContact_Explicit% , FrictionCoefficient, ControlThicknessMomentum)

FrictionCoefficient: Viscous friction in the sense

$$\boldsymbol{S}\left(\boldsymbol{v}^{n+1}\right) \cdot \boldsymbol{a} = \alpha \cdot \left(\boldsymbol{v}^{n+1} - \boldsymbol{v}_{opp}^{n}\right) \cdot \boldsymbol{a}$$

Here, α is the FrictionCoefficient, $\alpha = 0$ would lead to pure slip, $\alpha \to \infty$ would lead to pure no-slip.
If the turbulence model is in action, the effective friction coefficient is given by $\alpha_{eff} = \alpha_{turb} + \alpha$, where $\alpha_{turb} = \alpha_{turb}(k, \epsilon)$.
$\boldsymbol{v}_{opp}^{n}$ is the local velocity of the opposite phase (contact phase) at the current time level.

ControlThicknessMomentum: Incorporation of the momentum balance into the boundary condition, especially important for big Re-numbers.
The thickness of the momentum control cell is ControlThicknessMomentum*H (smoothing length).

Make sure to set the ControlThicknessMomentum to ZERO if using EULER , EULERIMPL or EULEREXPL!

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_CNTFORCE

BC_CNTFORCE

force contact between phases

BC_CNTFORCE ($BCindex$) = 1.0

default: BC_CNTFORCE ($BCindex$) = 0.0

Force the contact to the other phase (chamber) if in the neighborhood.

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_T

BC_T

temperature boundary conditions

```
BC_T ($xyz$) = ( %BND_inflow% , inflow boundary condition)
BC_T ($xyz$) = ( %BND_outflow% , outflow boundary condition)



BC_T ($xyz$) = ( %BND_ROBIN% , alpha, T_opp)
BC_T ($xyz$) = ( %BND_CAUCHY% , alpha, T_opp) # see %BND_ROBIN%
```

List of members:

HeatEquation1D	Solves 1D heat equation for each boundary point. Can be used for temperature boundary condition.
%BND_inflow%	temperature inflow boundary condition
%BND_wall%	temperature wall boundary condition
%BND_wall_nosl%	temperature wall no-slip boundary condition
%BND_outflow%	temperature outflow boundary condition
%BND_free%	free surface boundary condition for temperature
%BND_far_field%	far-field temperature boundary condition
%BND_ROBIN%	Robin boundary condition
%BND_RADIATION%	applies heat flux at the boundary due to radiation
%BND_AVERAGE%	weighted average from the inner points
%BND_DIRICH%	temperature Dirichlet boundary condition
%BND_NEUMANN%	temperature Neumann boundary condition
%BND_NUSSEL%	temperature Nusselt boundary condition

%BND_AVERAGE%

weighted average from the inner points

```
#UCVCODE
BC_T ($BOundaryName$) = ( %BND_AVERAGE%, OPTIONAL: useOnlyInnerPoints )
```

Applies weighted average in the Shepard sense:

$$T_i^{n+1} = \frac{\sum_{k=1}^{N_{points}} W_{ik} T_k^{n+1}}{\sum_{k=1}^{N_{points}} W_{ik}}$$

If the oprional parameter is 1, then the weighted average is computed only with respect to the inner points, i.e.

$$T_i^{n+1} = \frac{\sum_{k=1}^{N_{points}} W_{ik} T_k^{n+1}}{\sum_{k=1}^{N_{points}} W_{ik}}$$

with $W_{ik} = 0$ if k denotes a boundary point.

%BND_DIRICH%

temperature Dirichlet boundary condition

Dirichlet (first-type) boundary condition for temperature.

$$T|_\Gamma = \alpha$$

Syntax:

BC_T (xyz) = (%BND_DIRICH% , α)

Example:

```
BC_T ( $wall$ ) = ( %BND_DIRICH% , 400)
```

Sets the temperature at the boundary with BC -flag $wall$ to 400 Kelvin.

Optional: Result of 1D heat equation can be used by keyword %HEAT_EQ_1D_BC% (see HeatEquation1D).
Example:

```
BC_T ( $wall$ ) = ( %BND_DIRICH% , [ &T_BND& ], 0.0, %HEAT_EQ_1D_BC%)
```

Replaces the temperature at the boundary &T_BND& by the result %ind_T1D(1)% of the 1D heat equation. So the value &T_BND& is ignored in this case!

%BND_NEUMANN%

temperature Neumann boundary condition

Neumann (second-type) boundary condition for temperature.

$$\left.\frac{\partial T}{\partial n}\right|_\Gamma = \alpha$$

Syntax:

BC_T (xyz) = (%BND_NEUMANN% , α)

Example 1:
Constant temperature gradient

```
BC_T ( $wall$ ) = ( %BND_NEUMANN% , 10)
```

Applies a constant temperature gradient of $10\frac{\mathrm{K}}{\mathrm{m}}$ at the boundary with BC -flag $wall$.

Example 2:
Constant heat flux

```
begin_alias{ }
"heatflux" = "100" # W/m^2
"heatConductivity" = "2" # W/(mK)
end_alias
...
BC_T ( $wall$ ) = ( %BND_NEUMANN% , [ &heatflux& / &heatConductivity& ])
```

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_T · %BND_NUSSEL%

%BND_NUSSEL%

temperature Nusselt boundary condition

Applies the Nusselt boundary condition for temperature at the boundary.

$$\left.\frac{\partial T}{\partial n}\right|_{\Gamma} = \alpha + \beta T,$$

where α is the flux and β is the flux of higher order.

Syntax:

BC_T (BC_index) = (%BND_NUSSEL% , α , β)

Example:

```
BC_T ( $wall$ ) = ( %BND_NUSSEL% , 10, 0.5)
```

Applies a flux with $\alpha = 10$ and $\beta = 0.5$ at the boundary with BC -flag $wall$.

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_T · %BND_RADIATION%

%BND_RADIATION%

applies heat flux at the boundary due to radiation

Applies a heat flux at the boundary according to the equation

$$\frac{\partial T}{\partial n} = \frac{\alpha}{\lambda}\left(T^4 - T_0^4\right),$$

where $\frac{\alpha}{\lambda} = \sigma\epsilon$ with σ the Stefan-Boltzmann constant and ϵ the emissivity.
λ is the heat conductivity of the material and T_0 the reference temperature.

Syntax:

BC_T (xyz) = (%BND_RADIATION%, α , T_0)

Example:

BC_T ($wall$) = (%BND_RADIATION%, 1.69E-3, 300)

%BND_ROBIN%

Robin boundary condition

In general, a Robin (third-type) boundary condition is a linear combination of a Dirichlet boundary condition (%BND_DIRICH%)
and a Neumann boundary condition (%BND_NEUMANN%) of the form

$$Af + B\frac{\partial f}{\partial n} = g$$

For the temperature T, this can be used to describe how the convective heat flux across the boundary/interface depends on the difference between the temperature of the material at the boundary/interface and the temperature on the opposite side

$$-\lambda\frac{\partial T}{\partial n} = \alpha(T - T_{\text{opp}}),$$

where λ is the heat conductivity of the material, α is a proportionality coefficient for the convective heat flux across the boundary/interface, and T_{opp} is the temperature on the opposite side.

Syntax:

BC_T (xyz) = (%BND_ROBIN% , α , T_{opp})

Example:

BC_T ($wall$) = (%BND_ROBIN% , 10.0, 500.0)

There is a third optional parameter setting the thickness of the control element. A good value is 0.3 .

Note: This type of boundary condition is sometimes known as 'Cauchy boundary condition', but the name is ambiguous. For backward compatibility, the flag %BND_CAUCHY% has the same effect as %BND_ROBIN% .

Optional: Result of 1D heat equation can be used by keyword %HEAT_EQ_1D_BC% (see HeatEquation1D).
Example:

BC_T ($wall$) = (%BND_ROBIN% , [&convective_heat_trans_coeff&], [&Topp&], 0.0, %HEAT_EQ_1D_BC%)

Replaces the temperature on the opposite side T_{opp} by the result %ind_T1D(1)% of the 1D heat equation. So the value &Topp& is ignored in this case!

%BND_far_field%

far-field temperature boundary condition

Dirichlet (first-type) boundary condition which sets the value to
the current temperature: %BND_DIRICH% with $T = T_{current}$.

Syntax:

```
BC_T ($xyz$) = ( %BND_far_field% )
```

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_T · %BND_free%

%BND_free%

free surface boundary condition for temperature

Default temperature boundary condition for free surfaces.

Same as %BND_NEUMANN% with $\frac{\partial T}{\partial n} = 0$.

Syntax:

```
BC_T ($xyz$) = ( %BND_free% )
```

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_T · %BND_inflow%

%BND_inflow%

temperature inflow boundary condition

Dirichlet (first-type) boundary condition which automatically sets
the boundary value to the current temperature (i.e. the temperature is not supposed to change).

Syntax:

```
BC_T ($xyz$) = ( %BND_inflow% )
```

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_T · %BND_outflow%

%BND_outflow%

temperature outflow boundary condition

This boundary condition adapts based on the major flow direction near the outflow boundary.

If the relative velocity of the MESHFREE point to the boundary is pointing outwards, we assume a Neumann (second-type) boundary
condition, i.e. %BND_NEUMANN% with $\frac{\partial T}{\partial n} = 0$.

If, however, the relative velocity is pointing inwards, we assume a Dirichlet (first-type) boundary condition,
i.e. %BND_DIRICH% , with initial temperature $T = T_0$.

Syntax:

```
BC_T ($xyz$) = ( %BND_outflow% )
```

%BND_wall%

temperature wall boundary condition

Same as %BND_NEUMANN% with $\frac{\partial T}{\partial n} = 0$.

Syntax:

```
BC_T ($xyz$) = ( %BND_wall% )
```


%BND_wall_nosl%

temperature wall no-slip boundary condition

Same as %BND_NEUMANN% with $\frac{\partial T}{\partial n} = 0$.
There is no difference to %BND_wall% .

Syntax:

```
BC_T ($xyz$) = ( %BND_wall_nosl% )
```


HeatEquation1D

Solves 1D heat equation for each boundary point. Can be used for temperature boundary condition.

Solves the 1D heat equation

$$\varrho\, c_v \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(\lambda \frac{\partial T}{\partial x}\right)$$

for each boundary point. The 1D points are always equidistantly distributed and the number of points can be controlled by the
common variable NB_POINTS_BC_HEAT_EQUATION_1D . The boundary conditions for the 1D equation are Robin type conditions.
At the interface **MESHFREE - 1D** we use

$$\lambda_{1\mathrm{D}} \frac{\partial T}{\partial x} = \alpha_{\mathrm{in}}(T_{\mathrm{Fluid}} - T_{1\mathrm{D}})$$

and at the interface **1D - outer surrounding** we use

$$\lambda_{1\mathrm{D}} \frac{\partial T}{\partial x} = \alpha_{\mathrm{out}}(T_{1\mathrm{D}} - T_{\mathrm{ambient}})$$

The physical properties for the 1D equation are specified in the following sense:
Syntax:

```
HEAT_EQ_1D($BC_wall$,%ind_xxx%) = (TotalLength, FirstInterval, PhysicalPropFirstInterval, SecondInterval, PhysicalPropSecondInterval, etc.)
```

where %ind_xxx% stands for %ind_T% , %ind_LAM% , %ind_r% or %ind_CV% .

- TotalLength is the total length of the 1D line
- FirstInterval is the length of the first subpart of 1D line
- PhysicalPropFirstInterval is the constant value for %ind_xxx% within the first subpart
- SecondInterval is the length of the second subpart of 1D line
- PhysicalPropSecondInterval is the constant value for %ind_xxx% within the second subpart
- etc.

Exception for %ind_T% : The temperature needs an additional parameter for the outer surrounding which must be always the last entry in HEAT_EQ_1D(BC_wall,%ind_T%).

Example: Modelling of an insulation liner around a cylinder, which consists of two different materials.

```
HEAT_EQ_1D( $BC_wall$ ,%ind_T%) = ( &liner_thickness& , &liner_end_interval1& , &TEMP_alu& ,
&liner_end_interval2& , &TEMP_carbon& , &TEMP_ambient& )
HEAT_EQ_1D( $BC_wall$ ,%ind_LAM%) = ( &liner_thickness& , &liner_end_interval1& , &LAM_alu& ,
&liner_end_interval2& , &LAM_carbon& )
HEAT_EQ_1D( $BC_wall$ ,%ind_r%) = ( &liner_thickness& , &liner_end_interval1& , &RHO_alu& ,
&liner_end_interval2& , &RHO_carbon& )
HEAT_EQ_1D( $BC_wall$ ,%ind_CV%) = ( &liner_thickness& , &liner_end_interval1& , &CV_alu& ,
&liner_end_interval2& , &CV_carbon& )
```

The heat transfer coefficients $\alpha_{\text{in}}, \alpha_{\text{out}}$ must be specified by

```
HEAT_EQ_1D_TRANSFER_COEFF_INTERNAL( $BC_wall$ ) = [ &alpha_in& ]
HEAT_EQ_1D_TRANSFER_COEFF_EXTERNAL( $BC_wall$ ) = [ &alpha_out& ]
```

The results of all 1D heat equations are stored in %ind_T1D(i)% , i = 1:NB_POINTS_BC_HEAT_EQUATION_1D+1. To use these results in the MESHFREE boundary conditions, use the keyword %HEAT_EQ_1D_BC% (see %BND_ROBIN% a n d %BND_DIRICH%). Then the corresponding temperature value in the MESHFREE boundary condition is replaced by %ind_T1D(1)% .

Example:

```
BC_T ( $BC_wall$ ) = ( %BND_DIRICH% , [ &T_BND& ], 0.0, %HEAT_EQ_1D_BC%)
```


BC_TearOffCriterion

establish a tear-off criterion for release from walls

```
BC_TearOffCriterion ($BC_name$) = ( Expression1, Experssion2, ... )
```

A MESHFREE point, attached to a wall, can be released from the wall and turned into a free surface point, if all the given expressions on the right hand side are positive. The expressions are of the form RightHandSideExpression , the typical standard.

BC_name is the BC -flag to be given in the ALIAS definition of the boundary, i.e.

```
begin_alias{ }
"AliasName" = " ... BC$BC_name$ ... "
end_alias
```

Example:

```
BC_TearOffCriterion ( $wall$ ) = ( [(Y %ind_TearOff% -0.5)] , equn{ $EQN_TearOff$ } )

begin_equation{ $EQN_TearOff$ }
(-7)*Y %ind_p_dyn% /Y %ind_r% - (Y %ind_v(1)% -Y %ind_v_p(1)% )^2 - (Y %ind_v(2)% -Y %ind_v_p(2)% )^2 - (Y
%ind_v(3)% -Y %ind_v_p(3)% )^2 - 1000.0/Y %ind_r% # threshold
end_equation
```

This criterion stems from the theoretical ansatz given here .
The last term "1000.0/Y%ind_r%" is a threshold of 1000 Pa in order to avoid release of wall points due to numerical noises.
The condition "[(Y%ind_TearOff%-0.5)]" chooses only those boundary points which are adjacent to a free surface.

The tear off criterion also works for free surface points that are in contact with other phases.
In this case, if the tear-off expressions are positive, the contact to the other phase is canceled,
i.e. the index of opposite point (value in %ind_iopp%) is set to zero.

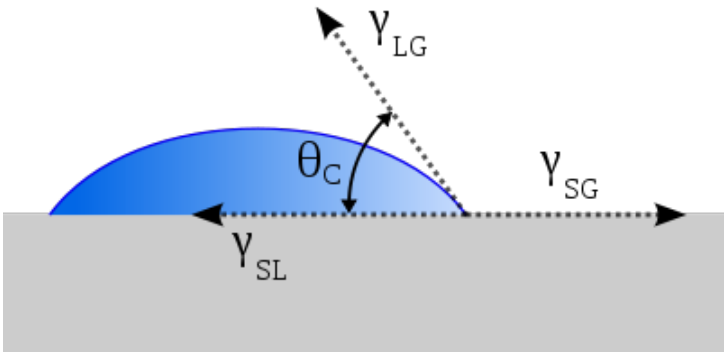
Also see the %ind_TearOff% variable.

BC_WettingAngle

define the contact angle between free surface

```
BC_WettingAngle ($BCindex$) = ( WettingAngle_in_radians )
```

The contact angle is defined between the solid wall and the free surface as shown in the picture below:



The angle is to be given in radians, i.e. a value of π (180 degrees) leads to absolutely hydrophobic (water-repellent) behavior.
A value of 0 leads to absolutely hydrophilic behavior of the liquid material towards the wall.

BC_eps

turbulence-epsilon boundary conditions

If you choose to simulate with the k-epsilon TurbulenceModel (specified in KindOfProblem), you must also provide boundary conditions for epsilon. Possible choices are:

```
BC_eps ( $BCindex$ ) = ( %BND_free% )
BC_eps ( $BCindex$ ) = ( %BND_wall% )
BC_eps ( $BCindex$ ) = ( %BND_wall_nosl% )
BC_eps ( $BCindex$ ) = ( %BND_inflow% )
BC_eps ( $BCindex$ ) = ( %BND_DIRICH% )
BC_eps ( $BCindex$ ) = ( %BND_NEUMANN% )
BC_eps ( $BCindex$ ) = ( %BND_NUSSEL% )
```

List of members:

%BND_free%	free surface boundary condition for turbulence-eps
%BND_wall%	wall boundary condition for turbulence-eps
%BND_wall_nosl%	no-slip wall boundary condition for turbulence-eps
%BND_inflow%	inflow boundary condition for turbulence-eps
%BND_DIRICH%	Dirichlet boundary condition for turbulence-eps
%BND_NEUMANN%	Neumann boundary condition for turbulence-eps
%BND_NUSSEL%	Nusselt boundary condition for turbulence-eps

%BND_DIRICH%

Dirichlet boundary condition for turbulence-eps

```
BC_eps ( $BCindex$ ) = ( %BND_DIRICH% , eps_Dirich )
```

Dirichlet (first-type) boundary condition which automatically sets the boundary value to **eps_Dirich** .

%BND_NEUMANN%

Neumann boundary condition for turbulence-eps

```
BC_eps ( $BCindex$ ) = ( %BND_NEUMANN% , eps_Neumann )
```

Neumann (second-type) boundary condition which automatically sets the normal derivative to **eps_Neumann** .

%BND_NUSSEL%

Nusselt boundary condition for turbulence-eps

Applies the Nusselt boundary condition for turbulence-eps at the boundary.

$$\left.\frac{\partial eps}{\partial n}\right|_{\Gamma} = \alpha + \beta eps,$$

where α is the flux and β is the flux of higher order.

BC_eps (BC_index) = (%BND_NUSSEL% , ,)

%BND_free%

free surface boundary condition for turbulence-eps

Default turbulence-eps boundary condition for free surfaces, i.e. the normal derivative of eps is equal to zero.

BC_eps ($BCindex$) = (%BND_free%)

In order to detect free surfaces, the parameter compute_FS must be set to 'YES' for the chamber, where the boundary condition shall be applied.

%BND_inflow%

inflow boundary condition for turbulence-eps

BC_eps ($BCindex$) = (%BND_inflow%)

Dirichlet (first-type) boundary condition which automatically sets the boundary value of eps to

$$\left(\left(0.0001 \cdot \frac{h}{\Delta t}\right)^2\right)^2 \cdot \frac{0.09}{\left(0.1\frac{\eta}{\rho}\right)}$$

Hereby, $0.0001 \cdot \frac{h}{\Delta t}^2$ corresponds to the inflow value of turbulence-k (see %BND_inflow%).

%BND_wall%

wall boundary condition for turbulence-eps

Default turbulence-eps boundary condition for walls, i.e. the normal derivative of eps is equal to zero.

BC_eps ($BCindex$) = (%BND_wall%)

%BND_wall_nosl%

no-slip wall boundary condition for turbulence-eps

```
BC_eps ( $BCindex$ ) = ( %BND_wall_nosl% )
```

Neumann (second-type) boundary condition which automatically sets the normal derivative to zero.

BC_k

turbulence-k boundary conditions

If you choose to simulate with the k-epsilon TurbulenceModel (specified in KindOfProblem), you must also provide boundary conditions for k. Possible choices are:

```
BC_k ( $BCindex$ ) = ( %BND_free% )
BC_k ( $BCindex$ ) = ( %BND_wall% )
BC_k ( $BCindex$ ) = ( %BND_wall_nosl% )
BC_k ( $BCindex$ ) = ( %BND_inflow% )
BC_k ( $BCindex$ ) = ( %BND_DIRICH% )
BC_k ( $BCindex$ ) = ( %BND_NEUMANN% )
BC_k ( $BCindex$ ) = ( %BND_NUSSEL% )
```

List of members:

%BND_free%	free surface boundary condition for turbulence-k
%BND_wall%	wall boundary condition for turbulence-k
%BND_wall_nosl%	no-slip wall boundary condition for turbulence-k
%BND_inflow%	inflow boundary condition for turbulence-k
%BND_DIRICH%	Dirichlet boundary condition for turbulence-k
%BND_NEUMANN%	Neumann boundary condition for turbulence-k
%BND_NUSSEL%	Nusselt boundary condition for turbulence-k

%BND_DIRICH%

Dirichlet boundary condition for turbulence-k

```
BC_k ( $BCindex$ ) = ( %BND_DIRICH% , k_Dirich )
```

Dirichlet (first-type) boundary condition which automatically sets the boundary value to **k_Dirich** .

%BND_NEUMANN%

Neumann boundary condition for turbulence-k

```
BC_k ( $BCindex$ ) = ( %BND_NEUMANN% , k_Neumann )
```

Neumann (second-type) boundary condition which automatically sets the normal derivative to **k_Neumann** .

%BND_NUSSEL%

Nusselt boundary condition for turbulence-k

Applies the Nusselt boundary condition for turbulence-k at the boundary.

$$\left.\frac{\partial k}{\partial n}\right|_{\Gamma} = \alpha + \beta k,$$

where α is the flux and β is the flux of higher order.

```
BC_k ( $BC_index$ ) = ( %BND_NUSSEL% , , )
```


%BND_free%

free surface boundary condition for turbulence-k

Default turbulence-k boundary condition for free surfaces, i.e. the normal derivative of k is equal to zero.

```
BC_k ( $BCindex$ ) = ( %BND_free% )
```

In order to detect free surfaces, the parameter compute_FS must be set to 'YES' for the chamber, where the boundary condition shall be applied.

%BND_inflow%

inflow boundary condition for turbulence-k

```
BC_k ( $BCindex$ ) = ( %BND_inflow% )
```

Dirichlet (first-type) boundary condition which automatically sets the boundary value of k to

$$\left(0.0001 \cdot \frac{h}{\Delta t}\right)^2$$

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_k · %BND_wall%

%BND_wall%

wall boundary condition for turbulence-k

```
BC_k ( $BCindex$ ) = ( %BND_wall% , OPTIONAL:WallLayerThickness )
```

The MESHFREE points are treated like interior points which are shifted to the interior of the flow domain by $\alpha \dot{h}$.
By default α is equal to WallLayer , for details see DOCUMATH_NumericalIntegrationOfTurbulence.pdf .

WallLayerThickness: α is equal to this parameter independent of the choice of WallLayer .

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_k · %BND_wall_nosl%

%BND_wall_nosl%

no-slip wall boundary condition for turbulence-k

```
BC_k ( $BCindex$ ) = ( %BND_wall_nosl% )
```

Neumann (second-type) boundary condition which automatically sets the normal derivative to zero.

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_p

BC_p

pressure boundary conditions

List of members:

%BND_inflow%	pressure boundary conditions: inflow condition
%BND_AVERAGE%	pressure boundary conditions: average of neighbors (Neumann type)
%BND_wall%	pressure boundary conditions: classical wall
%BND_wall_nosl%	pressure boundary conditions: classical wall
%BND_slip%	pressure boundary conditions: classical wall
%BND_slip_InContact_Explicit%	pressure boundary condition for the case that the contact phase is the heavy phase
%BND_wall_InContact_Explicit%	pressure outflow boundary condition
%BND_free%	pressure free surface boundary condition
%BND_free_InContact_Explicit%	pressure contact boundary conditions for the case the contact phase is the light phase
%BND_slip_InContact%	pressure contact boundary conditions for the case the contact phase is the heavy phase

%BND_AVERAGE%

pressure boundary conditions: average of neighbors (Neumann type)

The pressure boundary condition %BND_AVERAGE% applies the average pressure of the neighbor points to the point. It is a lower order Neumann type condition that sometimes is more robust than %BND_NEUMANN%.
Example:

```
BC_p ( $outflow$ ) = ( %BND_AVERAGE% )
```


%BND_free%

pressure free surface boundary condition

Syntax:

```
BC_p ( $...$ ) = ( %BND_free% ,p0)
```

Equation:

$$p = p^\star + n^T \cdot S \cdot n$$

$$p^\star = p_0 + p_\sigma$$

if the surface tension is 0, then

$$p^\star = p_0$$

In order to detect free surfaces, the parameter compute_FS must be set to 'YES' for the chamber, where the boundary

condition shall be applied.

%BND_free_InContact_Explicit%

pressure contact boundary conditions for the case the contact phase is the light phase

Syntax:

```
BC_p ( $...$ ) = ( %BND_free_InContact_Explicit% )
```

Equation:

$$p = p^\star + n^T \cdot S \cdot n$$

$$p^\star = p_\sigma$$

if the surface tension is 0, then

$$p^\star = p_0$$

the stress tensor and the surface tension pressure is evaluated at the partner point of the opposite phase. This set up mimics that the phase under consideration sees its contact partner as an external pressure
In order to detect free surfaces, the parameter compute_FS must be set to 'YES' for the chamber, where the boundary condition shall be applied.

%BND_inflow%

pressure boundary conditions: inflow condition

```
BC_p ( $...$ ) = ( %BND_inflow% )
```

Applies standard pressure wall boundary condition to the inflow.

$$= \nabla p \cdot n = \rho g \cdot n$$

%BND_slip%

pressure boundary conditions: classical wall

Syntax:

```
BC_p ( $...$ ) = ( %BND_wall% , OPTIONAL: c_div , OPTIONAL: c_NUS)
```

Equation:

$$\frac{\partial p}{\partial n} = \nabla^T p \cdot \mathbf{n} = \rho \left(\hat{\mathbf{g}}^T \cdot \mathbf{n} \right)$$

optional parameters:

- c_div -> currently not used
- c_NUS -> regularize the boundary condition in the sense $\frac{\partial p}{\partial n} = \rho \left(\hat{\mathbf{g}}^T \cdot \mathbf{n} \right) + c_{NUS} \cdot p$

%BND_slip_InContact%

pressure contact boundary conditions for the case the contact phase is the heavy phase

Syntax:

```
BC_p ( $...$ ) = (%BND_slip_InContact%)
```

Equation:

$$\nabla p = \rho g \cdot n$$

where n is the normal of the contact interphase plane. I.e. the point under consideration sees the phase it is in contact with as a wall

%BND_slip_InContact_Explicit%

pressure boundary condition for the case that the contact phase is the heavy phase

Syntax:

```
BC_p ( $...$ ) = (%BND_slip_InContact_Explicit%)
```

Equation:

$$= \nabla p \cdot n = \rho g \cdot n$$

%BND_wall%

pressure boundary conditions: classical wall

Syntax:

```
BC_p ( $...$ ) = ( %BND_wall% , OPTIONAL: c_div , OPTIONAL: c_NUS)
```

Equation:

$$\frac{\partial p}{\partial n} = \nabla^T p \cdot \mathbf{n} = \rho \left(\hat{\mathbf{g}}^T \cdot \mathbf{n} \right)$$

optional parameters:

- c_div -> currently not used
- c_NUS -> regularize the boundary condition in the sense $\frac{\partial p}{\partial n} = \rho \left(\hat{\mathbf{g}}^T \cdot \mathbf{n} \right) + c_{NUS} \cdot p$

%BND_wall_InContact_Explicit%

pressure outflow boundary condition

Syntax:

```
BC_p ( $...$ ) = ( %BND_wall_InContact_Explicit% )
```

Equation:

$$= \nabla p \cdot n = \rho g \cdot n$$

Syntax:

```
BC_p ( $...$ ) = ( %BND_outflow% )
```

Equation:

$$= \nabla p \cdot n = 0$$

%BND_wall_nosl%

pressure boundary conditions: classical wall

Syntax:

```
BC_p ( $...$ ) = ( %BND_wall% , OPTIONAL: c_div , OPTIONAL: c_NUS)
```

Equation:

$$\frac{\partial p}{\partial n} = \nabla^T p \cdot \mathbf{n} = \rho \left(\hat{\mathbf{g}}^T \cdot \mathbf{n} \right)$$

optional parameters:

- c_div -> currently not used
- c_NUS -> regularize the boundary condition in the sense $\frac{\partial p}{\partial n} = \rho \left(\hat{\mathbf{g}}^T \cdot \mathbf{n} \right) + c_{NUS} \cdot p$

BC_v

velocity boundary conditions

```
BC_v ($BCindex$) = ( %BND_inflow% , v_n, v_a, v_b )
BC_v ($BCindex$) = ( %BND_wall% , Parameter )
BC_v ($BCindex$) = ( %BND_wall_nosl% )
BC_v ($BCindex$) = ( %BND_slip% , FrictionCoefficient, ControlThicknessMomentum, vPenetration, uBoundary, vBoundary, wBoundary )
BC_v ($BCindex$) = ( %BCON_Vdot% , Vdot_n, Vdot_a, Vdot_b, BubbleVdot, BubbleRadius, FileNumber )
```

Also the syntax

```
BCON ($BCindex$,%ind_v(1)%) = ( ... )
```

is possible.

references to CODI and CODI_min_max
Make sure to set the ControlThicknessMomentum to ZERO if using EULER , EULERIMPL or EULEREXPL!

List of members:

%BND_wall_InContact_Explicit%	velocity wall boundary condition
%BND_inflow%	inflow velocity boundary condition (Dirichlet type)
%BND_slip%	velocity boundary conditions: slip with viscous friction
%BND_wall%	velocity boundary conditions: pure slip
%BND_wall_nosl%	velocity boundary conditions: pure no-slip
%BND_outflow%	velocity outflow boundary condition
%BND_free%	free surface boundary condition for velocities
%BND_free_InContact%	
%BND_free_InContact_Explicit%	
%BND_free_NoVisc%	non-viscous boundary condition for velocities
%BND_far_field%	far-field velocity boundary condition
%BCON_Mdot%	velocity boundary condition: mass flux
%BCON_Vdot%	velocity boundary conditions: volume flux
%BND_DIRICH%	Dirichlet velocity boundary condition
%BND_NEUMANN%	Neumann velocity boundary condition
%BND_NUSSEL%	Nusselt velocity boundary condition

%BCON_Mdot%

velocity boundary condition: mass flux

BC_v ($BCindex$) = (%BCON_Mdot% , Mdot_n, Area)

%BCON_Mdot% computes the velocity inflow condition based on the given mass flux in normal direction of the boundary element **Mdot_n** .
The size of the inflow **Area** is also required.

%BCON_Vdot%

velocity boundary conditions: volume flux

BC_v ($BCindex$) = (%BCON_Vdot% , Vdot_n, Vdot_a, Vdot_b, BubbleVdot, BubbleRadius, FileNumber)

%BCON_Vdot% computes the velocity inflow conditions based on the given volume fluxes in normal direction of the boundary element **Vdot_n** and
in the tangential directions of the boundary element **Vdot_a** and **Vdot_b** . The parameters **Vdot_n** , **Vdot_a** , and **Vdot_b** are obligatory.

If the parameters **BubbleVdot** and **BubbleRadius** are given, the bubbly inflow algorithm is activated with the *fractional* bubble volume flux
BubbleVdot and the expected bubble radius **RadiusBubble** . The algorithm will create bubbles at random positions at the boundary element with
random size with expectation value RadiusBubble. The positions and sizes of the bubbles are saved in the files BUBBLYINFLOW_Centers00000.dat and BUBBLYINFLOW_Areas00000.dat in the result folder.

If in addition the parameter **FileNumber** is given, MESHFREE expects the files BUBBLYINFLOW_CentersFileNumber.dat and
BUBBLYINFLOW_AreasFilenumber.dat to be present at the path where MESHFREE is executed and reads the sizes as well as positions
of the bubbles from those files instead of creating them randomly. The files have to named according to the following convention:
- FileNumber = 0 ---> BUBBLYINFLOW_Centers00000.dat, BUBBLYINFLOW_Areas00000.dat
- FileNumber = 10 ---> BUBBLYINFLOW_Centers00010.dat, BUBBLYINFLOW_Areas00010.dat
- ...

%BND_DIRICH%

Dirichlet velocity boundary condition

Dirichlet (first-type) boundary condition for velocities. Sets the
velocity at the boundary to a fixed value or user-defined equation:

$$\mathbf{v} = \mathbf{v}_{\text{user}}$$

Syntax:

BC_v (xyz) = (%BND_DIRICH% , , ,)

Example 1: constant inflow

```
BC_v ( $c_inflow$ ) = ( %BND_DIRICH% , 10, 0, 0)
```

Constant inflow of $10\frac{m}{s}$ in x-direction, i.e. parallel to the x-axis. Use %BND_inflow% instead if you want an inflow perpendicular to a wall.

Example 2: inflow with equation

```
BC_v ( $sine_inflow$ ) = ( %BND_DIRICH% , [1.0 + sin(Y%ind_t%)], 0, 0)
```

An alternating inflow in x-direction with speeds $v \in [0, 2]$.

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_v · %BND_NEUMANN%

%BND_NEUMANN%

Neumann velocity boundary condition

Applies a Neumann (second-type) boundary condition for velocities.

$$\frac{\partial v_x}{\partial n} = \alpha, \frac{\partial v_y}{\partial n} = \beta, \frac{\partial v_z}{\partial n} = \gamma$$

Syntax:

```
BC_v ( $xyz$ ) = ( %BND_NEUMANN% , , , )
```

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_v · %BND_NUSSEL%

%BND_NUSSEL%

Nusselt velocity boundary condition

Applies the Nusselt boundary condition for velocities at the boundary:

$$\frac{\partial \vec{v}}{\partial n} = \alpha + \beta \vec{v}$$

Syntax:

```
BC_v ( $xyz$ ) = ( %BND_NUSSEL% , a, b, c, d, e, f)
```

which applies the equations $\frac{\partial v_x}{\partial n} = a + bv_x$, $\frac{\partial v_y}{\partial v_n} = c + dv_y$, $\frac{\partial v_z}{\partial n} = e + fv_z$.

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_v · %BND_far_field%

%BND_far_field%

far-field velocity boundary condition

Applies a Dirichlet (first-type) boundary condition with the current velocity at the boundary. This means that the velocity at the boundary is constant and does not change over time.

Syntax:

```
BC_v ( $xyz$ ) = ( %BND_far_field% )
```

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_v · %BND_free%

%BND_free%

free surface boundary condition for velocities

Syntax:

```
BC_v ( $xyz$ ) = ( %BND_free% )
BC_v ( $xyz$ ) = ( %BND_free% , , , , Thickness, RegularizationParameter),
```

where

- ($S_{n_x}, S_{n_y}, S_{n_z}$) is the stress in normal direction with the normal pointing outwards, (optional, default 0,0,0)
- **Thickness** is the thickness of the control element, (optional, default is 0.0)
- **RegularizationParameter** is a numerical regularization parameter for the $\frac{\partial}{\partial n}$ -operator. (optional, default 0.0)

These parameters are optional! If not set, they are using the default value of 0.

Good to know:

- Make sure to set the Thickness to ZERO if using EULER , EULERIMPL or EULEREXPL!, i.e.

  ```
  BC_v ( $xyz$ ) = ( %BND_free% , , , , 0.0 , RegularizationParameter),
  ```

- In order to detect free surfaces, the parameter compute_FS must be set to 'YES' for the chamber, where the boundary condition shall be applied.

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_v · %BND_free_InContact%

%BND_free_InContact%

Syntax:

```
BC_v ( $xyz$ ) = ( %BND_free_InContact% )
```

Good to know:

- Additionally, the same optional parameters as for %BND_free% are available.
- In order to detect free surfaces, the parameter compute_FS must be set to 'YES' for the chamber, where the boundary condition shall be applied.

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_v · %BND_free_InContact_Explicit%

%BND_free_InContact_Explicit%

Syntax:

```
BC_v ( $xyz$ ) = ( %BND_free_InContact_Explicit% )
```

Good to know:

- Additionally, the same optional parameters as for %BND_free% are available.
- In order to detect free surfaces, the parameter compute_FS must be set to 'YES' for the chamber, where the boundary condition shall be applied.

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_v · %BND_free_NoVisc%

%BND_free_NoVisc%

non-viscous boundary condition for velocities

Applies a Dirichlet (first-type) boundary condition which is independent of any viscous shear stresses:

$$\mathbf{v}_{\text{boundary}} = \mathbf{v} + \Delta t \mathbf{g} - \Delta t \frac{\mathbf{n}_p}{\rho_{sm}}$$

with gravity g , n_p the gradient of the pressure in normal direction

Syntax:

```
BC_v ( $xyz$ ) = ( %BND_free_NoVisc% )
```

In order to detect free surfaces, the parameter compute_FS must be set to 'YES' for the chamber, where the boundary condition shall be applied.

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_v · %BND_inflow%

%BND_inflow%

inflow velocity boundary condition (Dirichlet type)

Normal Velocity

%BND_inflow% defines a boundary condition of Dirichlet type in normal direction. The syntax for defining the velocity $v_{\text{in}}^{\vec{n}}$ in normal direction at an inflow boundary is:

```
BC_v ($xyz$) = ( %BND_inflow% , v_in^n )
```

Example 1: Inflow with velocity of $10 \frac{m}{s}$ normal to boundary elements with BC -flag $inflow$.

```
BC_v ( $inflow$ ) = ( %BND_inflow% , 10)
```

Good to know:

- The boundary condition is relative to a movement of the boundary element, in particular: if the inflow boundary element is moving, the resulting total velocity will be the sum of the velocity of the movement plus the normal velocity at the inflow.
- The normal vector $\vec{n}$ points to the inside.
- This boundary condition for the velocity is of Dirichlet type as the velocity is explicitely prescribed. Hence the boundary condition for the hydrostatic and dynamic pressure should be of Neumann type e.g.:

```
BC_p ( $inflow$ ) = ( %BND_wall% )
BCON ( $inflow$ ,%ind_p_dyn%) = ( %BND_AVERAGE% , 0) # a lower order Neumann type condition
```

Special Case: Tangential Components

The statement %BND_inflow% with one parameter defines the velocity in normal direction. Sometimes it is also necessary to have tangential components in the inflow boundary conditions, e.g. for modeling realistic inflow behavior in filling processes. This can be done by specifying two further parameters.
Let $\vec{n}$ be the normal on the boundary element. Then two tangential vectors (non-unique!) $\vec{a}, \vec{b}$ can be found such that $\vec{n}, \vec{a}, \vec{b}$ are all perpendicular to each other. Then also velocities $v_{\text{in}}^{\vec{n}}, v_{\text{in}}^{\vec{a}}, v_{\text{in}}^{\vec{b}}$ for each of these directions can be prescribed. Syntax:

```
BC_v ($xyz$) = ( %BND_inflow% , v_in^n , v_in^a , v_in^b )
```

Example 2: Add a random fluctuation of tangential velocities at the inflow. Total order of magnitude of these velocities is around 7 percent of the inflow velocity:

```
begin_alias{ }
"v_in" = " 10.0 " # normal inflow velocity
"InflowFluctuations" = " 0.07 " # magnitude of fluctuations relative to normal velocity
end_alias
BC_v ( $inflow$ ) = ( %BND_inflow% , &v_in& , ... # normal inflow velocity
[ &v_in& * rand(- &InflowFluctuations& ) ], ... # velocity component in tangential direction a
[ &v_in& * rand(- &InflowFluctuations& ) ]) # velocity component in tangential direction b
```

Good to know:

- For the special case of a filling with perturbation it is ok, that the $\vec{a}, \vec{b}$ are not uniquely defined, because we want to model a random behavior there and for that it is only important that the tangential vectors are perpendicular.
- MESHFREE issues a warning if it is detected that tangential velocities are prescribed by the user, because in most cases, this is not what the user intended to do.

%BND_outflow%

velocity outflow boundary condition

This boundary condition adapts based on the major flow direction near the outflow boundary.

If the relative velocity of the MESHFREE point to the boundary is pointing outwards, we assume a Neumann (second-type) boundary condition, i.e. %BND_NEUMANN% with $\frac{\partial v}{\partial n} = 0$.

If, however, the relative velocity is pointing inwards, the boundary condition is identical to %BND_wall% .

Syntax:

```
BC_v ($xyz$) = ( %BND_outflow% )
```

%BND_slip%

velocity boundary conditions: slip with viscous friction

BC_v ($BCindex$) = (%BND_slip% , FrictionCoefficient, ControlThicknessMomentum, vPenetration, uBoundary, vBoundary, wBoundary)

FrictionCoefficient
Viscous friction in the sense

$$\boldsymbol{S}\left(\boldsymbol{v}^{n+1}\right)\cdot\boldsymbol{a}=\alpha\cdot\left(\boldsymbol{v}^{n+1}-\boldsymbol{v}_0\right)\cdot\boldsymbol{a}.$$

Here, α is the FrictionCoefficient, $\alpha=0$ would lead to pure slip, $\alpha\to\infty$ would lead to pure no-slip. If the turbulence model is in action, the effective friction coefficient is given by $\alpha_{eff}=\alpha_{turb}+\alpha$, where $\alpha_{turb}=\alpha_{turb}(k,\epsilon)$.

ControlThicknessMomentum
Incorporation of the momentum balance into the boundary condition, especially important for big Re-numbers. The thickness of the momentum control cell is ControlThicknessMomentum*H (smoothing length).

For current scientific reasons: by putting a minus in front of ControlThicknessMomentum, a special tear-off criterion is launched. In fact, an additional component is locally added to the gradient of pressure in tangential direction, if

- the point is marked as tear-off-point (see %ind_TearOff%).
- the point is in a local suction regime (pressure decreases from the free surface towards the interior).

This additional pressure component might provoke tear-off, as it forces the tear-off-point to move away from the free surface.

Make sure to set the ControlThicknessMomentum to ZERO if using EULER , EULERIMPL or EULEREXPL!

vPenetration
Force the normal component of the flow to penetrate through the wall, i.e.

$$v_{\text{Penetration}}=\left(\boldsymbol{v}^{n+1}-\boldsymbol{v}_0\right)\cdot\boldsymbol{n}.$$

{u,v,w}Boundary
Usually, MESHFREE checks the appropriate wall and applies the movement of this wall as the basis wall velocity $\boldsymbol{v}_0$. Optionally, the user is able to redefine the components of the velocity of the wall movement by {u,v,w}Boundary. Note however, that turbulence effects on α due to this movement are neglected in this case.

An alternative approach for simulations with turbulence is to instead directly define the relative wall movement via an EVENT . This works essentially in the same way as %MOVE_VirtualRotation% in %MOVE_TranslationRotation% .

Example:

```
EVENT (1) = ( [binA("MovedWallAlias")], %EVENT_FunctionManipulation% , ...
%ind_v_p(1)% , [Y %ind_v_p(1)% + &relativeWallMovementX& ], ...
%ind_v_p(2)% , [Y %ind_v_p(2)% + &relativeWallMovementY& ], ...
%ind_v_p(3)% , [Y %ind_v_p(3)% + &relativeWallMovementZ& ])
```


%BND_wall%
velocity boundary conditions: pure slip

BC_v ($BCindex$) = (%BND_wall%)

MESHFREE determines the velocity of the boundary element the point is attached to.
This velocity is a Dirichlet condition on the normal component of the velocity and a Neumann condition on the tangential components.

As an option, we can set:

```
BC_v ( $BCindex$ ) = ( %BND_wall% , 1 )
```

In this case, MESHFREE tries to interpolate the velocities from the neighborhood of the given point from the previous time step.
In this case the advantage is that only Dirichlet conditions are set forth to the velocity (much better conditioning of the linear system).
The disadvantage is that it is an explicit boundary condition within an implicit numerical framework.

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_v · %BND_wall_InContact_Explicit%

%BND_wall_InContact_Explicit%

velocity wall boundary condition

Same as %BND_wall% .

MESHFREE · InputFiles · USER_common_variables · BoundaryConditions · LIQUID__BC__ · BC_v · %BND_wall_nosl%

%BND_wall_nosl%

velocity boundary conditions: pure no-slip

```
BC_v ( $BCindex$ ) = ( %BND_wall_nosl% )
```

MESHFREE determines the velocity of the boundary element the point is attached to.
This velocity is prescribed as a Dirichlet boundary condition to the MESHFREE point.

MESHFREE · InputFiles · USER_common_variables · BoundaryElements

3.1.6. BoundaryElements

definition of the boundary elements to be used during simulation

The boundary element section is embedded in the following structure:

```
begin_boundary_elements{ }
...
end_boundary_elements
```

Options:

1.) Read in geometry files by the include clause (include{ File}).
2.) Define points, planes, lines, triangles, and simple bodies like cylinders and cubes (PlainBoundaryElements).

Every boundary needs an alias which describes its behavior, e.g. connects it to boundary conditions and movement. This is described in AliasForGeometryItems .

Sometimes, it is also necessary to make GeometryManipulations dependent on previously read or defined boundary elements.

With the ConstructClause , there is the chance to construct scalars or vectors that can be used to manipulate geometries. In that aspect, it is necessary to keep a certain sequence: read in a subset of files, establish a sequence of the ConstructClause items, and
apply the results of the ConstructClause items in a subsequent read-of-file.

Example:

```
begin_boundary_elements{ }
...
include{ FileNameA} # contains (at least) geometry part inflow
...
end_boundary_elements
begin_construct{ }
"xMeanInflow" = CONSTRUCT ( %CONSTRUCT_BoxMidPoint% , 0.5, "inflow" ) # mid point of geometry part inflow
end_construct
begin_boundary_elements{ }
...
include{ FileNameB} offset{ &xMeanInflow& } # contains other geometry parts
...
end_boundary_elements
```

Note: begin_boundary_elements and begin_construct blocks are read sequentially.
All geometry parts used in a construct statement need to be defined beforehand.

List of members:

include{	definition of a geometry file to be read by MESHFREE
CuttingCurveCluster	define clusters of boundary elements by cutting the geometry along given curves
manipulate{	manipulate (move, rotate, ...) the geometry belonging to an alias-group
delete{	delete all the geometry belonging to a given alias-group
CreateBEfromGeometry	from the already existing geometry, create new boundary elements
ConstructClause	mathematical construction of scalars and vectors
PlainBoundaryElements	definition of a plain geometry directly in MESHFREE

ConstructClause

mathematical construction of scalars and vectors

Construct statements offer the possibility to automatically construct quantities (Examples: centre of gravity, bounding box) for GeometryManipulations like offset{ , scale{ , rotate{ ... based on a geometry read in. The construct statements are evaluated in the startup phase together with the reading of the geometry files.

A construct statement may look as follows:

```
begin_boundary_elements{ }
include{ FileName} offset{ CONSTRUCT (%CONSTRUCT_...%, , , , ) }
end_boundary_elements
```

Construct Environments

If the construct result is to be used at several occasions, it is worthwhile putting it into a construct environment:

```
begin_construct{ }
"xMeanInflow" = CONSTRUCT ( %CONSTRUCT_...%, , , , )
end_construct
```

It can then be referenced by the name given on the left hand side in the same way as an ALIAS . See also Variables .

The begin_construct{ environment is only evaluated at regular startup, not if a restart is performed. Hence, for construct results that are supposed to be computed during restart, use

```
begin_construct_atRestart{ }
...
end_construct_atRestart {}
```

These are not read during normal initialization. Thus, to update values, you need to use both.

Examples

Example 1:

```
begin_boundary_elements{ }
...
include{ FileNameA} # contains (at least) geometry part inflow
...
end_boundary_elements
begin_construct{ }
"xMeanInflow" = CONSTRUCT ( %CONSTRUCT_BoxMidPoint% , 0.5, "inflow" ) # mid point of geometry part inflow
end_construct
begin_boundary_elements{ }
...
include{ FileNameB} offset{ &xMeanInflow& } # contains other geometry parts
include{ FileNameC} offset{ &xMeanInflow(1)& , &xMeanInflow(2)& , &xMeanInflow(3)& } # contains other geometry
parts
...
end_boundary_elements
```

In addition, any RightHandSideExpression can be used to save the result of a calculation during the initialisation phase into an alias, for example to create new boundary elements from scratch after reading in other geometry.

Example 2:

```
begin_construct{ }
"Nnode" = "real(%BND_count_NP%)"
end_construct
begin_boundary_elements{ }
BND_node [ &Nnode& +1] [0] [0] [0]
BND_node [ &Nnode& +2] [0] [0] [1]
BND_node [ &Nnode& +3] [0] [1] [1]
BND_node [ &Nnode& +4] [1] [0] [0]

BND_tria &inflow& [ &Nnode& +1] [ &Nnode& +2] [ &Nnode& +3]
BND_tria &inflow& [ &Nnode& +3] [ &Nnode& +4] [ &Nnode& +1]
end_boundary_elements
```

Note:

- begin_boundary_elements{ and begin_construct{ blocks are read sequentially.
  All geometry parts used in a construct statement need to be defined beforehand.
  Values that are saved into an alias stay constant throughout the simulation,
  irrespective of for example geometry movements.
- By default, CONSTRUCT -aliases are not recomputed on RESTART . If recomputation is desired,
  the begin_construct_atRestart{ -functionality has to be used.

The possible CONSTRUCT -keywords can be found below.

List of members:	
%CONSTRUCT_Area%	area of given alias-entities
%CONSTRUCT_BoxMax%	maximum of enclosing box around given alias-entities
%CONSTRUCT_BoxMidPoint%	mid point of enclosing box around given alias-entities
%CONSTRUCT_BoxMin%	minimum of enclosing box around given alias-entities
%CONSTRUCT_COG%	center of gravity for given alias-entities
%CONSTRUCT_EstablishCurveVolumeVersusHeight%	establish a 2-row-curve that provides the height-volume-relation of a closed part of geometry
%CONSTRUCT_Normal%	normal with respect to given alias-entities
%CONSTRUCT_NormalDividedByArea%	area-averaged normal with respect to given alias-entities
%CONSTRUCT_PointBasedOnAbsoluteVolume%	Computes a point that defines a given volume inside a closed structure
%CONSTRUCT_PointBasedOnRelativeVolume%	compute a point that defines a given volume inside a closed structure
%CONSTRUCT_Tangent1%	first tangent with respect to given normal vector and alias-entities
%CONSTRUCT_Tangent2%	second tangent with respect to given normal vector and alias-entities
%CONSTRUCT_Volume%	volume of a (necessarily) closed geometrical part
%CONSTRUCT_VolumeForGivenHeight%	compute the volume of a closed body restricted by a certain height
%CONVERT_TO_INTEGER%	convert a set of construct variables to integer

%CONSTRUCT_Area%

area of given alias-entities

```
CONSTRUCT ( %CONSTRUCT_Area% , "alias1", "alias2", ... )
```

Determines the area of the geometry elements belonging to the given list of aliases.

%CONSTRUCT_BoxMax%

maximum of enclosing box around given alias-entities

```
CONSTRUCT ( %CONSTRUCT_BoxMax% , "alias1", "alias2", ... )
```

Constructs an enclosing box around the geometry elements belonging to the given list of aliases.
The maximum of the enclosing box in x-, y-, and z-direction is computed.

%CONSTRUCT_BoxMidPoint%

mid point of enclosing box around given alias-entities

```
CONSTRUCT ( %CONSTRUCT_BoxMidPoint% , RelativePosition, "alias1", "alias2", ... )
```

Constructs an enclosing box around the geometry elements belonging to the given list of aliases.

RelativePosition:

- 0 will return the lower left corner of this box
- 1 will return the upper right corner of this box
- 0.5 will return the box mid point

Any value is allowed for RelativePosition.

OPTIONAL PARAMETER:

```
CONSTRUCT ( %CONSTRUCT_BoxMidPoint% , %CONSTRUCT_IncludeIGESfaces% , RelativePosition, "alias1",
"alias2", ... )
```

If this oprional parameter is set, then MESHFREE will inlcude IGES faces in the measurement of the enclosing boxes.

%CONSTRUCT_BoxMin%

minimum of enclosing box around given alias-entities

```
CONSTRUCT ( %CONSTRUCT_BoxMin% , "alias1", "alias2", ... )
```

Constructs an enclosing box around the geometry elements belonging to the given list of aliases.
The minimum of the enclosing box in x-, y-, and z-direction is computed.

%CONSTRUCT_COG%

center of gravity for given alias-entities

```
CONSTRUCT ( %CONSTRUCT_COG% , "alias1", "alias2", ... )
```

Determines the center of gravity for the geometry elements belonging to the given list of aliases.

%CONSTRUCT_EstablishCurveVolumeVersusHeight%

establish a 2-row-curve that provides the height-volume-relation of a closed part of geometry

```
begin_construct{ }
"Curve" = CONSTRUCT ( %CONSTRUCT_EstablishCurveVolumeVersusHeight% , nRef_x, nRef_y, nRef_z, pRef_x, pRef_y, pRef_z, nTicks, "alias1", "alias2", ...)
end_construct
begin_curve{ $CurveName$}
&Curve&
end_curve
```

The text item "Curve" is really a curve in the MESHFREE -sense, i.e. it will contain carriage-return and line-feed characters,
such that it can be used in a curve definition.

%CONSTRUCT_Normal%

normal with respect to given alias-entities

```
CONSTRUCT ( %CONSTRUCT_Normal% , "alias1", "alias2", ... )
```

Determines the normal with respect to the geometry elements belonging to the given list of aliases.

%CONSTRUCT_NormalDividedByArea%

area-averaged normal with respect to given alias-entities

```
CONSTRUCT ( %CONSTRUCT_NormalDividedByArea% , "alias1", "alias2", ... )
```

Determines the area-averaged normal with respect to the geometry elements belonging to the given list of aliases.

%CONSTRUCT_PointBasedOnAbsoluteVolume%

Computes a point that defines a given volume inside a closed structure

Given a closed geometry (such as a tank) by a list of ALIAS names, this functionally places a point on a given axis. The point and the given axis describe a plane. The plane shall subdivide the closed structure such that the required absolute volume is below the plane.

```
begin_construct{ }
"x_Reference" = CONSTRUCT ( %CONSTRUCT_PointBasedOnAbsoluteVolume% , axis_x, axis_y, axis_z, absoluteVolume, "alias1", "alias2", ...)
end_construct
```

(axis_x, axis_y, axis_z) describe the axis that defines the normal direction of the (cutting) plane
absoluteVolume is the absolute volume required by the cutting plane, hence the unit is m^3

Remarks:

- The subroutine cuts the given geometry by the described plane, and calculates the volume in the shape below the plane
- We use the principal that total volume of a 3D shape is equal to the net flux at its surface
  We cut the mesh with a plane, use the resulting closed geometry below the plane.

DOWNLOAD COMPREHENSIVE EXAMPLE

%CONSTRUCT_PointBasedOnRelativeVolume%

compute a point that defines a given volume inside a closed structure

Given a closed geometry (such as a tank) by a list of ALIAS names, this functionally places a point on a given axis. The point and the given axis describe a plane. The plane shall subdivide the closed structure such that the required relative volume (based on the structures total volume) is below the plane.

```
begin_construct{ }
"x_Reference" = CONSTRUCT ( %CONSTRUCT_PointBasedOnRelativeVolume% , axis_x, axis_y, axis_z,
relativeVolume, "alias1", "alias2", ... )
end_construct
```

(axis_x, axis_y, axis_z) describe the axis that defines the normal direction of the (cutting) plane
relativeVolume is the relative volume required by the cutting plane, hence to be kept between 0 and 1

Remarks:

- This function converts relative volume into absolute volume by multiplying relative volume value into total volume, then behaves exactly like %CONSTRUCT_PointBasedOnAbsoluteVolume%

%CONSTRUCT_Tangent1%

first tangent with respect to given normal vector and alias-entities

```
CONSTRUCT ( %CONSTRUCT_Tangent1% , nRef_x, nRef_y, nRef_z, "alias1", "alias2", ... )
```

Determines the first tangent $\mathbf{t}_1$ with respect to the given normal $\mathbf{n}_{\text{Ref}}$ = (**nRef_x** , **nRef_y** , **nRef_z**)
and the normal $\mathbf{n}_{\text{alias1,alias2,...}}$ of the geometry elements belonging to the given list of aliases in the following sense:

$$\mathbf{t}_1 = \frac{\mathbf{n}_{\text{Ref}} \times \mathbf{n}_{\text{alias1,alias2,...}}}{\|\mathbf{n}_{\text{Ref}} \times \mathbf{n}_{\text{alias1,alias2,...}}\|_2}$$

%CONSTRUCT_Tangent2%

second tangent with respect to given normal vector and alias-entities

CONSTRUCT (%CONSTRUCT_Tangent2% , nRef_x, nRef_y, nRef_z, "alias1", "alias2", ...)

Determines the second tangent $\mathbf{t}_2$ with respect to the given normal $\mathbf{n}_{\text{Ref}}$ = (**nRef_x** , **nRef_y** , **nRef_z**) and the normal $\mathbf{n}_{\text{alias1,alias2,...}}$ of the geometry elements belonging to the given list of aliases in the following sense:

$$\mathbf{t}_2 = \frac{\mathbf{n}_{\text{Ref}} \times \mathbf{t}_1}{\|\mathbf{n}_{\text{Ref}} \times \mathbf{t}_1\|_2},$$

where $\mathbf{t}_1$ is the first tagent given by

$$\mathbf{t}_1 = \frac{\mathbf{n}_{\text{Ref}} \times \mathbf{n}_{\text{alias1,alias2,...}}}{\|\mathbf{n}_{\text{Ref}} \times \mathbf{n}_{\text{alias1,alias2,...}}\|_2}$$

See also %CONSTRUCT_Tangent1% .

MESHFREE · InputFiles · USER_common_variables · BoundaryElements · ConstructClause · %CONSTRUCT_Volume%

%CONSTRUCT_Volume%

volume of a (necessarily) closed geometrical part

Given a closed geometry by a list of ALIAS names, this functionality computes the internal volume of the geometry.

```
begin_construct{ }
"volume" = CONSTRUCT ( %CONSTRUCT_Volume% , "alias1", "alias2", ...)
end_construct
```

Remarks:

- See %CONSTRUCT_PointBasedOnAbsoluteVolume%

MESHFREE · InputFiles · USER_common_variables · BoundaryElements · ConstructClause · %CONSTRUCT_VolumeForGivenHeight%

%CONSTRUCT_VolumeForGivenHeight%

compute the volume of a closed body restricted by a certain height

For a closed geometry, defined by a list of ALIAS names, compute the volume that turns out due to a given filling height.

```
begin_construct{ }
"VolumeVariable" = CONSTRUCT ( %CONSTRUCT_VolumeForGivenHeight% , nRef_x, nRef_y, nRef_z, pRef_x,
pRef_y, pRef_z, height, "alias1", "alias2", ...)
end_construct
```

(nRef_x, nRef_y, nRef_z,) is the reference direction
(pRef_x, pRef_y, pRef_z,) is the reference point
height is the filling level of the closed structure above the reference point, in the direction of the reference direction.

This functionality is the inverse operation of %CONSTRUCT_PointBasedOnRelativeVolume% and %CONSTRUCT_PointBasedOnAbsoluteVolume% .

MESHFREE · InputFiles · USER_common_variables · BoundaryElements · ConstructClause · %CONVERT_TO_INTEGER%

%CONVERT_TO_INTEGER%

convert a set of construct variables to integer

```
CONSTRUCT ( %CONVERT_TO_INTEGER% , N , "constructVariable1", "constructVariable2", ... )
```

N : if N=0 -> normal integer conversion; if N>0, fill leading zeros such that total length of integer is **N**

CreateBEfromGeometry

from the already existing geometry, create new boundary elements

With this statement the user is able to create new boundary elements from already existing geometry.

A create statement has to be embedded in the boundary element environment, i.e.

```
begin_boundary_elements{ }
...
CreateStatement comes here
...
end_boundary_elements
```

For details see the options below.

List of members:

BNDpoints_ExtractFromNodes{	create BND_points from existing geometry nodes

BNDpoints_ExtractFromNodes{

create BND_points from existing geometry nodes

```
begin_boundary_elements{ }
...
BNDpoints_ExtractFromNodes{ [Eqn], "AliasTheNewPointsAreSupposedToTake",
"FirstAliasOfGeometryTheNodesAreTakenFrom", "SecondAliasOfGeometryTheNodesAreTakenFrom", ... }
...
end_boundary_elements
```

M E S H F R E E goes through all existing boundary elements whose alias is one of **FirstAliasOfGeometryTheNodesAreTakenFrom** , **SecondAliasOfGeometryTheNodesAreTakenFrom** , ...
From their nodes, new elements of B N D _ p o i n t are created which take the alias **AliasTheNewPointsAreSupposedToTake** .

The aliases have to exist, i.e.

```
begin_alias{ }
...
"AliasTheNewPointsAreSupposedToTake" = " ... " # alias for new points
"FirstAliasOfGeometryTheNodesAreTakenFrom" = " ... " # first original alias
"SecondAliasOfGeometryTheNodesAreTakenFrom" = " ... " # second original alias
...
end_alias
```

CuttingCurveCluster

define clusters of boundary elements by cutting the geometry along given curves

To cut the boundary geometry by cutting curves given in the IGES file FileName
and to determine the CuttingCurveCluster IDs, use:

```
include_CCC_curves{ FileName}
```

The IDs can then be used, for example in Equations and INTEGRATION statements,
via the functions CID() and isCID() .

Example:

```
SAVE_ITEM = ( %SAVE_scalar%, [CID(0)], "CCCID" )
```

See below for further optional parameters that can be set.

List of members:

include_CCC_curves	define the geometry file containing cutting curves for clustering
CCC_maxSegmentLength	maximum segment length for linearization of cutting curves (optional)
CCC_minNewEdgeLength	minimum absolute length for new triangle edges (optional)
CCC_relativeEdgeLength	minimum relative length for new triangle edges (optional)
CCC_CuttingDistance	distance up to which boundary element nodes are considered to lie on a cutting curve (optional)
CCC_clusterAllTriangles	flag whether or not to determine clusters without given starting points (optional)
CCC_seeds	seeds starting points for CuttingCurveCluster (optional)

CCC_CuttingDistance

distance up to which boundary element nodes are considered to lie on a cutting curve (optional)

This parameter is used to determine initial CuttingCurveCluster , which are then increased up to the cutting curves.
In some cases, changing this value can improve the accuracy of the clustering algorithm.

Example:

```
CCC_CuttingDistance = 2.0
```

If it is not given or if the value is not greater than zero, a default value will be computed
from the characteristics of the geometry triangulization and cutting curve linearization.

If the clustering algorithm detects that several cluster starting points define the same cluster,
then it will automatically try to make them unique by increasing this parameter.

See also CCC_seeds .

CCC_clusterAllTriangles

flag whether or not to determine clusters without given starting points (optional)

If set to true, all boundary triangles will be assigned to clusters, irrespective of whether or not that cluster can be reached from any of the CCC_seeds .

Examples:

```
CCC_clusterAllTriangles = 0
CCC_clusterAllTriangles = 1
```

If it is not given or if the value is invalid, the following defaults will be used:

- 0 if at least one cluster starting point is given,
- 1 if no cluster starting point is given.

See also CCC_seeds .

CCC_maxSegmentLength

maximum segment length for linearization of cutting curves (optional)

Set this parameter to define the maximum segment length for the linearization of the cutting curves used for CuttingCurveCluster .

Example:

```
CCC_maxSegmentLength = 0.01
```

This parameter is optional. If it is not given or if the value is not greater than zero, a default will be computed from the characteristics of the geometry triangulization.

CCC_minNewEdgeLength

minimum absolute length for new triangle edges (optional)

Set this parameter to define the minimum length for new triangle edges when cutting the geometry along cutting curves to determine CuttingCurveCluster .

Example:

```
CCC_minNewEdgeLength = 0.001
```

This parameter is optional. If it is not given or if the value is not greater than zero, a default will be computed from the characteristics of the geometry triangulization.

CCC_relativeEdgeLength

minimum relative length for new triangle edges (optional)

Set this parameter to a minimum relative tolerance (between 0 and 0.5) for cutting triangle edges when cutting the geometry along cutting curves to determine CuttingCurveCluster .

Example:

```
CCC_minNewEdgeLength = 0.1
```

An edge will not be cut if either of the new edges would be shorter than
$\mathrm{CCC\_minNewEdgeLength} \cdot (\mathrm{old\ edge\ length})$.
If the parameter is not given or if the value is not greater than zero, a default value will be set.

CCC_seeds

seeds starting points for CuttingCurveCluster (optional)

Seeds are points near boundary elements or rays pointing into the geometry that are used to assign certain IDs to specific CuttingCurveCluster , so that these clusters can be addressed in INTEGRATION statements.

The cluster IDs are defined in the order in which the different types of CCC_seeds appear in USER_common_variables .

The different ways in which seeds can be defined are given below.

See also CCC_clusterAllTriangles .

List of members:

begin_CCC_seeds2D	add 2D seeds for CuttingCurveCluster (optional)
begin_CCC_seeds3D	add 3D seeds for CuttingCurveCluster (optional)
begin_CCC_seeds6D	add 6D seeds for CuttingCurveCluster (optional)
include_CCC_seeds2D	include 2D seeds for CuttingCurveCluster from file (optional)
include_CCC_seeds3D	include 3D seeds for CuttingCurveCluster from file (optional)
include_CCC_seeds6D	include 6D seeds for CuttingCurveCluster from file (optional)

begin_CCC_seeds2D

add 2D seeds for CuttingCurveCluster (optional)

Seeds are used to specify the IDs of certain CuttingCurveCluster .
A 2D seed defines a point on a 2D plane at one face of the box enveloping the cutting curves.

```
begin_CCC_seeds2D {dim, end}
x y
...
end_CCC_seeds2D{}
```

The parameter **dim** is an integer (1,2,3) which determines the dimension held constant in the plane.

The parameter **end** is either 'min' or 'max' and determines whether the minimal or maximal value of the enclosing box for that dimension is supposed to be used.

The first triangle that is hit by the ray starting in the determined point and directed perpendicular to the plane into the box is used to seed the cluster.

Example:

```
begin_CCC_seeds2D {3, min}
1.0 2.0
end_CCC_seeds2D{}

begin_CCC_seeds2D {1, max}
10.0 0.0
end_CCC_seeds2D{}
```

The cluster IDs are defined in the order in which the different types of CCC_seeds appear in USER_common_variables .

See also CCC_seeds .

---

begin_CCC_seeds3D

add 3D seeds for CuttingCurveCluster (optional)

Seeds are used to specify the IDs of certain CuttingCurveCluster .
A 3D seed defines a point in the coordinate system of the cutting curves.

```
begin_CCC_seeds3D {}
x y z
...
end_CCC_seeds3D{}
```

The nearest boundary triangle to the point (**x** , **y** , **z**) is used to build the cluster.

The cluster IDs are defined in the order in which the different types of CCC_seeds appear in USER_common_variables .

See also CCC_seeds .

---

begin_CCC_seeds6D

add 6D seeds for CuttingCurveCluster (optional)

Seeds are used to specify the IDs of certain CuttingCurveCluster .
A 6D seed defines a point and a direction from that point towards the geometry in the coordinate system of the cutting curves.

```
begin_CCC_seeds6D {}
x y z dx dy dz
...
end_CCC_seeds6D{}
```

The first triangle that is hit by the ray starting in point (**x** , **y** , **z**) and going into direction (**dx** , **dy** , **dz**) is used to build the cluster.

The cluster IDs are defined in the order in which the different types of CCC_seeds appear in USER_common_variables .

See also CCC_seeds .

---

include_CCC_seeds2D

include 2D seeds for CuttingCurveCluster from file (optional)

The command

```
include_CCC_seeds2D {dim, end, FileName}
```

is equivalent to

```
begin_CCC_seeds2D {dim, end}
[contents of file FileName]
end_CCC_seeds2D{}
```

See also CCC_seeds .

---

include_CCC_seeds3D

include 3D seeds for CuttingCurveCluster from file (optional)

The command

```
include_CCC_seeds3D {FileName}
```

is equivalent to

```
begin_CCC_seeds3D {}
[contents of file FileName]
end_CCC_seeds3D{}
```

See also CCC_seeds .

---

include_CCC_seeds6D

include 6D seeds for CuttingCurveCluster from file (optional)

The command

```
include_CCC_seeds6D {FileName}
```

is equivalent to

```
begin_CCC_seeds6D {}
[contents of file FileName]
end_CCC_seeds6D{}
```

See also CCC_seeds .

MESHFREE · InputFiles · USER_common_variables · BoundaryElements · CuttingCurveCluster · include_CCC_curves

include_CCC_curves

define the geometry file containing cutting curves for clustering

To add the geometry file containing cutting curves and determine the CuttingCurveCluster IDs, use:

```
include_CCC_curves{ FileName}
```

So far, only a single IGES file can be included. Non-curve elements in the file will be ignored.

MESHFREE · InputFiles · USER_common_variables · BoundaryElements · PlainBoundaryElements

PlainBoundaryElements

definition of a plain geometry directly in MESHFREE

Some boundary entities can be defined manually via

```
begin_boundary_elements{ }
...
BND_entity &AliasName& coordinates GeometryManipulations
...
end_boundary_elements
```

Alternatively, the alias definition as described in AliasForGeometryItems can be written directly instead of referencing an AliasName.

For the possible choices of **BND_entity** see below.

List of members:

BND_cube	create an independent rectangular cuboid (box)
BND_cylinder	create a cylinder
BND_disk	create a disk
BND_line	create an independent line
BND_node	create an independent node for use in other boundary entity definitions
BND_plane	
BND_point	create an independent point
BND_quad	create an independent quadrilateral
BND_tria	create an independent triangle
BND_tria6N	create an independent 6-node triangle

BND_cube

create an independent rectangular cuboid (box)

A rectangular cuboid (box) with edges parallel to the axes is defined by the coordinates of two opposite corners

```
BND_cube &AliasName& x1 y1 z1 x2 y2 z2 GeometryManipulations
```


BND_cylinder

create a cylinder

A cylinder can be defined by:

```
BND_cylinder &aliasDefinition& x0 y0 z0 nx ny nz height radiusA radiusB OPTIONAL:NumberOfSegmentsInCircle
```

The cylinder is given by the point (**x0** , **y0** , **z0**), the direction of the axis (**nx** , **ny** , **nz**), the height, and the two radius at the bottom (**radiusA**) and the top (**radiusB**). Hence, even a truncated cone is possible.

NumberOfSegmentsInCircle defines the number of discretization ticks for the circle.

BND_disk

create a disk

A disk can be defined by:

```
BND_disk &aliasDefinition& x0 y0 z0 nx ny nz radius OPTIONAL:NumberOfSegmentsInCircle
```

The disk is given by the center point (**x0** , **y0** , **z0**), the direction of the axis (**nx** , **ny** , **nz**), and the radius.

NumberOfSegmentsInCircle defines the number of discretization ticks for the circle. By default is 51.

BND_line

create an independent line

An independent line can be defined by

- the (initial) coordinates (**x1** , **y1** , **z1**) and (**x2** , **y2** , **z2**) for the starting and ending point of the line, respectively:

  ```
  BND_line &AliasName& x1 y1 z1 x2 y2 z2 GeometryManipulations
  ```

- the node indices **ip1** and **ip2** for the starting and ending point of the line, respectively, of already existing nodes:

  ```
  BND_line &AliasName& ip1 ip2 GeometryManipulations
  ```

If the line is defined by coordinates, MESHFREE automatically creates new node points as a basis.

BND_node

create an independent node for use in other boundary entity definitions

An independent node, which can be used in definitions of boundary entity definitions
is defined by an optional NodeIndex and its coordinates.

```
BND_node OPTIONAL:NodeIndex xi yi zi
```


BND_plane

A plane can be defined by the coordinates of a point on the plane (**x0** , **y0** , **z0**)
and the direction of its normal (**nx** , **ny** , **nz**):

```
BND_plane &aliasDefinition& x0 y0 z0 nx ny nz
```

Such planes can be used for the following tasks:

- Define flat initial free surfaces, see example SimpleBox .
- Cut off points once they pass the plane, see %BND_cut% .
- Use as feeder or cutter, see example SimpleBoxFeederCutter.

BND_point

create an independent point

An independent point can be defined by

- its (initial) coordinate (**x** , **y** , **z**):

  ```
  BND_point &AliasName& x y z GeometryManipulations
  ```

- the node index **ip** of an already existing node:

  ```
  BND_point &AliasName& ip GeometryManipulations
  ```

If it is defined by its coordinates, MESHFREE automatically creates a new node point as a basis.

A BND_point can be used to trigger:

- SMOOTH_LENGTH definitions
- INTEGRATION -statements using values at this point, e.g. %POINT_APPROXIMATE%

BND_quad

create an independent quadrilateral

An independent quadrilateral can be defined by

- the (initial) coordinates (x1,y1,z1), (x2,y2,z2), (x3,y3,z3) and (x4,y4,z4) of its corners.

```
BND_quad &AliasName& x1 y1 z1 x2 y2 z2 x3 y3 z3 x4 y4 z4 GeometryManipulations
```

- the node indices ip1, ip2, ip3 and ip4 for the corners of the quadrilateral:

```
BND_quad &AliasName& ip1 ip2 ip3 ip4 GeometryManipulations
```

If the quadrilateral is defined by coordinates, MESHFREE automatically creates new node points as a basis. Internally, MESHFREE divides the quadrilateral into two triangles (1-2-3 and 3-4-1, see BND_tria).

Note: The algorithm COMP_SortBEintoBoxes_Version = 4 only works for planar quadrilaterals.

BND_tria

create an independent triangle

An independent triangle can be defined by

- the (initial) coordinates (x1,y1,z1), (x2,y2,z2), (x3,y3,z3) of the three corners of the triangle.
  The cross product (x2-x1,y2-y1,z2-z1)x(x3-x1,y3-y1,z3-z1) forms the inward pointing direction of the triangle:

```
BND_tria &AliasName& x1 y1 z1 x2 y2 z2 x3 y3 z3 GeometryManipulations
```

- the node indices ip1, ip2, and ip3 for the three corners of the triangle:

```
BND_tria &AliasName& ip1 ip2 ip3 GeometryManipulations
```

If the triangle is defined by coordinates, MESHFREE automatically creates new node points as a basis.

BND_tria6N

create an independent 6-node triangle

A 6-node triangle is defined by the coordinates of its corners and the midpoints of its curved edges.
The curved edges are the quadratic parametric $(x_{ij}(s), y_{ij}(s), z_{ij}(s))^T$ such that

$$x_{ij}(0) = \text{xi}, \qquad x_{ij}(0.5) = \text{xij}, \qquad x_{ij}(1) = \text{xj}, \qquad \text{etc.}$$

In MESHFREE , the two possible definitions of an independent 6-node triangle are via

- the (initial) coordinates (x1,y1,z1), (x2,y2,z2), (x3,y3,z3) of the three corners of the triangle
  and the (initial) coordinates (x12,y12,z12), (x23,y23,z23), (x31,y31,z31) for the three edge midpoints of the triangle.
  The cross product (x2-x1,y2-y1,z2-z1)x(x3-x1,y3-y1,z3-z1) forms the inward pointing direction of the triangle:

```
BND_tria6N &AliasName& x1 y1 z1 x2 y2 z2 x3 y3 z3 x12 y12 z12 x23 y23 z23 x31 y31 z31 GeometryManipulations
```

- the node indices ip1, ip2, and ip3 for the three corners of the triangle
  and the node indices ip12, ip23, and ip31 for the three edge midpoints of the triangle:

```
BND_tria6N &AliasName& ip1 ip2 ip3 ip12 ip23 ip31 GeometryManipulations
```


delete{

delete all the geometry belonging to a given alias-group

```
begin_boundary_elements{ }
...
delete{ "Alias1","Alias2","Alias3",...}
...
end_boundary_elements
```

All geometry elements which belong to the given alias "Alias1", "Alias2", and "Alias3" are deleted. MESHFREE tries to shrink the boundary element arrays if possible.

include{

definition of a geometry file to be read by MESHFREE

Reading a geometry file is done in the following way:

```
begin_boundary_elements{ }
...
include{ FileName}
...
end_boundary_elements
```

No need to put the file name in double quotes!

A geometry file usually provides a set of node points as well as a set of topological connections of the node points in order to create triangles, quads, but also points and lines.

Supported formats:

- PAMCRASH
- STL (ascii only!)
- MSH
- OBJ
- FDNEUT
- NASTRAN

Sometimes, it is necessary to geometrically modify geometry entities. That can be done by

```
begin_boundary_elements{ }
...
include{ FileName} GeometryManipulations GeometryRestrictions exportGeometry{ }
...
end_boundary_elements
```

The categories GeometryManipulations , GeometryRestrictions , exportGeometry{ } (or exportFile{ }) are optional. None, a choice of them, or even all
of them in the same statement/line are accepted.

List of members:	
GeometryManipulations	geometrical modifications of boundary elements files read
GeometryRestrictions	restrictions for boundary elements files read
exportGeometry{	export the actually imported geometry file in STL or OBJ format
exportFile{	export the actually imported geometry file in STL or OBJ format
MSH	.msh file format for geometries
OBJ	.obj file format for geometries
STL	.stl file format for geometries

GeometryManipulations

geometrical modifications of boundary elements files read

Sometimes, it is necessary to geometrically modify geometry entities. That can be done by

```
begin_boundary_elements{ }
...
include{ FileName} GeometryManipulations
...
end_boundary_elements
```

GeometryManipulations can be a list. It will be executed in the order as they appear.
Actions like scale, offset etc. can even be repeatedly be applied.

Example:

```
begin_boundary_elements{ }
...
include{ FileName} scale{...} offset{...} scale{...} rotate{...} offset{...}
...
end_boundary_elements
```

The geometry manipulations are applied to the node points. The topology connections describing the elements are not touched.

List of members:

applyAlias{	Rename BoundaryElements with the given alias name
coarsenGeometry{	coarsen the triangulation of the specified part of the geometry
duplicate{	Duplicate part of the geometry and apply a new alias
extrude{	Extrude a 2D surface in one direction to a 3D object
layerByCluster{	assign the layer-property of a geometrical entity, possibly overrides the user given value from the ALIAS block
mirror{	generalized mirroring across a plane
offset{	shift the given geometry by a vector
removeBEonCondition{	remove boundary elements based on a (mathematical) condition
removeCluster{	removes cluster(s) of the current geometry subset due to given conditions
removeIsolatedClusters{	remove clusters who have less than a given number of single geometry elements (triangles, quads, etc.)
removeOuterShell{	for shell geometry given by two closed surfaces, remove outer surface
removeTinyClusters{	remove tiny parts from a geometrical entity
reorientation{	reorientation (inside/outside) of parts of the geometry
revOrient{	Invert orientation of boundary elements
rotate{	rotate the given geometry about a point with a rotation axis and angle
scale{	scale the given geometry about the origin
symmetryfaceByCluster{	automatic distribution of SYMMETRYFACE-flags to geometry components
thickenabs{	move a given part of the geometry by an absolute value of distance
thickenexp{	move the given part of the boundary by a relative value, correlated to the locally given smoothing length
turn_6NodeTriangles_into_3NodeTriangles{	Turn 6-node triangles into 3-node triangles

applyAlias{

Rename BoundaryElements with the given alias name

Rename BoundaryElements with the given AliasForGeometryItems , for example to give the same alias to all geometry parts in a geometry file, irrespective of what names were defined in the file.

Example:

```
begin_boundary_elements{ }
include{ cube.msh} applyAlias{ "cube"} # whole geometry gets renamed to cube
end_boundary_elements
```

Note: Internally, this command overwrites aliases after the geometry file has been read completely. This implies that, even when applyAlias{ is used, all parts in the geometry file need to have a valid alias (default alias is sufficient) in order to successfully complete the reading process.

---

coarsenGeometry{

coarsen the triangulation of the specified part of the geometry

```
begin_boundary_elements{ }
include{ FileName}, ..., coarsenGeometry{ lengthThreshold }
manipulate{ "someAlias"}, ..., coarsenGeometry{ lengthThreshold }
end_boundary_elements
```

lengthThreshold : MESHFREE will cluster all those geometry node points, whose distance is less than the given threshold.

Prior to the clustering, all node points obtain an importance-weight. Points on a geometry edge have a higher weight than regular node points.
The new location of the clustered points NP_1, NP_2 is the mean value $\mathbf{x}_{clustered} = \frac{1}{2}\left(\mathbf{x}_{NP_1} + \mathbf{x}_{NP_2}\right)$, if the weights are equal.
Otherwise, $\mathbf{x}_{clustered} = \mathbf{x}_{NP_k}$ where NP_k is the index with the bigger weight.

This feature is currently experimental !

---

duplicate{

Duplicate part of the geometry and apply a new alias

Duplicate part of the geometry and apply a new alias. By default, the orientation of the duplicated geometry is inverted for use with a second chamber. If the original orientation is required, this can be achieved with an extra call to revOrient{ }.

Note: The aliases of the duplicated geometry have to be defined as usual (see AliasForGeometryItems). This means, flags such as BC , ACTIVE , IDENT , etc. are not inherited from the original geometry.

Examples:

- With inverted orientation for use with a second chamber (duplicate geometry with alias "sphere" and apply alias "bubble"):

  ```
  begin_boundary_elements{ }
  include{ sphere.msh} # contains alias "sphere"
  manipulate{ "sphere"} duplicate{ "bubble"}
  end_boundary_elements
  ```

- With original orientation for a translated copy of the geometry with different alias (step 1 - duplicate geometry with alias "cube" and apply alias "cube_offset", step 2 - restore original orientation and translate alias "cube_offset"):

```
begin_boundary_elements{ }
include{ cube.msh} # contains alias "cube"
manipulate{ "cube"} duplicate{ "cube_offset"}
manipulate{ "cube_offset"} revOrient{ } offset{ 0,0,1}
end_boundary_elements
```


extrude{

Extrude a 2D surface in one direction to a 3D object

Extrude a 2D surface in one direction to a 3D object. This is useful, if you have a 2D sketch and want to create a 3D geometry from it.

```
manipulate{ "Alias"} extrude{ OPTIONAL: %GEO_open% , DirectionX, DirectionY, DirectionZ, OPTIONAL: DirectionLength }
```

The vector (**DirectionX** , **DirectionY** , **DirectionZ**) gives the direction in which to extrude. It can optionally be normalized,

so that the user can specify the length of extrusion with DirectionLength .
For example, to construct an open container, the user also has the option to leave the extruded object open at the other end.
For this the keyword %GEO_open% must be set. The default is %GEO_close%.

Note: Always check the normals for an extrude command! It may be that the normals still have to be reoriented with revOrient{ } as needed.
Example:

```
begin_boundary_elements{ }
manipulate{ "Alias1"} extrude{ 0, 0, 0.5 } # extrude Alias1 in z direction with length 0.5
manipulate{ "Alias2"} extrude{ 0, 0, 0.5, 2.0 } # extrude Alias2 in z direction with length 2.0
manipulate{ "Alias3"} extrude{ %GEO_open%, 0, 0, 0.5, 2.0 } # extrude Alias3 in z direction with length 2.0 and leave the extrusion open
end_boundary_elements
```


layerByCluster{

assign the layer-property of a geometrical entity, possibly overrides the user given value from the ALIAS block

After detecting all clusters of the geometry, see (CLUSTER) , MESHFREE assigns the cluster index to the LAYER information.
By this, possibly given LAYER indices by the user within the ALIAS defintions are overwritten.

mirror{

generalized mirroring across a plane

include{ File} ... mirror{ X,Y,Z, NormalX, NormalY, NormalZ, OPTIONAL:NormalLength } ...

Given a point (**X** , **Y** , **Z**) and a unit normal (**NormalX** , **NormalY** , **NormalZ**)
or a normal (**NormalX** , **NormalY** , **NormalZ**) that is scaled to NormalLength ,
this operation mirrors the geometry across the plane through (X,Y,Z)
perpendicular to its normal (NormalX, NormalY, NormalZ).

Examples:

include{ File} ... mirror{ 0,0,0, 1,0,0} ...
include{ File} ... mirror{ 1.5,2.0,0.5, 1,1,1, 1.0 } ...

The generalized behavior for non-unit length L of the normal
would move each node P of the geometry to $P + L^2(P_{\text{mirrored}} - P)$.

offset{

shift the given geometry by a vector

include{ File} ... offset{ shift_x, shift_y, shift_z, OPTIONAL:ShiftDistance } ...

The geometry is shifted by the given vector (**shift_x** , **shift_y** , **shift_z**).

If the optional parameter **ShiftDistance** is given AND non-zero, then the vector (shift_x, shift_y, shift_z) only represents the shifting direction, into which the object is shifted by the given distance ShiftDistance.

removeBEonCondition{

remove boundary elements based on a (mathematical) condition

For the given geometrical entity (file or AliasName), boundary elements are deletrd, if they fulfill a given condition.
The conditions can be based on the geometry of the node points or the center of gravity
Additional conditions might be possible based on the layer, the size of the boundary element etc. (see the list below)

```
begin_boundary_elements{ }
include{ FileName} removeBEonCondition{ %GEO_removeBasedOnNodes% | %GEO_removeBasedOnCOG% , [
equationText ] } # remove boundary elements if evaluation of the given equation returns a positive number
...
manipulate{ "AliasName"} removeBEonCondition{ %GEO_removeBasedOnNodes% | %GEO_removeBasedOnNodes%
, [equationText ] } # remove boundary elements if evaluation of the given equation returns a positive number
...
end_boundary_elements
```

examples :
manipulate{ "AliasName"} removeBEonCondition{ %GEO_removeBasedOnNodes% , [Y %ind_x(3)% > 0.8] }

-> delete element, if the z-components of all of the node points are bigger than 0.8

manipulate{ "AliasName"} removeBEonCondition{ %GEO_removeBasedOnCOG% , [Y %ind_x(3)% > 0.8] }

-> delete element, if the z-component of the center of gravity (COG) is bigger than 0.8

Additional conditions are possible using the items

- nromal information -> Y %ind_n(1)% , Y %ind_n(2)% , Y %ind_n(3)%
- area -> Y %ind_dA%
- layer information -> Y %ind_layer%
- boundary condition information -> Y %ind_BC%
- movement information -> Y %ind_MOVE%
- index of boundary element -> Y %ind_BE1%

IMPORTANT: The user connot use predefined equations here, so the construct

```
begin_equation{ "z_limit"}
Y %ind_x(3)% > 0.8
end_equation
begin_boundary_elements{ }
manipulate{ "AliasName"} removeBEonCondition{ %GEO_removeBasedOnCOG% , [ equn($z_limit$) ] }
end_boundary_elements
```

will not work, as equation definitions are not yet read at the time of BE-read-in.

removeCluster{

removes cluster(s) of the current geometry subset due to given conditions

```
manipulate{ "Alias1", ..., "AliasN"} removeCluster{ %GEO_RemoveClusterByIndex% , iIndex }
manipulate{ "Alias1", ..., "AliasN"} removeCluster{ %GEO_RemoveClusterClosestToGivenPoint% , x, y, z }
```

List of members:

%GEO_RemoveClusterByIndex%	%GEO_RemoveClusterByIndex%
%GEO_RemoveClusterClosestToGivenPoint%	%GEO_RemoveClusterClosestToGivenPoint%

%GEO_RemoveClusterByIndex%

```
manipulate{ "Alias1", ..., "AliasN"} removeCluster{ %GEO_RemoveClusterByIndex% , iIndex}
```

remove the cluster with the index iIndex. This function is difficult to use, as MESHFREE distributes the cluster indices automatically in the order as it finds them.
So, the way to use is to

- first let the simulation run with the SimCut functionality
- by postprocessing, check the cluster index MESHFREE has given to the particular partitions of the geometry
- add the statement in the frame above to the end of the inpuit file, i.e. add the lines

```
begin_boundary_elements{ }
mamipulate{"Alias1", ..., "AliasN"} removeCluster{ %GEO_RemoveClusterByIndex% , iIndex}
end_boundary_elements
```

where iIndex is now the dedicated cluster index found.
bulltelist#
This now removes ONE cluster. If more clusters are to be removed, repeat the procedure.

%GEO_RemoveClusterClosestToGivenPoint%

```
manipulate{ "Alias1", ..., "AliasN"} removeCluster{ %GEO_RemoveClusterClosestToGivenPoint% , x, y, z}
```

remove the cluster which is closest to the point with the coordinates (x,y,z).
OPTION IS NOT ACTIVE YET!!!

removeIsolatedClusters{

remove clusters who have less than a given number of single geometry elements (triangles, quads, etc.)

- For the given geometrical entity (file or AliasName), the geometry is scanned for all clusters, i.e. topologically connected parts of the geometry.
- Count the number of single elements (BND_tria , BND_quad , etc.) inside of each identified cluster.
- A cluster is deleted if the number of single entities is less than the given number.

```
begin_boundary_elements{ }
include{ FileName} removeIsolatedClusters{ N_min } # remove tiny clusters based on all the geometry read from the given file
...
manipulate{ "AliasName"} removeIsolatedClusters{ N_min } # remove tiny clusters based on the geometry described by the given AliasName
...
end_boundary_elements
```

N_min : minimum number of single elements required for a valid cluster.

removeOuterShell{

for shell geometry given by two closed surfaces, remove outer surface

For the simulation of the fluid dynamics inside a closed container, only the inner boundary of the container, facing the fluid domain, is required in MESHFREE . If the geometry file to be used contains the complete description of the container as two closed surfaces (shells), then the outer, unneccessary, one should be removed to save time in the point cloud organisation part of the simulation. MESHFREE can do this automatically with removeOuterShell{ factor }.

The parameter factor, chosen between 0 and 1, is used to check whether the volumes enclosed by the two surfaces are close enough to each other for a shell description of the geometry, that is, the outer shell is only removed if

$$V_{\text{inside}} \geq \text{factor} \cdot V_{\text{outside}}$$

Example:

```
begin_boundary_elements{ }
include{ FileName} removeOuterShell{ factor }
...
manipulate{ "AliasName"} removeOuterShell{ factor }
...
end_boundary_elements
```


removeTinyClusters{

remove tiny parts from a geometrical entity

- For the given geometrical entity (file or AliasName), the geometry is scanned for all clusters, i.e. topologically connected parts of the geometry.
- The area of the clusters is measured.
- The cluster with the biggest area is identified.
- The clusters whose area is, by a given factor, smaller than the biggest one, are removed. I.e. remove cluster i if $A_i < \alpha \cdot A_{biggest}$.

```
begin_boundary_elements{ }
include{ FileName} removeTinyClusters{ factor } # remove tiny clusters based on all the geometry read from the given file
...
manipulate{ "AliasName"} removeTinyClusters{ factor } # remove tiny clusters based on the geometry described by the given AliasName
...
end_boundary_elements
```

factor : the factor needed for the tiny-decision, i.e. the α above. This factor should be (much) smaller than 1.

reorientation{

reorientation (inside/outside) of parts of the geometry

Manipulate the orientation of the boundary elements of geometry parts, given by file or ALiasName.

- upon read in of a dedicated geometry files
- after reading the geometry based on the ALIAS name given

```
begin_boundary_elements{ }
include{ FileName} reorientation{ %GEO_Tube% , ... ) # orientation manipulation directly upon reading of file, the orientation
include{ FileName} reorientation{ %GEO_Vector% , ... ) # manipulation is effective for all entities read from file
...
manipulate{ "AliasName"} reorientation{ %GEO_Tube% , ... ) # orientation manipulation for a given alias, the orientation is adjusted for
manipulate{ "AliasName"} reorientation{ %GEO_Vector% , ... ) # all boundray elements which are so far read in and carry the name "AliasName"
...
end_boundary_elements
```

List of members:

%GEO_Vector%	geometry reorientation based on a given vector
%GEO_Tube%	reorient a part of the geometry in the tube sense

%GEO_Tube%

reorient a part of the geometry in the tube sense

Reorientation of tube-like or topologically closed parts of the boundary.

```
begin_boundary_elements{ }
include{ FileName}, ..., reorientation{ %GEO_Tube% , %GEO_Inside% , OPTIONAL:RatioForInternalParts }
include{ FileName}, ..., reorientation{ %GEO_Tube% , %GEO_Outside% , OPTIONAL:RatioForInternalParts }
end_boundary_elements
```

The geometry part should topologically be connected, i.e. triangles share the same nodes in order to provide geometrical connectivity.

The inside/outside orientation definition is given by the following infinitessimal movement approach:

1.) Define the normal direction of the i-th triangle formed by the points P_l, P_m, P_n by

$$\boldsymbol{n}_i = (P_m - P_l) \times (P_n - P_m),$$

where the area of the triangle is

$$A_i = \frac{1}{2}\|\boldsymbol{n}_i\|_2.$$

2.) Define an infinitessimal displacement of the j-th node point by

$$\tilde{P}_j = P_j + \sum_{k}^{\substack{\text{all triangles} \\ \text{attached to} P_j}} \epsilon \cdot \boldsymbol{n}_k,$$

$$\tilde{\boldsymbol{n}}_i = \left(\tilde{P}_m - \tilde{P}_l\right) \times \left(\tilde{P}_n - \tilde{P}_m\right),$$

$$\tilde{A}_i = \frac{1}{2}\|\tilde{\boldsymbol{n}}_i\|_2.$$

3.) The geometry is oriented to the inside, if

$$\sum_{i}^{\text{all triangles}} A_i > \sum_{i}^{\text{all triangles}} \tilde{A}_i.$$

4.) RatioForInternalParts: If the geometry is a closed chamber (such as a tank) that contains internal parts, then these parts will be oriented in the opposite direction. This is only the case if these parts fulfill the following criterion:

$$\text{RatioForInternalParts} \cdot \sum_{i}^{\text{all triangles}} A_i > \sum_{i}^{\text{all triangles of internal part}} A_i$$

List of members:

%GEO_Inside%	reorient (parts of) geometry towards its inside
%GEO_Outside%	reorient (parts of) geometry towards its outside

MESHFREE · InputFiles · USER_common_variables · BoundaryElements · include{ · GeometryManipulations · reorientation{ · %GEO_Tube% · %GEO_Inside%

%GEO_Inside%

reorient (parts of) geometry towards its inside

Reorientation of tube-like or topologically closed parts of the boundary towards the INSIDE. The way how to reorient is given in %GEO_Tube% .

MESHFREE · InputFiles · USER_common_variables · BoundaryElements · include{ · GeometryManipulations · reorientation{ · %GEO_Tube% · %GEO_Outside%

%GEO_Outside%

reorient (parts of) geometry towards its outside

Reorientation of tube-like or topologically closed parts of the boundary towards the OUTSIDE. The way how to reorient is given in %GEO_Tube% .

MESHFREE · InputFiles · USER_common_variables · BoundaryElements · include{ · GeometryManipulations · reorientation{ · %GEO_Vector%

%GEO_Vector%

geometry reorientation based on a given vector

Adjust the orientation of a geometrical entity based on a given vector. The boundary elemnts are adjusted such that the scalar product of their boundary normal and the given vector is positive.

```
begin_boundary_elements{ }
...
include{ FileName} reorientation{ %GEO_Vector% , Vx, Vy, Vz ) # manipulation of the whole fiel contents by the vector constraint
...
manipulate{ "AliasName"} reorientation{ %GEO_Vector% , Vx, Vy, Vz ) # manipulate all boundary elements having the "AliasName" by the vector contraint
...
end_boundary_elements
```

(Vx, Vy, Vz) are the components of the vector respectively. The vector does not necessarily have unit length.

MESHFREE · InputFiles · USER_common_variables · BoundaryElements · include{ · GeometryManipulations · revOrient{

revOrient{

Invert orientation of boundary elements

Invert orientation of boundary elements for a given file or alias for example in these cases

- Invert orientation of all geometry parts inside a geometry file
- Multiple geometry files with data for the same alias but with different orientations (in which case REV_ORIENT is insufficient)
- Duplication of geometry parts with same orientation at multiple locations using duplicate{ }.

Examples:

```
begin_boundary_elements{ }
include{ cube.msh} revOrient{ } # orientation of whole geometry in this file inverted
end_boundary_elements
```

```
begin_boundary_elements{ }
include{ cube.msh}
manipulate{ "top"} revOrient{ } # only orientation of alias "top" inverted
end_boundary_elements
```

If the orientation of parts of the geometry is inconsistent, use reorientation{ } instead.

MESHFREE · InputFiles · USER_common_variables · BoundaryElements · include{ · GeometryManipulations · rotate{

rotate{

rotate the given geometry about a point with a rotation axis and angle

```
include{ File} ... rotate{ O_x, O_y, O_z, Phi_x, Phi_y, Phi_z, OPTIONAL:RotationAngle } ...
```

The geometry is rotated about the point (**O_x** , **O_y** , **O_z**) with the rotation vector (**Phi_x** , **Phi_y** , **Phi_z**).
The vector (Phi_x,Phi_y,Phi_z) provides the rotation axis. If RotationAngle is NOT given, then
the length of the vector (Phi_x,Phi_y,Phi_z) provides the angle of rotation in radians.

I f RotationAngle is given, then the length of (Phi_x,Phi_y,Phi_z) does not play any role. MESHFREE will (internally) normalize
this vector and apply the rotation angle given in the optional variable RotationAngle.

Warning: If the length of the vector (Phi_x,Phi_y,Phi_z) is zero, no rotation can be effected.

MESHFREE · InputFiles · USER_common_variables · BoundaryElements · include{ · GeometryManipulations · scale{

scale{

scale the given geometry about the origin

The geometry is scaled about the origin. Either a global factor is given, that scales the geometry identically in all main directions,
or three factors are given, representing the stretching in the three main directions (x,y,z):

```
include{ File} ... scale{ Factor} ...
```

or

```
include{ File} ... scale{ Factor_x, Factor_y, Factor_z} ...
```

or one-dimensional strectching

```
include{ File} ... scale{ nx, ny, nz, Factor_n} ...
```

or scaling around a certain point of origin

```
include{ File} ... scale{ Ox, Oy, Oz, Factor_x, Factor_y, Factor_z} ...
```

---

symmetryfaceByCluster{

automatic distribution of SYMMETRYFACE-flags to geometry components

```
begin_boundary_elements{ }
include{ FileName} ... symmetryfaceByCluster{ }
end_boundary_elements
```

The geometry part might contain separated components or clusters. MESHFREE will set the SYMMETRYFACE -flag by the automatically given cluster indices. All cluster flags provided by the ALIAS -constraints are overwritten.

---

thickenabs{

move a given part of the geometry by an absolute value of distance

Thicken the geometry by moving the node points of the defined geometry parts.

```
begin_boundary_elements{ }
include{ FileName} thickenabs{ thickeningDistance, OPTIONAL: N_ThickeningLoops }
manipulate{ "AliasName"} thickenabs{ thickeningDistance, OPTIONAL: N_ThickeningLoops }
end_boundary_elements
```

thickeningDistance = the distance the boundary elements have to be moved is absolutely given by $D_i = \text{thickeningDistance}$ (no relative movement!!!!)
N_ThickeningLoops = the moving of the distance D_i is subdivided into N_ThickeningLoops steps.

EXPERIMENTAL only.

---

thickenexp{

move the given part of the boundary by a relative value, correlated to the locally given smoothing length

Thicken the geometry by moving the node points of the defined geometry parts.

```
begin_boundary_elements{ }
include{ FileName} thickenexp{ thickeningDistance, OPTIONAL: N_ThickeningLoops }
manipulate{ "AliasName"} thickenexp{ thickeningDistance, OPTIONAL: N_ThickeningLoops }
end_boundary_elements
```

thickeningDistance = the distance the boundary elements have to be moved is given by $D_i = \text{thickeningDistance} \cdot H_i$, i.e. the parameter thickeningDistance is relative with respect to the local smoothing length.
N_ThickeningLoops = the moving of the distance D_i is subdivided into N_ThickeningLoops steps.

EXPERIMENTAL only.

---

turn_6NodeTriangles_into_3NodeTriangles{

Turn 6-node triangles into 3-node triangles

Any 6-node triangle found among the considered parts of the geometry is turned into a 3-node triangle, that is the information about curved edge midpoints is ignored.

Example:

```
begin_boundary_elements{ }
include{ sphere.msh} turn_6NodeTriangles_into_3NodeTriangles{ }
end_boundary_elements
```

GeometryRestrictions

restrictions for boundary elements files read

Sometimes, it is desirable to use certain restrictions during read-in of boundary elements files. That can be done by

```
begin_boundary_elements{ }
...
include{ FileName} GeometryRestrictions
...
end_boundary_elements
```

GeometryRestrictions can be a list. It will be executed in the order as they appear.

Example:

```
begin_boundary_elements{ }
...
include{ FileName} only{...} sloppy{ }
...
end_boundary_elements
```

List of members:

append{	append the given string to all aliases in the geometry file
ignore{	ignore listed aliases from a geometry file
only{	read only elements of a given alias from file
sloppy{	do not stop program if geometry file contains an undefined alias

append{

append the given string to all aliases in the geometry file

```
begin_boundary_elements{ }
...
include{ FileName} append{ "aliasextension"}
...
end_boundary_elements
```

The aliases in FileName will be appended by the given string.

ignore{

ignore listed aliases from a geometry file

```
begin_boundary_elements{ }
...
include{ FileName} ignore{ "alias1", "alias2", ...}
...
end_boundary_elements
```

Do not read a boundary element from FileName, if it belongs to one of the given alias names.

only{

read only elements of a given alias from file

```
begin_boundary_elements{ }
...
include{ FileName} only{ "alias1", "alias2", ...}
...
end_boundary_elements
```

Read only the boundary elements from FileName, if they belong to one of the given alias names.

sloppy{

do not stop program if geometry file contains an undefined alias

```
begin_boundary_elements{ }
...
include{ FileName} ... sloppy{ } ...
...
end_boundary_elements
```

sloppy{ } avoids that the program stops execution if some of the alias given in the file does not exist.

MSH

.msh file format for geometries

Meshes generated in Gmsh - a free tool for mesh generation - are saved in the .msh file format.
MESHFREE supports triangular surface meshes of **version 2.2 and 4.1 only** .
Define physical entities for the boundary elements in Gmsh and refer to their names in AliasForGeometryItems to define the properties of the boundary elements.

Good to know:

- As .msh can be also potentially dangerous email attachment under Windows, many spam filters filter these files out, even if they are in a zip-file. You can however prevent this by renaming the ending. MESHFREE will still be able to interpret the geometry information.

List of members:

PrepareGeometryBy_GMSH	prepare MESHFREE geometries by GMESH, an open source software for geometrical preprocessing

---

PrepareGeometryBy_GMSH

prepare MESHFREE geometries by GMESH, an open source software for geometrical preprocessing

Important:
MESHFREE needs the surface/shell of the considered geometry. MESHFREE is able to read-in different geometry formats:

- STL (ASCII)
- FDNEUT (native Fidap Neutral geometry format)
- PAMCRASH
- NASTRAN
- GMSH
- OBJ

Especially OBJ and STL are formats, that can easily be generated by most of the classical CAD-tools. STL is widely used, however consumes a lot of memory. OBJ is more efficient in memory, however the standard of this format is more extensive, not all features are implemented in MESHFREE .

The problem using STL or OBJ consists in the fact, that the used usually does not have control over the orientation (inside/outside) of the shells, which however is a necessary information for MESHFREE .

There are several features to control the orientation during start-up of MESHFREE .

A true control of the orientation and the way of surface triangulation can be taken by using the program GMESH (which is free). If you intend to work with formats like STL or OBJ (native CAD formats), you can skip this section.
In the tutorials, the geometries are mostly given as GMESH-generated files. It is clear, that the geometry could be given in different ways, of course.

Goals of this Unit:

- Creation of the geometry with GMSH.

Three Dimensional geometry generation:
If we want to generate the geometrical configuration file with GMSH the following steps have to be done in order to get the information required by MESHFREE :

- Construction of the geometry
- Generation of a mesh for each face which does not belong to a volume
- Generation of a mesh for each volume (in case a volume has been constructed at all)
- Specification of the boundary type (“WALL”) and naming faces or groups of faces
- 2D the meshing and save the mesh.

In a three dimensional setting volumes are not necessary. It suffices to build the faces, because MESHFREE only requires the geometrical information for the faces. If any face or boundary is not required even after the generation of the mesh by GMSH one can simply ignore the unnecessary face or boundary by using the flag “IGNORE” in the ALIAS section in “USER_common_variables.dat”.

begin_alias{ }
...
“groupname” = “ IGNORE ”
...
end_alias
How to generate a geometry using GMSH:

GMSH generates two kinds of files, namely filename.geo and filename.msh. The first file deals with the operations used to define the geometry and the second file contains the mesh generated by GMSH.
To open GMSH in Linux, you may use a shortcut of the following type:

```
alias mygmsh='/p/tv/local/Gmsh/gmsh-2.8.3-Linux/bin/gmsh &'
```

An alias can be defined in the start-up file, such as by editing .bashrc .

1.) kate ~/. bashrc &
2.) Edit alias section.
3.) save the file

Now the shortcut command is active any time a Linux-bash shell is launched.

Now, GMSH can be started by using the command mygmsh (since the alias is defined in this way, one can change this name accordingly).
At the moment when the GMSH window appears a file untitled.geo at the cd where user has opened the GMSH is generated.
The interface of GMSH has the following options to use:
Modules
1. Geometry .
2. Mesh.
3. Solver.
We will not use its solver section.

STARTING WITH GMSH:
(Here we generate a geometry using GMSH).
The reader should be familiar with the geometry which will be created in this section; the annulus. The mesh will be of higher resolution.

The following steps will be undertaken:

1.) Create the geometry using the GMSH GUI:

- A rectangle will be formed. This rectangle represents a radial plane through the annulus.
- The plane will be extruded-rotated in order to form a quadrant of the annulus.
- The above step is then repeated, until the complete annulus is formed.

2.) Define physical groups.
3.) Customize the geometry by editing the geometry script file.
4.) Produce a mesh.

Creating the geometry: Forming an annulus with extrusions

Through the interface go to “Geometry>>Elementary entities>>Add>>Point” and create the three points (2.5,0.0,0.0), (5.25,0.0,0.0), (8.0,0.0,0.0).

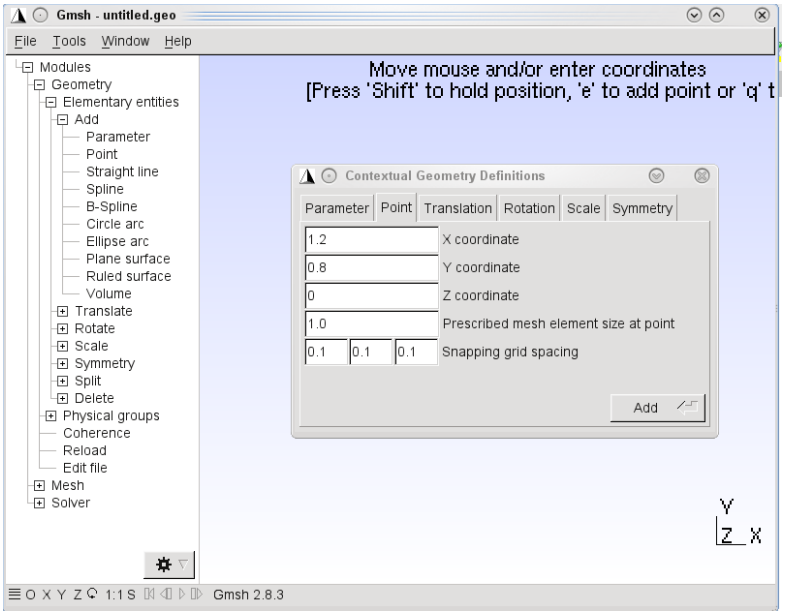


Figure 1:

The click at “Point” will create a “Contextual Geometry Definitions” window as shown in Figure 1.
At the first three spaces, the coordinates of the wanted point have to be defined and afterwards the point has to been added. Moving the mouse over the Gmsh window while adding a point may change the value of the coordinates in an unwanted way.
The above created points should lie on a line along the x-direction and form a radial line across the annulus. Create two lines: the first line should connect point 1 and point 2 and the second line point 2 and point 3. This can be done by clicking on “Geometry>>Elementary entities>>Add>>Straight line” and select via mouse the points which form the line. Now, extrude the lines to create surfaces (“Geometry>>Elementary entities>>Translate>>Extrude line”) as shown in Figure 2.

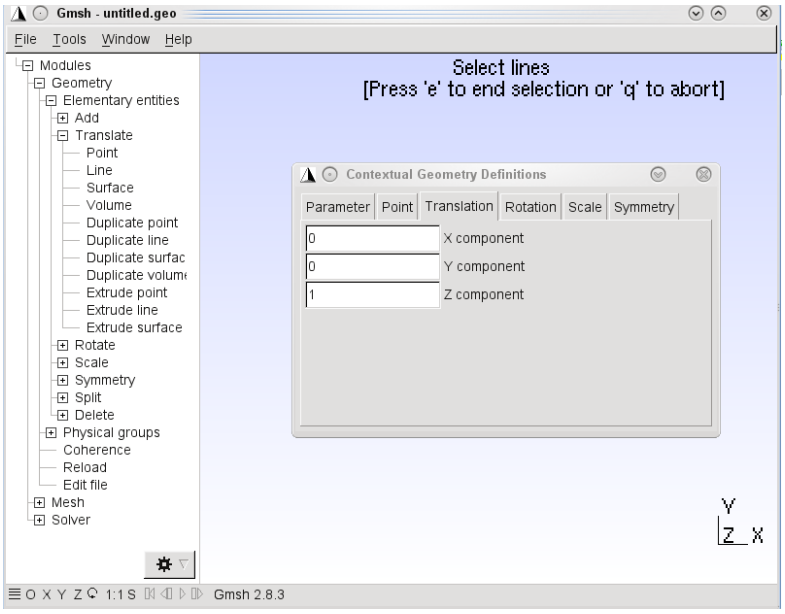


Figure 2:

Fill in (x,y,z) = (0.0,0.0,14.0). Note that both lines should be selected during the extrusion step. Finally change the view of point, so that you can see the rectangles just created, the result should be something similar as in Figure 3.

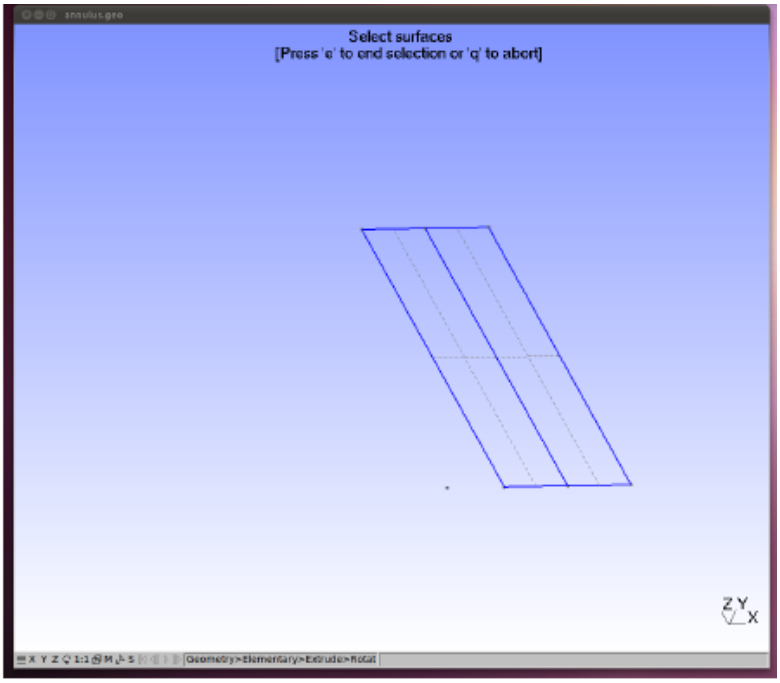


Figure 3:

The next step is to extrude-rotate the rectangular surface in order to create the annulus .

1.) Go to the top of the geometry module in the GMSH menu window.
2.) Click on Geometry>>Rotate>>Extrude surface. The contextual Geometry Definitions window will appear, on the Rotation tab. Also they will now be high-lighted on the graphic window, as red. The window is used to define the axis of revolution and the sweeping angle. The axis of revolution is defined by specifying any point on it and the components of a vector parallel to the axis. In addition, the sweeping angle must be specified in radians, in the anti-clockwise sense.
3.) Change the parameters in the Contextual Definitions window to the following:
0,0,0,0,0,1,pi/2.

4.) Pick both the surfaces on the graphic and press “e” (“e” is for adding the selection “u” is for undoing the last selection and “q” is for abort the mission.). A quarter of the annulus should have been formed in the graphic window, as shown in left panel of Figure 4.

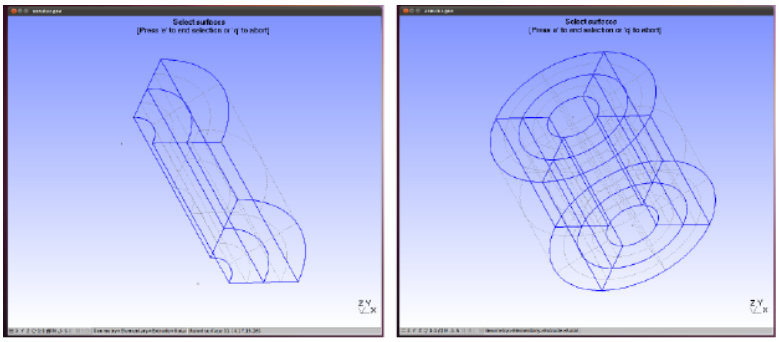


Figure 4:

Without changing the parameters in the “Contextual Geometry Definitions” window, pick the newly formed surfaces normal to the x- axis. And press the “e” key, to form half of the annulus. Repeat the procedure to form the complete annulus, shown in the right panel of Figure 4.

Physical Groups:
In case of 3D, MESHFREE needs only surfaces of the boundaries. So if created at all delete the volumes using “Geometry>>Delete>>Volume”. Doing so a small ball with yellow color will appear, selecting this volume will turn it red.

1.) Go to the top of the Geometry module in the GMSH menu window.
2.) Click on “Geometry>>Physical groups>>Add>>Surface”. Select the surfaces needed to be specified as boundaries and press “e”. All the surfaces selected in this way will from one group. If another group should be formed, pressing “e ” will differentiate between the groups.
3.) Open the script file i.e. filename.geo. Here, the names of the groups can be changed.

Meshing:
Before meshing the normal size has to be changed. In order to do so go to “Tools>>Options>>Mesh>>Visibility>>Normals and Tangents” and change the first space to 20 (or accordingly as shown in Figure 5).

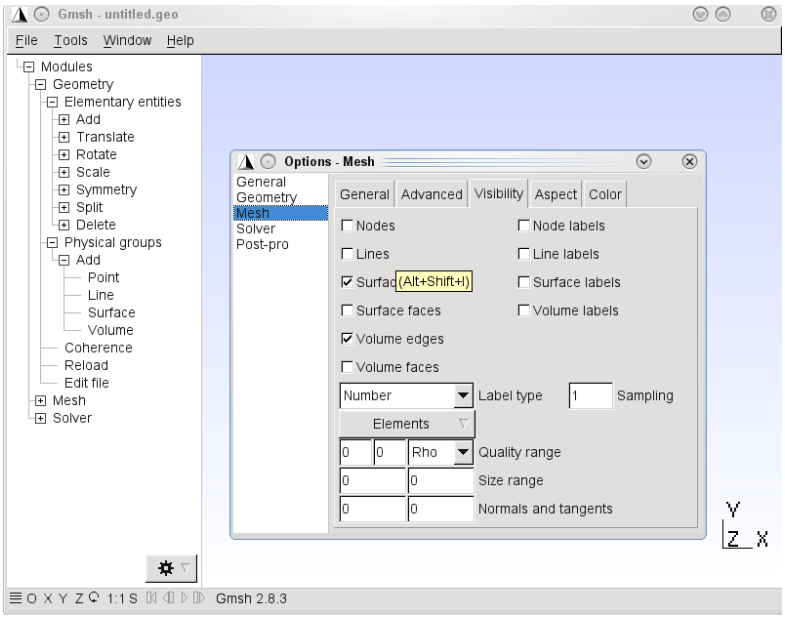


Figure 5:

Also, the Element size factor should be changed to 0.1. Therefore, go to “Tools>>Options>>Mesh>>General”.

Now go to the Mesh section and build a 2D mesh by clicking on “Mesh>>2D”. Go to “File” and save the mesh (“File>>Save Mesh”).
Close GMSH.
Change the name of the files generated by editing its name.
Note:- Before meshing user needs to reload the file (Geometry>>Reload), in the case that changes have been done by the user that should be reloaded by GMSH.

Understanding the steps of filename.geo:

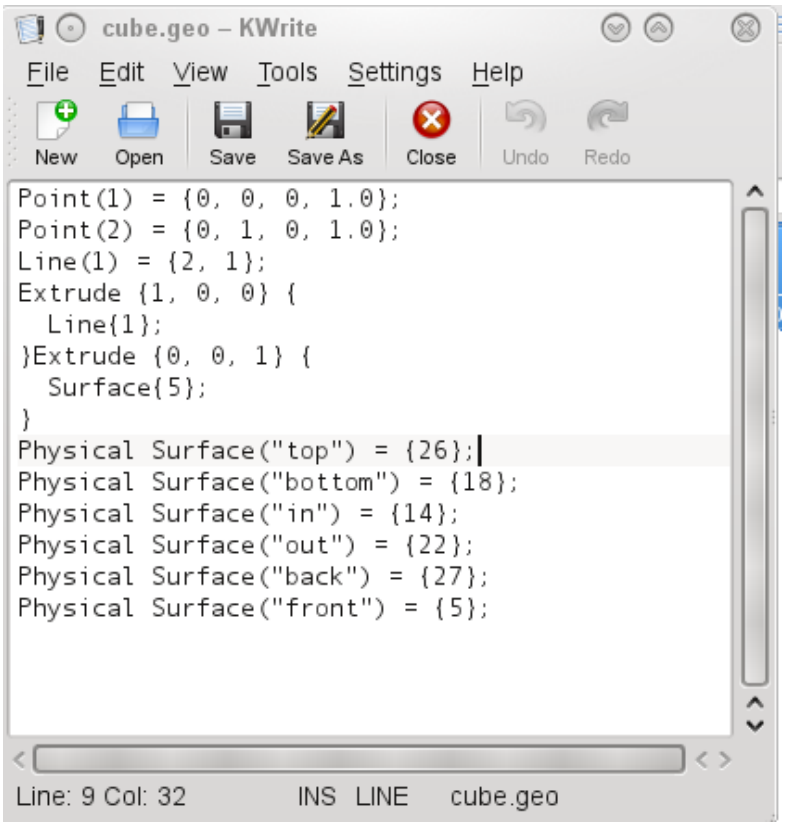


Figure 6:

Open the file cube.geo in tut3d_01 , a window as in Figure 6 will appear.
The first two lines of the document show that two points have been created followed by their corresponding coordinates.
The third line implies that a line has been generated connecting Point(1) and Point(2).
The fourth line says that in the direction of the x-axis (positive direction) Line{1} has been extruded to form a rectangular geometry.
Fifth line tells that the surface created has been extruded in the positive direction of z for one unit, to form a cube.
After that each line shows that a physical group has been added each time “e” was pressed (while preparing the geometry using GMSH). The names of the physical groups can be changed by writing “User_defined_Name” between the brackets following the text “Physical Surface” .
Note :- While saving the mesh one should check the direction of the normals in GMSH. If normals are pointing out side to the volume then it is correct if not the flag REV_ORIENT has to be used in front of the alias definitions of the corresponding surfaces in User_common_variables.dat.

OBJ

.obj file format for geometries

MESHFREE supports surface .obj surface meshes, where the boundary elements of the mesh are triangles or quadrilaterals.
MESHFREE derives the alias name of the surface entity in AliasForGeometryItems from the given group in the .obj file.

Group names for faces and surfaces can be added in .obj file in the following way.

```
v x_value y_value z_value
v x_value y_value z_value
...
g GroupName1
f v1 v2 v3
...
g GroupName2
f v1 v2 v3
...
```

If no group is given the alias name is taken from the file name.

STL

.stl file format for geometries

The current STL reader supports only ASCII representation.
Names for surfaces (solids) can be added in the .stl file in the following way.

```
solid Name1
facet normal n1 n2 n3
outer loop
vertex v11 v12 v13
vertex v21 v22 v23
vertex v31 v32 v33
endloop
...
endsolid Name1
...
```

If no solid name is given, it is called "face".

Note:

- Upon read-in of the solid, if "name" is written between quotation marks, it will be modified to name (without quotation marks).
- If the solid name is enumerated with numbers in brackets, e.g. 'solid name(1)', then MESHFREE stops the simulation.
- MESHFREE ignores color definition in the .stl file.
- Using the wildcard functionality (see AliasForGeometryItems) is recommended in case of additional information in the solid definition,
  e.g. 'solid cube <stl unit=MM>'.

---

exportFile{

export the actually imported geometry file in STL or OBJ format

See exportGeometry{ }.

---

exportGeometry{

export the actually imported geometry file in STL or OBJ format

Export the currently read geometry in STL or OBJ format. These formats can be directly visualized by ParaView.
Export is done either before or after GeometryManipulations are executed, or both.

```
begin_boundary_elements{ }
...
include{ FileName} ... exportGeometry{ ARGUMENTS} ...
...
end_boundary_elements
```

ARGUMENTS can be one or a subset of:

- OBJ (export in obj format)
- OBJs (export in obj format and separate geometry files for each group)
- OBJc (export in obj format and single geometry file with group names)
- STL (export in stl format)
- BEFORE or BeforeGeometryTransformations (export the geometry exactly as discribed in the original file FileName)
- AFTER or AfterGeometryTransformations (export the geometry after all manipulations have been executed)

The resulting files are written to ExportGeometryDirectory .

Example:

```
begin_boundary_elements{ }
...
include{ FileName} ... exportGeometry{ STL, BEFORE, AFTER} ...
...
end_boundary_elements
```

Exports the geometry before and after the GeometryManipulations have been executed. In this way, most preferably together with the option CONTROL_StopAfterReadingGeometry , one is able to quick-check the consistency of the GeometryManipulations .

Note: Currently, also the keyword exportFile{ } works in the same way as exportGeometry{ }.

If the exportGeometry{ }-option is used as standalone, i.e.

```
begin_boundary_elements{ }
...
include{ FileName}
...
manipulate{ "AliasName"}
...
exportGeometry{ [ STL ,OBJ],AFTER}
...
end_boundary_elements
```

a file with the name GIFgeometry.stl or GIFgeometry_*.obj is created in ExportGeometryDirectory .

List of members:

ExportGeometryDirectory	folder where to export the actually imported geometry

MESHFREE · InputFiles · USER_common_variables · BoundaryElements · include{ · exportGeometry{ · ExportGeometryDirectory

ExportGeometryDirectory

folder where to export the actually imported geometry

The exported files are written to the folders

```
ExportInputGeometry_BeforeGeometryTransformations/
ExportInputGeometry_AfterGeometryTransformations/
```

depending on the choice made.

---

manipulate{

manipulate (move, rotate, ...) the geometry belonging to an alias-group

```
begin_boundary_elements{ }
...
manipulate{ "Alias1","Alias2","Alias3",...} ListOfGeometryManipulations
...
end_boundary_elements
```

The *ListOfGeometryManipulations* might contain all valid elements from GeometryManipulations .

The working schedule of MESHFREE with respect to the boundary elements is sequential. Manipulation can be done only if the appropriate geometry elements (aliases) have already been read in from file.

Note:

- The geometry manipulations are performed for the specified aliases only, i.e. in multiphase simulations all desired phases/chambers of an alias have to be specified explicitly ("Alias1", "Alias1{2}", ...).
- The use of wildcards is possible (see AliasForGeometryItems).

Example:

- Wrong order

```
begin_alias{ }
"AliasA" = " ..." # definition of AliasA
end_alias
begin_boundary_elements{ }
manipulate{ "AliasA"} offset{ 1,1,0} rotate{ 0,0,0,3.14,0,0}
include{ File1Containing_AliasA_}
end_boundary_elements
```

- Correct order

```
begin_alias{ }
"AliasA" = " ..." # definition of AliasA
end_alias
begin_boundary_elements{ }
include{ File1Containing_AliasA_}
manipulate{ "AliasA"} offset{ 1,1,0} rotate{ 0,0,0,3.14,0,0}
end_boundary_elements
```

The true advantage becomes apparent if the feature is used together with the ConstructClause :

```
begin_alias{ }
"AliasA" = " ..." # definition of AliasA
end_alias
begin_boundary_elements{ }
include{ File1Containing_AliasA_} # read in geometry
end_boundary_elements
begin_construct{ }
"xMidPoint" = CONSTRUCT ( %CONSTRUCT_BoxMidPoint% , 0.5, "AliasA" ) # determine the mid point of the geometry

end_construct
begin_boundary_elements{ }
manipulate{ "AliasA"} rotate{ &xMidPoint(1)& , &xMidPoint(2)& , &xMidPoint(3)& , 2.1, 3.3, 0.1} # rotate about the mid point of the geometry
end_boundary_elements
```


3.1.7. CODI

solve additional COnvection-DIffusion-problems (CODI)

COnvection DIffusion Equation

Suppose there is a scalar item Φ and there exists a MESHFREE -index for Φ , such as %ind_PHI% .
The general type of equation to be solved is

$$d\Phi/dt + \nabla^T(\boldsymbol{v}_{\mathrm{Imp}} \cdot \Phi) + \boldsymbol{v}^T \cdot \nabla\Phi + A \cdot \Phi = \rho \cdot \nabla^T(D \cdot \nabla\Phi) + Q.$$

In USER_common_variables , the definition would look like:

```
CODI_Vimplicit ($Material$,%ind_PHI%) = (vImp_x, vImp_y, vImp_z)
CODI_V ($Material$,%ind_PHI%) = (v_x, v_y, v_z)
CODI_A ($Material$,%ind_PHI%) = A
CODI_rho ($Material$,%ind_PHI%) = rho
CODI_D ($Material$,%ind_PHI%) = D
CODI_Q ($Material$,%ind_PHI%) = Q
```

These items are optional. Therefore, by the reduced set

```
CODI_A ($Material$,%ind_PHI%) = A
CODI_Q ($Material$,%ind_PHI%) = Q
```

for each point the ODE $d\Phi/dt + A \cdot \Phi = Q$ will be solved for instance.

In order to assure some minimum and maximum conditions, the solution Φ can be restricted by:

```
CODI_min_max ($Material$,%ind_PHI%) = (min_PHI, max_PHI, OPTIONAL:AllowedSlopePHI )
```

MESHFREE simply cuts the solution of Φ after solving the differential equation.

If AllowedSlopePHI is given, then the solution is adapted such that

$$\|\nabla\Phi\|_2 \leq \mathrm{AllowedSlopePHI}.$$

AllowedSlopePHI, naturally, has to be positive.

Boundary conditions for the problem are set with BCON .

Function Value Assignment

Besides differential equations, one can also just assign a value to Φ by an algebraic equation:

```
CODI_eq ($Material$,%ind_PHI%) = RightHandSideExpression
```

Usually, the user does not want to provide additional PDEs for items like velocity, pressure etc because they are already solved by MESHFREE in the most efficient way. In order to construct new, additional MESHFREE -variables, UserDefinedIndices %indU_...% (or %ind_addvar% , legacy code) can be used.

See also CODI_min_max and CODI_min_max_RejectLinearSolution .

List of members:

CODI_A	See CODI
CODI_D	See CODI
CODI_eq	See CODI
CODI_Integration	CODI type of integration and time step size
CODI_min_max	set lower and upper bound for any MESHFREE variable
CODI_min_max_RejectLinearSolution	rejection of the solution of a sparse linear system if minimum-maximum criteria are not fulfilled
CODI_Q	See CODI
CODI_rho	See CODI
CODI_V	See CODI
CODI_Vimplicit	See CODI

MESHFREE · InputFiles · USER_common_variables · CODI · CODI_A

CODI_A

See CODI

See CODI .

MESHFREE · InputFiles · USER_common_variables · CODI · CODI_D

CODI_D

See CODI

See CODI .

MESHFREE · InputFiles · USER_common_variables · CODI · CODI_Integration

CODI_Integration

CODI type of integration and time step size

CODI_Intagration($Material$,%ind_PHI%) = (TypeOfIntegration, OPTIONAL: CODI_dt)

- either %CODI_implicit% (default) or %CODI_explicit%
- OPTIONAL: give a time step size for integration of this particular CODI , which might be bigger than the current numerical time step size of LIQUID -integration.
  If not given, the CODI_dt is as big as the time step size of the running simulation.
  CODI_dt chosen smaller than the time step size of the running simulation be be ignored. In this case, reduce the time step size in general.

MESHFREE · InputFiles · USER_common_variables · CODI · CODI_Q

CODI_Q

See CODI

See CODI .

MESHFREE · InputFiles · USER_common_variables · CODI · CODI_V

CODI_V

See CODI

See CODI .

MESHFREE · InputFiles · USER_common_variables · CODI · CODI_Vimplicit

CODI_Vimplicit

See CODI

See CODI .

MESHFREE · InputFiles · USER_common_variables · CODI · CODI_eq

CODI_eq

See CODI

See CODI .

MESHFREE · InputFiles · USER_common_variables · CODI · CODI_min_max

CODI_min_max

set lower and upper bound for any MESHFREE variable

Equivalent to ENFORCE_min_max .

MESHFREE · InputFiles · USER_common_variables · CODI · CODI_min_max_RejectLinearSolution

CODI_min_max_RejectLinearSolution

rejection of the solution of a sparse linear system if minimum-maximum criteria are not fulfilled

Equivalent to ENFORCE_min_max_RejectLinearSolution .

CODI_rho

See CODI

See CODI .

3.1.8. COUPLING

couple the running MESHFREE simulation to another, currently running simulation

Currently, only MESHFREE -MESHFREE coupling is implemented.

The COUPLING functionality, however, is set up in a general way,
such that coupling to other codes shall be possible.

List of members:

BFT	coupling to other running simulations by file transfer (BFT=ByFileTransfer)

BFT

coupling to other running simulations by file transfer (BFT=ByFileTransfer)

All necessary data of the coupling are transfered by files (unformatted, streaming).

List of members:

CouplingBFT_WorkingDirectoryOfOtherSimulation	working directory of another simulation to which couling has to be performed
CouplingBFT_TypeOfOfOtherSimulation	give the type of the other simulation (MESHFREE, PAMCRASH, ABAQUS, ...)
CouplingBFT_DataRequest	launch data request to another running MESHFREE-simulation
CouplingBFT_Synchronization	synchronize two running simulations if coupled to each other

CouplingBFT_DataRequest

launch data request to another running MESHFREE-simulation

This simulation sends out positions (i.e. point coordinates) to another MESHFREE simulation. At these points, requested function values are interpolated
by least-squares-approximation

```
CouplingBFT_WorkingDirectoryOfOtherSimulation (1) = '/a/b/c/MF2'
CouplingBFT_Synchronization (1) = (%CouplingBFT_RequestOtherProcessToWait%)
CouplingBFT_DataRequest (1) = ( AfterHowManyTimeSteps, [FunctionalToMarkTheRequestPoints],
[ChamberIndexInMF2], listOf(%ind_...%), listOf(%indU_...%) )
```

AfterHowManyTimeSteps : data request is launched with a certain frequency (that means one can prevent MESHFREE from doing that data request in every time step)

FunctionalToMarkTheRequestPoints : a typical RightHandSideExpression in order to mark those points at which data is requested

ChamberIndexInMF2 : iterpolate data out of this chamber in MF2

listOf(%ind_...%) : list of entities in MF2 (given by their proper %ind_...%) that have to be interpolated

listOf(%indU_...%) : list of indices where the interpolation results have to be stored in MF1

Example:
Suppose, simulation MF1 runs water-flow for which wind forces have to be taken into account

```
CouplingBFT_WorkingDirectoryOfOtherSimulation (1) = '/a/b/c/MF2' # path of MF2
CouplingBFT_Synchronization (1) = (%CouplingBFT_RequestOtherProcessToWait%, 0) # let MF2 be in standby for all
times
# interpolate the velocity in first chamber of MF2 at the locations for x>0.1 and store them ind the indices %indU_v(n)%
CouplingBFT_DataRequest (1) = ( 10, [Y%ind_x(1)%>0.1], 1, %ind_v(1)% , %ind_v(2)% , %ind_v(3)% , %indU_v(1)%,
%indU_v(2)%, %indU_v(3)% )
```

Simulation MF2 is in standby. It contains, on its pointcloud, the results of a stationary wind-profile. The wind profile might be a result of a MESHFREE -simulation,
or it might have been read from file as result of another flow simulation such as openFOAM, FLUENT, etc.

```
#UCVCODE
CouplingBFT_WorkingDirectoryOfOtherSimulation (1) = '/a/b/c/MF1' # path of MF1
# step 1: let us read in some flow solution of another simulation tool by MESHFREE's read-in-functionality, see for example
ASCII___RIPC___
# step 2: allow MF1 to launch data requests to this running MESHFREE process (MF2),
# this also means that MF1 will put it in pure standby, no simulation is performed in MF2,
# only answering to data requests.
UCVCODE# frame#
```

List of members:	
DataStructure_ToBeSentToFPM	exact data structure to send data request to MESHFREE
DataStructure_SentBackFromFPM	exact data structure returned by MESHFREE upon data request

DataStructure_SentBackFromFPM

exact data structure returned by MESHFREE upon data request

This is the documentation of the data structure send back from MESHFREE after launching a data request by DataStructure_SentToFPM .

DataStructure_ToBeSentToFPM

exact data structure to send data request to MESHFREE

This is the documentation of the data structure to be sent to MESHFREE in order to launch a data request at a set of locations.

CouplingBFT_Synchronization

synchronize two running simulations if coupled to each other

The simulation running in the folder '/a/b/c/MF1' requires the following lines:

```
CouplingBFT_WorkingDirectoryOfOtherSimulation (1) = '/a/b/c/MF2' # couple to the simulation running in this folder
CouplingBFT_Synchronization (1) = ( %CouplingBFT_RequestOtherProcessToWait% ) # request the other simulation to wait for the current simulation
```

The simulation running in the folder '/a/b/c/MF2' requires the following lines:

```
CouplingBFT_WorkingDirectoryOfOtherSimulation (1) = '/a/b/c/MF1' # couple to this simulation
CouplingBFT_Synchronization (1) = ( %CouplingBFT_RequestOtherProcessToWait% ) # this line is optional if it is clear that this simulation runs faster
```

The simulation running in '/a/b/c/MF1' will create in the folder '/a/b/c/MF2/CouplingBFT/MF1'
the file 'Synchronization_RequestToWait' which contains the current simulation time of simulation MF1.
Simulation MF2 interpretes this time as strong request and will continue only, if t(MF1) >= t(MF2).

The waiting business makes sense only if the two simulations exchange data. See CouplingBFT_DataRequest .
If no synchronization request is launched, no waiting/standby takes place, each simulation runs on its own. However, still, each simulation checks for CouplingBFT_DataRequest .

List of members:

%CouplingBFT_RequestOtherProcessToWait%	request another running simulation to wait for myself
%CouplingBFT_RequestMyselfToWait%	request myself (current simulation) to wait for another running simulation

---

%CouplingBFT_RequestMyselfToWait%

request myself (current simulation) to wait for another running simulation

TO BE IMPLEMENTED SOON

%CouplingBFT_RequestOtherProcessToWait%

request another running simulation to wait for myself

```
CouplingBFT_WorkingDirectoryOfOtherSimulation (1) = '/a/b/c/MF2' # couple MF1 to the simulation running in this folder (MF2)
CouplingBFT_Synchronization (1) = ( %CouplingBFT_RequestOtherProcessToWait% , OPTIONAL: timeAtWhichTheOtherProcessHasToWait ) # request the other simulation to wait for the current simulation
```

1.) if no optional argument is given, then the present simulation sends its current time t(MF1) to MF2, running in '/a/b/c/MF2'. If this process agrees to couple, then it will go into standby modus if t(MF2) > t(MF1) . MF2 then regularly checks the timeAtWhichTheOtherProcessHasToWait sent by MF1 until t(MF2) <= t(MF1)
2.) If the oprional argument is given, then MF2 goes to standby, if t(MF2) > timeAtWhichTheOtherProcessHasToWait

Remarks:

1.) in standby modus, MESHFREE regularly (every 0.01 seconds) if new synchronization or data request have arrived.
2.) by setting timeAtWhichTheOtherProcessHasToWait = 0, MF2 will be always in standby, waiting for data requests only, see CouplingBFT_DataRequest .

CouplingBFT_TypeOfOfOtherSimulation

give the type of the other simulation (MESHFREE, PAMCRASH, ABAQUS, ...)

Currently, only coupling to other MESHFREE processes is implemented, so this statement is optional.

Syntax:

```
CouplingBFT_TypeOfOfOtherSimulation (n) = %CouplingBFT_OtherSimulation_IsFPM%
```

This is also the default.

List of members:

%CouplingBFT_OtherSimulation_IsFPM%	other running (coupled) simulation is MESHFREE

CouplingBFT_WorkingDirectoryOfOtherSimulation

working directory of another simulation to which couling has to be performed

```
CouplingBFT_WorkingDirectoryOfOtherSimulation (n) = 'FullPath_Or_relativePath'
```

Coupling to the n-th process which runs in the given directory.

If two (or more) processes are to couple, then this statement is essential.
If two (or more) MESHFREE -processes are to couple, all MESHFREE processes have to give this link to the other running simulation.
In fact, this statement finally allowes that other processes send requests to the current process.

If this statement is given, a local folder 'CouplingBFT' is created. It contains subfolders, whose names are the ones of the coupled simulations.
A subfolder with the own name is created as well.

3.1.9. ConsistencyChecksAtStartup

check the physical/mathematical consistency for user-given input data

```
ConsistencyChecksAtStartup = ( Identification, RightHandSideExpression , WhatShallMESHFREEdo,
"ErrorOrWarningText" )
ConsistencyChecksAtStartup = ( Identification, RightHandSideExpression , WhatShallMESHFREEdo, "Warning or error
text", SomeExpression, "more text // more text in the next line", "// And even more text in the next line" )
```

Identification:

- -1 (check done after reading geometry, but before filling points)
- 0 (check done before the first time step, i.e. after filling the geometry by points)
- N (check done after each time cycle until N-th time cycle is reached)

RightHandSideExpression: If positive, then MESHFREE will handle the problem. In this case, it depends on what is given in WhatShallMESHFREEdo.

SomeExpression: Can be of the type RightHandSideExpression . It shall deliver a numerical value.

WhatShallMESHFREEdo:

- %ConsistencyChecksAtStartup_STOP%
- %ConsistencyChecksAtStartup_WARNING%

"ErrorOrWarningText": Text to appear in the warnings file. In order to have more readable text, use '//' in order to invoke a line break.

Example:

```
begin_alias{ }
"H_MESH" = "0.001" # user-defined triangulation size
"ScaleGeo" = "1.0" # user-defined scaling of the geometry
end_alias
begin_construct{ }
"IGESmin" = CONSTRUCT ( %CONSTRUCT_BoxMin% , "tube", "face", "out", "outflow" ) # minimum point enclosing box

"IGESmax" = CONSTRUCT ( %CONSTRUCT_BoxMax% , "tube", "face", "out", "outflow" ) # maximum point enclosing
box
"IGESdx" = "(&IGESmax(1)&-(&IGESmin(1)&))" # side length enclosing box, x-component
"IGESdy" = "(&IGESmax(2)&-(&IGESmin(2)&))" # side length enclosing box, y-component
"IGESdz" = "(&IGESmax(3)&-(&IGESmin(3)&))" # side length enclosing box, z-component
end_construct
ConsistencyChecksAtStartup (1) = ( -1, [sqrt( &IGESdx& ^2 + &IGESdy& ^2 + &IGESdz& ^2) > 1000* &H_MESH& ],
%ConsistencyChecksAtStartup_WARNING% ,
"Inconsistent dimensions of the problem. ",
"//Length scale (x,y,z) of IGES file = (", [ &IGESdx& ], [ &IGESdy& ], [ &IGESdz& ], " ). ",
"//Length scale of triangles is H_MESH = ", &H_MESH& , ". ",
"//Maybe the wrong scaling factor. Currently, ScaleGeo = ", &ScaleGeo& , ". ",
"//Or, maybe the wrong meshsize: H_MESH = ", &H_MESH& )

ConsistencyChecksAtStartup (2) = ( -1, [ &H_MESH& > 0.1 ], %ConsistencyChecksAtStartup_WARNING% ,
"Mesh size seems too big. H_MESH = ", &H_MESH& , ". //The mesh size has to be given in meters, no matter what is
the unit system of //the appropriate IGES file" )
```

List of members:

%ConsistencyChecksAtStartup_STOP%	stop MESHFREE if the consistency check applies
%ConsistencyChecksAtStartup_WARNING%	write a message in the warnings file if the consistency check applies

%ConsistencyChecksAtStartup_STOP%

stop MESHFREE if the consistency check applies

MESHFREE will stop. The text, which is given in the ConsistencyChecksAtStartup command, is launched as error message.

%ConsistencyChecksAtStartup_WARNING%

write a message in the warnings file if the consistency check applies

MESHFREE will NOT stop. Instead, the text, which is given in the ConsistencyChecksAtStartup command, is put in the warnings file.

3.1.10. Curves

define curves in the input file

In MESHFREE , curves are tables of values that can be used to assign any physical or geometrical quantity,
such as density depending on pressure or viscosity depending on temperature. They are defined in a begin_curve{ environment.

```
begin_curve{ $CurveName$ }
...
end_curve
```

We distinguish between 1D_Curves and 2D_Curves , for details see there.
Once a curve is defined, it can be used in a RightHandSideExpression , e.g. in a boundary condition:

```
BC_T ( $wall$ )=( %BND_DIRICH% , curve{ $CurveName$ })
```

Or within equations:

```
[ ... curve( $CurveName$ ) ... ]
```

They return linearly interpolated values between the given interpolation points.

List of members:

1D_Curves	define curves with one independent variable
2D_Curves	define curves with two independent variables

1D_Curves

define curves with one independent variable

1D curves define a relationship between an independent variable (in the first column) and one or more dependent variables (columns 2 and more) by giving data in a tabular way. Given a value for the independent variable, it will return linearly interpolated values for the dependent variable columns.

After the begin_curve{ statement, a default independent variable can be specified by the user with depvar_default . If this is not specified, the default independent variable is the simulation time, %ind_time% .

The data for the independent variable is in the first column and always has to be sorted in ascending order.

Example 1: density depending on temperature (ONE depending variable), first the definition:

```
begin_curve{ $DensityOnTemperature$ } depvar_default{ %ind_T% }
-273.15 1100
0 1000
4 1050
100 990
end_curve
```

and then the usage:

```
density( $MAT_user$ ) = curve{ $DensityOnTemperature$ }
```

If there are several dependent variables, the number has to be indicated by nb_functions , see the example below.

Example 2: gravity components depending on time (SEVERAL depending variables)

```
begin_curve{ $GravityOnTime$ } depvar_default{ %ind_time% } nb_functions {3}
0 0 0 -9.81
1 0 0 -9.81
1.01 0 0 9.81
10 0 0 9.81
end_curve
```

Note:

- Currently, it is not possible to use Equations in the independent variable column, i.e. equn{...} or []. This is only possible for the dependent variables.
- However, a ConstructClause can be used to define aliases that can be referenced in the independent variable column. Simple arithmetics are allowed in their definition; however, blanks are not. As a ConstructClause is evaluated only at the start of a simulation, only numbers or refrences to aliases can be used in the definition.

  Example 3: curve with ConstructClause -based aliases in independent variable

```
begin_alias{ }
"t1" = "1"
"t2" = "3"
end_alias
begin_construct{ }
"T_StartTest" = "&t1&" # result is 1
"T_EndTest" = "&t1&+&t2&" # result is 4; no blanks allowed
end_construct
begin_curve{ $CV_test$ } depvar_default{ %ind_time% }
0.0 0.0
&T_StartTest& 1.0
&T_StartTest& 1.0
&T_EndTest& 1.0
&T_EndTest& 1.0
10.0 0.0
end_curve
```

- If no dependent variable is specified, the simulation time at which the curve is interpolated will be used. For example, removing depvar_default{ %ind_time% } in Example 2 while still calling the curve without argument, will lead to the same result as Example 2 would.
- To overwrite a depvar_default by the standard behavior of using the simulation time, one can evaluate the curve with argument zero, e.g.

```
curve{ $DensityOnTemperature$ }{0}
```

- While the standard behavior and the use of %ind_time% is almost always equivalent, there can be niche cases where using %ind_time% leads to different results: For example, up until beta2020.08 of MESHFREE , using %ind_time% in a %PUBLICVALUE% integration statement would return uninitialized values on MPI processes with no points, while the standrad behavior would return the expected time. This is due to the fact that, for the case of %ind_time% , point data is acessed to retrieve time. If there are no points, no valid time can be retrieved. On the other hand, for the standard behavior, a global time variable is used, which is valid on every MPI process. Note that, while this specific interaction was algorithmically correct, it should, for convenience sake, not occur in newer versions (see %PUBLICVALUE%).

List of members:

nb_functions	defines the number of dependent variables in 1D curves
depvar_default	defines the index for the independent variable in 1D curves

depvar_default

defines the index for the independent variable in 1D curves

Options:

- Definition based on existing MESHFREE -variables (see Indices) by enclosing %-signs

```
begin_curve{ $...$ } depvar_default{ %ind_T% }
...
end_curve
```

- Definition based on Equations by enclosing the equation name by curly brackets

```
begin_curve{ $...$ } depvar_default{ equn{ $EQN_radius$ }}
...
end_curve
begin_equation{ $EQN_radius$ }
sqrt( Y %ind_x(1)% ^2 + Y %ind_x(2)% ^2 + Y %ind_x(3)% ^2 )
end_equation
```


nb_functions

defines the number of dependent variables in 1D curves

The keyword nb_functions defines the number of dependent variables in a 1D curve.

Example: 1D curve with one independent (time) and three dependent variables. The values for the dependent variables are found in the second to fourth column of the table.

```
begin_curve{ $GravityOnTime$ } depvar_default{ %ind_time% } nb_functions {3}
0 0 0 -9.81
1 0 0 -9.81
1.01 0 0 9.81
10 0 0 9.81
end_curve
```


2D_Curves

define curves with two independent variables

2D curves are characterized by a vertical and a horizontal independent variable as well as one dependent variable.
The first row is the horizontal variable (dhj, padded by a void 0.0 at the beginning).
The first column is the vertical variable (dvi). Both variables have to be sorted ascendingly.
The values inside the table (rij) represent the corresponding results.

```
begin_curve{ $CurveName$ }, depvar_horizontal {...}, depvar_vertical {...}
0.0 dh1 dh2 ... dhm
dv1 r11 r12 ... r1m
dv2 r21 r22 ... r2m
. . . . .
. . . . .
. . . . .
dvn rn1 rn2 ... rnm
end_curve
```

Example: impact angle depending on velocity magnitude and mean diameter

```
begin_curve{ $angle_of_impact$ }, depvar_horizontal{ equn{ $velocity_magnitude$ }}, depvar_vertical{ equn{
$mean_diameter_micrometers$ }}
0.0 0.0 22.0 32.0 45.0 56.0 63.0 77.0
10.0 0.0 0.0 0.0 0.0 0.01 0.01 0.01
54.0 0.0 0.98 1.50 2.66 3.56 4.69 5.93
107.5 0.0 10.42 15.92 28.24 37.90 49.84 63.11
152.5 0.0 32.04 48.94 86.80 116.52 153.22 194.00
215.0 0.0 91.73 140.10 248.51 333.57 438.65 555.40
427.5 0.0 634.38 968.87 1718.59 2306.84 3033.49 3840.88
605.0 0.0 1522.37 2325.07 4124.23 5535.88 7279.68 9217.24
855.0 0.0 3426.73 5233.56 9283.34 12460.85 16386.02 20747.32
end_curve
```

Note:

- Instead of references to equations by equn{...}, also references to MESHFREE variables can be used, i.e. depvar_horizontal{ %ind_...%}.
- Currently, it is not possible to use Equations in the horizontal (dhj) and vertical (dvi) variables, i.e. equn{...} or []. This is only possible for the results (rij).
- However, a ConstructClause can be used to define aliases that can be referenced in the horizontal and vertical variables, c.f. 1D_Curves for an example.

List of members:

depvar_horizontal	defines the index for the horizontal independent variable in 2D curves
depvar_vertical	defines the index for the vertical independent variable in 2D curves

depvar_horizontal

defines the index for the horizontal independent variable in 2D curves

Options:

- Definition based on existing MESHFREE -variables (see Indices) by enclosing %-signs

```
begin_curve{ $...$ }, depvar_horizontal{ %ind_T% }, depvar_vertical {...}
...
end_curve
```

- Definition based on Equations by enclosing the equation name by curly brackets

```
begin_curve{ $...$ }, depvar_horizontal{ equn{ $EQN_radius$ }}, depvar_vertical {...}
...
end_curve
begin_equation{ $EQN_radius$ }
sqrt( Y %ind_x(1)% ^2 + Y %ind_x(2)% ^2 + Y %ind_x(3)% ^2 )
end_equation
```

Note: The options for depvar_horizontal and depvar_vertical can be mixed (i.e. one can use the definition based on existing MESHFREE -variables while the other uses the definition based on Equations).

depvar_vertical

defines the index for the vertical independent variable in 2D curves

Options:

- Definition based on existing MESHFREE -variables (see Indices) by enclosing %-signs

```
begin_curve{ $...$ }, depvar_horizontal {...}, depvar_vertical{ %ind_T% }
...
end_curve
```

- Definition based on Equations by enclosing the equation name by curly brackets

```
begin_curve{ $...$ }, depvar_horizontal {...}, depvar_vertical{ equn{ $EQN_radius$ }}
...
end_curve
begin_equation{ $EQN_radius$ }
sqrt( Y %ind_x(1)% ^2 + Y %ind_x(2)% ^2 + Y %ind_x(3)% ^2 )
end_equation
```

Note: The options for depvar_horizontal and depvar_vertical can be mixed (i.e. one can use the definition based on existing MESHFREE -variables while the other uses the definition based on Equations).

3.1.11. DropletSource

generate a sequence of spherical droplets

```
DropletSource (n) = ( V_dot, sizeOfNewDroplets, xPosOfNewDroplet, yPosOfNewDroplet, zPosOfNewDroplet, iChamber, $Material$, OPTIONAL %DropletSource_doNotCreateDropletsOutside% )
```

n: index of the DropletSource sequence (up to 99)

V_dot: volume flux to be generated by the droplet sequence in m^3/s

sizeOfNewDroplets: volume of next droplet in the sequence in m^3

xPosOfNewDroplet: x-position of next droplet in the sequence

yPosOfNewDroplet: y-position of next droplet in the sequence

zPosOfNewDroplet: z-position of next droplet in the sequence

iChamber: chamber index to which each new droplet of the sequence will have to belong

$Material$: material index to which each new droplet will have to belong

%DropletSource_doNotCreateDropletsOutside% : give this flag to prevent creation of droplets outside of EVENT -cuts, such that V_dot is preserved for the reduced creation area

Example:

```
DropletSource (1) = ( 5, [ &Hmax& ^3], [20*rand(1)], [1.5*rand(-1)], [2], 1, $Mat1$ ) # the droplet positions to be created are random: 0 < x < 20
# -1.5 < y < 1.5
# z = 2
```

In order to generate a unique sequence of droplets, the functionalities given in real() can be used, especially in TwoArguments the options

- %DropletSource_provideCounter% ,
- %DropletSource_provideTargetVolume% ,
- %DropletSource_provideCurrentVolume% .

REMARK: Radius correction
From the volume V given by the user, we compute the radius r of the sphere classically by

$$r = \left(\frac{3V}{4\pi}\right)^{\frac{1}{3}}$$

However, taking into account that the volume of the discrete particle sphere will be less (linear approximation of a convex, curved manifold), we correct the radius by

$$\tilde{r} = r \cdot \left(2 - \sqrt{1 - \left(\frac{\alpha}{2}\frac{h}{r}\right)^2}\right)$$

where α is the value of radius_hole and h is the current smoothing length.

DOWNLOAD COMPREHENSIVE EXAMPLE

3.1.12. EVENT

events defined for the point cloud

An EVENT is a feature, that evaluates a **event_trigger_expression** on all MESHFREE points at the beginning of the timestep. If the trigger expression is evaluated positively an action is performed. The action to be performed is defined by the type of event and the following types are available:

1.) Function manipulation (can be used e.g. for the rotation of points hitting a certain part of the boundary geometry)
2.) Deletion of points (can be used for "metageometries")
3.) Stop MESHFREE and exit cleanly
4.) Abort MESHFREE with an error
5.) Display an event message
6.) Write a restart file independent of the definition given in RestartStepSize
7.) Write a resume file
8.) Save computational results independent of the definition given in SAVE_interval

In USER_common_variables the definition of an event looks as follows:

```
EVENT ( $EvInd1$ ) = ( event_trigger_expression, %EVENT_FunctionManipulation% , %ind_xyz%, expression_xyz
[,%ind_abc%, expression_abc ...] )
EVENT ( $EvInd2$ ) = ( event_trigger_expression, %EVENT_DeletePoint% , OPTIONAL:MessageCode )
EVENT ( $EvInd3$ ) = ( event_trigger_expression, %EVENT_StopFPM% , OPTIONAL:MessageCode )
EVENT ( $EvInd4$ ) = ( event_trigger_expression, %EVENT_AbortFPM% , OPTIONAL:MessageCode )
EVENT ( $EvInd5$ ) = ( event_trigger_expression, %EVENT_Msg% , MessageCode )
EVENT ( $EvInd6$ ) = ( event_trigger_expression, %EVENT_WriteRestart% , OPTIONAL:MessageCode )
EVENT ( $EvInd7$ ) = ( event_trigger_expression, %EVENT_WriteResume% , OPTIONAL:MessageCode )
EVENT ( $EvInd8$ ) = ( event_trigger_expression, %EVENT_SaveResults% , OPTIONAL:MessageCode )
```

For each MESHFREE point, the **event_trigger_expression** is evaluated. If it is larger than zero for the considered point, the event is triggered and the action is performed.

The event_trigger_expression as well as the manipulations for the indices are defined by Equations .

The (optional) MessageCode is a non-negative integer, that associates the event with an EventMessage .
In case the event is triggered within a timestep the EventMessage is printed to the output and written to the warnings-file (once per timestep).
In this way, the user can check on the triggering of the defined events.

Good to know

- The soft variables on the left hand side of the definition are optional. If none is given, then MESHFREE counts the number of event statements by their appearance in USER_common_variables .
  Warning: The syntax with and without soft variables must not be mixed.
- Instead of a soft variable $EvInd$, also the legacy syntax with natural number n is possible. In this case, all event statements in USER_common_variables have to be numbered consecutively to prevent overwriting.
- Event types 3 and 4 can be used for further stopping criteria besides time and number of time steps.
- Additional feature (for performance): Execute the event handler to execute the particular event only every **N_CycEvent** time steps by prepending
  the additional parameters %EVENT_PerformAfterHowManyTimeCycles% and N_CycEvent to the RightHandSideExpression .

  ```
  EVENT ( $EvInd9$ ) = ( %EVENT_PerformAfterHowManyTimeCycles% , N_CycEvent, event_trigger_expression, %EVENT_...%, ... )
  ```

- Currently, it is possible to define 40 EVENT definitions.

List of members:

%EVENT_PerformAfterHowManyTimeCycles%	cycle of event execution
%EVENT_FunctionManipulation%	pointwise function manipulation event handle
%EVENT_DeletePoint%	deletion of point event handle
%EVENT_StopFPM%	stop MESHFREE event handle
%EVENT_AbortFPM%	abort MESHFREE event handle
%EVENT_Msg%	print message event handle
%EVENT_WriteRestart%	write restart event handle
%EVENT_WriteResume%	write resume event handle
%EVENT_SaveResults%	save computational results event handle
EventMessage	event message with message code

%EVENT_AbortFPM%

abort MESHFREE event handle

```
EVENT ( $EvInd1$ ) = ( event_trigger_expression, %EVENT_AbortFPM% , OPTIONAL:MessageCode )
```

For each MESHFREE point the **event_trigger_expression** is evaluated. If it is larger than zero for the considered point, the event is triggered
and MESHFREE is aborted with an error.

The event_trigger_expression is defined by Equations .

The optional MessageCode is a non-negative integer, that associates the event with an EventMessage , which is printed once if that event has been triggered.

%EVENT_DeletePoint%

deletion of point event handle

```
EVENT ( $EvInd1$ ) = ( event_trigger_expression, %EVENT_DeletePoint% , OPTIONAL:MessageCode )
```

For each MESHFREE point the **event_trigger_expression** is evaluated. If it is larger than zero for the considered point, the event is triggered and the point is deleted.

The optional MessageCode is a non-negative integer, that associates the event with an EventMessage , which is printed once if that event has been triggered.

The event_trigger_expression is defined by Equations .

Note:

- If points on and in the vicinity of BoundaryElements (geometry) are deleted at the same time by such an EVENT , try to do it orthogonal to the geometry. Unphysical behavior might be observed otherwise.
- If surface tension sigma > 0 and thin films are deleted by such an EVENT , the classical free surface boundary condition
  for the hydrostatic pressure should be replaced by a Dirichlet condition in the vicinity of the EVENT . Without this adaption, the thin films close to the EVENT might swell unphysically.
  Example:

```
...
EVENT ( $EvInd1$ ) = ( [if(Y %ind_x(1)% > 0.5) :: 1.0 else :: 0.0 endif], %EVENT_DeletePoint% )
...
BC_p (0) = ( [if(Y %ind_x(1)% > 0.5-0.1*Y %ind_h% ) :: %BND_DIRICH% else :: %BND_free_implicit% endif],
[if(Y %ind_x(1)% > 0.5-0.1*Y %ind_h% ) :: 0 else :: 0 endif] )
...
```

%EVENT_FunctionManipulation%

pointwise function manipulation event handle

```
EVENT ( $EvInd1$ ) = ( event_trigger_expression, %EVENT_FunctionManipulation% , %ind_xyz%, expression_xyz
[,%ind_abc%, expression_abc ...] )
```

For each MESHFREE point the **event_trigger_expression** is evaluated. If it is larger than zero for the considered point, the event is triggered and
for the given indices (**%ind_xyz%** , **%ind_abc%** , ...) the defined function manipulations (**expression_xyz** , **expression_abc** , ...) are executed.

The event_trigger_expression as well as the manipulations for the indices are defined by Equations .

The optional MessageCode is a non-negative integer, that associates the event with an EventMessage , which is printed once if that event has been triggered.

See %ind_event_FunctionManipulation% for further information.

Note: A function manipulation event is classified as a geometrical function manipulation event, if it changes at least one of:

- %ind_x(1)% , %ind_x(2)% , %ind_x(3)%
- %ind_n(1)% , %ind_n(2)% , %ind_n(3)%
- %ind_kob%
- %ind_sha(1)% , %ind_sha(2)% , %ind_sha(3)% , %ind_sha(4)%
- %ind_BC%

Points that have been influenced by a geometrical function manipulation event are marked for the free surface check irrelevant of their current kob-value (%ind_kob%).

See %ind_event_GeometricalFunctionManipulation% for further information.

MESHFREE · InputFiles · USER_common_variables · EVENT · %EVENT_Msg%

%EVENT_Msg%

print message event handle

```
EVENT ( $EvInd1$ ) = ( event_trigger_expression, %EVENT_Msg% , MessageCode )
```

For each MESHFREE point the **event_trigger_expression** is evaluated. If it is larger than zero for the considered point, the event is triggered
and the defined **MessageCode** is printed once. MessageCode is a non-negative integer, that associates the event with an EventMessage .

MESHFREE · InputFiles · USER_common_variables · EVENT · %EVENT_PerformAfterHowManyTimeCycles%

%EVENT_PerformAfterHowManyTimeCycles%

cycle of event execution

Force the event handler to execute an event only every **N_CycEvent** time steps by:

```
EVENT ( $EvInd1$ ) = ( %EVENT_PerformAfterHowManyTimeCycles% , N_CycEvent, event_trigger_expression, %EVENT_...%, ... )
```

The two additional, optional parameters (%EVENT_PerformAfterHowManyTimeCycles% , N_CycEvent) have to come at the beginning of the RightHandSideExpression .

MESHFREE · InputFiles · USER_common_variables · EVENT · %EVENT_SaveResults%

%EVENT_SaveResults%

save computational results event handle

```
EVENT ( $EvInd1$ ) = ( event_trigger_expression, %EVENT_SaveResults% , OPTIONAL:MessageCode )
```

For each MESHFREE point the **event_trigger_expression** is evaluated. If it is larger than zero for the considered point, the event is triggered
and the computational results are saved independent of the definition given in SAVE_interval .

The event_trigger_expression is defined by Equations .

The optional MessageCode is a non-negative integer, that associates the event with an EventMessage ,

which is printed once if that event has been triggered.

MESHFREE · InputFiles · USER_common_variables · EVENT · %EVENT_StopFPM%

%EVENT_StopFPM%

stop MESHFREE event handle

```
EVENT ( $EvInd1$ ) = ( event_trigger_expression, %EVENT_StopFPM% , OPTIONAL:MessageCode )
```

For each MESHFREE point the **event_trigger_expression** is evaluated. If it is larger than zero for the considered point, the event is triggered
and MESHFREE is stopped (clean normal exit).

The event_trigger_expression is defined by Equations .

The optional MessageCode is a non-negative integer, that associates the event with an EventMessage ,
which is printed once if that event has been triggered.

MESHFREE · InputFiles · USER_common_variables · EVENT · %EVENT_WriteRestart%

%EVENT_WriteRestart%

write restart event handle

```
EVENT ( $EvInd1$ ) = ( event_trigger_expression, %EVENT_WriteRestart% , OPTIONAL:MessageCode )
```

For each MESHFREE point, the **event_trigger_expression** is evaluated. If it is larger than zero for the considered point, the event is triggered
and a restart file is written independent of the definition given in RestartStepSize .

The event_trigger_expression is defined by Equations .

The optional MessageCode is a non-negative integer, that associates the event with an EventMessage ,
which is printed once if that event has been triggered.

Note: In case of using %RESTART_sequence% to define the RestartStepSize , the user can limit the number of kept restart
files triggered by EVENT similarly to the number of kept restart files triggered by standard. See %RESTART_sequence% for details.

MESHFREE · InputFiles · USER_common_variables · EVENT · %EVENT_WriteResume%

%EVENT_WriteResume%

write resume event handle

```
EVENT ( $EvInd1$ ) = ( event_trigger_expression, %EVENT_WriteResume% , OPTIONAL:MessageCode )
```

For each MESHFREE point the **event_trigger_expression** is evaluated. If it is larger than zero for the considered point, the event is triggered
and a resume file is written. See checkpoint for details.

The event_trigger_expression is defined by Equations .

The optional MessageCode is a non-negative integer, that associates the event with an EventMessage ,

which is printed once if that event has been triggered.

EventMessage

event message with message code

Define an event message for a message code which can be used in EVENT statements.

```
EventMessage (MessageCode) = "MessageText"
```

Example:

```
EventMessage (12345) = "MESHFREE was stopped due to an event."
```


3.1.13. Equations

define functions, equations, and algebraic expressions

In most positions within the USER_common_variables in the RightHandSideExpression it is possible to include user defined equations, e.g. into boundary conditions, initial conditions, user defined variables, and many more. The equation is then evaluated on point basis, in particular the equation is automatically evaluated for each the statement on the left hand side concerning point. Equations can be defined and invoked in the following ways:

(explicit) Definition - UseByReference

An equation definition for a user chosen Reference Name $EquationName$ and a user chosen BodyOfEquation takes the general form:

```
begin_equation{ $EquationName$ }
BodyOfEquation
end_equation
```

BodyOfEquation

Many of the equations are evaluated on point basis. The values of physical and organizational quantities can be accessed by the **Y-Syntax** by an index from Indices

```
Y%index%
```

Example 1: kinetic energy of a MESHFREE point uses pointwise quantities

```
begin_equation{ $KineticEnergy$ }
0.5*Y %ind_r% *( Y %ind_v(1)% ^2 + Y %ind_v(2)% ^2 + Y %ind_v(3)% ^2 )
end_equation
```

The BodyOfEquation can moreover incorporate:

- __Constants__ : Meshfree internal constants, e.g. %BND_free% can be compared to Y %ind_kob% to evaluate if a point belongs to a free surface, see Example 2.
- Functions : refer to functions such as cos() and many more.
- Operators for comparisons or elementary calculations.

Example 2: boolean returning 1 if point is a free surface point

```
begin_equation{ $IsFreeSurface$ }
if (Y %ind_kob% = %BND_free% ) :: 1
else :: 0
endif
end_equation
```

Referencing

These equations can be referenced by their $EquationName$ in two ways:

- directly on the RightHandSideExpression of statements by **equn{$EquationName$}** , see Example 3.
- within another equation definition by using the Function **equn($EquationName$)** , see Example 4.
- Inside a curve definition (Curves), see Example 5.

Examples (Referencing)

Example 3: Dirichlet temperature boundary condition with temperature given by the evaluation of the equation

```
BC_T ( $wall$ ) = ( %BND_DIRICH% , equn{ $EquationName$ })
```

Example 4: Referring to an equation from another equation:

```
begin_equation{ $AnotherEquation$ }
... equn( $EquationName$ ) ...
end_equation
```

Example 5: Referring from a curve to an equation:

```
begin_curve{ $CurveName$ }
0 equn{ $EquationName$ }
1 2
3 87.5
end_curve
```

Inline definition of equations

If the equation is not so complicated and only used on one location within the setup, then there is a comfortable way of defining the equation inline by using the **inline square bracket []-syntax** .

Example 6

```
BC_T ( $wall$ ) = ( %BND_DIRICH% , [BodyOfEquation])
```

Equations for boundary elements

Most equations are evaluated for the pointcloud, but we also have a limited amount of functions, that can be used in the context of boundary elements.

(e.g. for SAVE_BE_ITEM): these are all functions starting with BE* in the list of Functions .

List of members:

Functions	standard math functions and MESHFREE-specific functions
Operators	standard math operators

Functions

standard math functions and MESHFREE-specific functions

See the list below.

List of members:

abs()	absolute value
acos()	inverse cosine
approxY()	approximation of a MESHFREE-entity by the MESHFREE least squares operators
asin()	inverse sine
atan()	inverse tangent
BE_n()	normal with respect to a boundary element
BEarea()	area of a boundary element
BEgauss()	BE local Gaussian curvature
BEhasCurv()	1 if curvature computation is successful
BEincidence()	number of incidental edges of a node point
BEisOnEdge()	1 if boundary node belongs to an edge
BEmap()	Fetch result of mapping onto boundary element
BEmaxCurv()	BE local maximum curvature
BEminCurv()	BE local minimum curvature
BEmon()	BE monitor item results
BEpos()	midpoint, minimum or maximum position of a boundary element
BEprincipalCurvatureEdge1()	first edge of principal curvature computation
BEprincipalCurvatureEdge2()	second edge of principal curvature computation
BEprincipalCurvatureEdge3()	third edge of principal curvature computation
BEprincipalCurvatureEdge4()	fourth edge of principal curvature computation
BEprincipalCurvatureNormal()	normal for principal curvature computation
BEsum()	summation over values given on boundary elements
binA()	step function for alias
ChkNP()	check for attributes of node points of boundary elements
CID()	CuttingCurveCluster ID

compareY()	compare function values between two given chambers
cos()	cosine
cosh()	hyperbolic cosine
cross()	flag if point crossed a BND_BlindAndEmpty boundary element in the current time step
curve()	incorporate curves in an equation
dcurv()	derivative of a given curve
dequn()	derivative of a given equation
dtBND()	(experimental) closest distance to boundary (free surface or regular) in the neighborhood of a MESHFREE point
DtDom()	distance to a given alias-domain
dYdn()	normal derivative of MESHFREE-entity
dYdx()	x-derivative of MESHFREE-entity
dYdy()	y-derivative of MESHFREE-entity
dYdz()	z-derivative of MESHFREE-entity
eigen()	eigenvalues and eigenvectors of a symmetric 3x3 matrix
equn()	incorporate existing equations
ExDom()	check if a point is outside a closed domain
exp()	exponential
fABND()	function evaluation for monitor points relative to the area of the corresponding boundary element
FCOG()	integrated forces acting on the center of gravity for a given MOVE-flag
if-then-else	logical branching in an equation
InDom()	check if a point is inside a closed domain
int()	integer part of a real value
integ()	incorporate integration results in an equation
isCID()	characteristic function for a CuttingCurveCluster
joint()	provide general information of a given rigid body being in joint/link-contact with other bodies
LCOG()	integrated/rotated local coordinate system of a rigid body of a given MOVE-flag
lenA()	length of alias string
log()	natural logarithm
log10()	logarithm with basis 10
max()	maximum of two or more arguments
MCOG()	moment about of the center of gravity for a given MOVE-flag

min()	minimum of two or more arguments
mod()	modulo operation
nbsum()	sum over points in neighbor list
nrand()	random sample from a normal distribution
ode()	incorporate results of ODE solvers
omCOG()	rotational speed of the center of gravity for a given MOVE-flag
phix()	
phiy()	
phiz()	
pmin()	minimum of all strictly positive values
projY()	projection of a MESHFREE-entity by smooth, Shepard-type approximation
rand()	random number generator
RasterCircleX	x-coordinate of a random midpoint of a raster of squares with respect to a circle
RasterCircleY	y-coordinate of a random midpoint of a raster of squares with respect to a circle
real()	incorporate standard MESHFREE-postprocessing and statistics
reduct()	incorporate results of PointCloudReduction operation
rot()	rotated vector
sin()	sine
sinh()	hyperbolic sine
sodst()	provide solution to sods shock tube problem
sqrt()	square root
step()	(unit) step function
tan()	tangent
tanh()	hyperbolic tangent
vCOG()	velocity of the center of gravity for a given MOVE-flag
xCOG()	position of the center of gravity for a given MOVE-flag
Y0()	MESHFREE-entity
Yopp()	MESHFREE-entity of the opposite MESHFREE point in contact problems

BE_n()

normal with respect to a boundary element

Computes the normal of a boundary element of a fixed boundary, e.g. of a triangle. As the function needs to return a scalar, the x-, y-, or z-component of the normal is selected by providing 1, 2, or 3 as argument, respectively.

This is useful when used together with BEsum() in the context of MOVE statements or within a BE_MONITOR_ITEM or SAVE_BE_MONITOR_ITEM .

Example:

```
begin_equation{ $normal_x$ }
BE_n(1)
end_equation
begin_equation{ $normal_y$ }
BE_n(2)
end_equation
begin_equation{ $normal_z$ }
BE_n(3)
end_equation
```


BEarea()

area of a boundary element

Computes the area of a boundary element of a fixed boundary, e.g. of a triangle.

This is useful when used together with BEsum() in the context of MOVE statements or within a BE_MONITOR_ITEM or SAVE_BE_MONITOR_ITEM .

Example:

```
begin_equation{ $EqunName$ }
... BEarea(1) ...
end_equation
```

Note: BEarea() needs a dummy argument (in the example, 1). So far, its value is ignored.

BEgauss()

BE local Gaussian curvature

Computes the local Gaussian curvature of a boundary element node. If the curvature cannot be computed 0 is returned. Use BEhasCurv() to to check for success.

```
[ ... BEgauss(1) ... ]
```

This function requires a dummy parameter which is ignored.

BEhasCurv()

1 if curvature computation is successful

Returns 1 if curvature computations by BEminCurv() , BEmaxCurv() , BEgauss() , BEprincipalCurvatureEdge1() , BEprincipalCurvatureEdge2() , BEprincipalCurvatureEdge3() BEprincipalCurvatureEdge4() are successful. Otherwise 0 is returned.

```
[ ... BEhasCurv(1) ... ]
```

This function requires a dummy parameter which is ignored.

BEincidence()

number of incidental edges of a node point

Returns the number of incidental edges of a node point.

```
[ ... BEincidence(1) ... ]
```

This function requires a dummy parameter which is ignored.

BEisOnEdge()

1 if boundary node belongs to an edge

Returns 1 if the node point of a boundary element belongs to an edge. Otherwise 0 is returned.

```
[ ... BEisOnEdge(1) ... ]
```

This function requires a dummy parameter which is ignored.

BEmap()

Fetch result of mapping onto boundary element

This function fetches the result of a BE_MAP command for the current boundary element.

Example 1 : Map hydrostatic and dynamic pressure to the boundary and save both pressures as well as the total pressure for each BE

```
SAVE_BE_ITEM = ( %SAVE_scalar%, [ BEmap( $BEmap_phyd$ ) ], "BE_BEmap_phyd" )
SAVE_BE_ITEM = ( %SAVE_scalar%, [ BEmap( $BEmap_pdyn$ ) ], "BE_BEmap_pdyn" )
SAVE_BE_ITEM = ( %SAVE_scalar%, [ BEmap( $BEmap_phyd$ ) + BEmap( $BEmap_pdyn$ ) ], "BE_BEmap_ptot" )
BE_MAP ( $BEmap_phyd$ ) = ( [ Y %ind_p% ] )
BE_MAP ( $BEmap_pdyn$ ) = ( [ Y %ind_p_dyn% ] )
```

Example 2 : Directly map the total pressure to the boundary and save total pressure for each BE

```
SAVE_BE_ITEM = ( %SAVE_scalar%, [ BEmap( $BEmap_ptot$ ) ], "BE_BEmap_ptot" )
BE_MAP ( $BEmap_ptot$ ) = ( [ Y %ind_p% + Y %ind_p_dyn% ] )
```

Note: The function BEmap() should currently only be used in conjunction with SAVE_BE_ITEM

BEmaxCurv()

BE local maximum curvature

Computes the local maximum curvature of a boundary element node. If the curvature cannot be computed 0 is returned. Use BEhasCurv() to to check for success.

```
[ ... BEmaxCurv(1) ... ]
```

This function requires a dummy parameter which is ignored.

BEminCurv()

BE local minimum curvature

Computes the local minimum curvature of a boundary element node. If the curvature cannot be computed 0 is returned. Use BEhasCurv() to to check for success.

```
[ ... BEminCurv(1) ... ]
```

This function requires a dummy parameter which is ignored.

BEmon()

BE monitor item results

Definition of a BE_MONITOR_ITEM :

```
BE_MONITOR_ITEM (n) = ( %CUMU_...%, ... )
```

The result of this particular monitor item can be used inside of an equation by:

```
[ ... BEmon(n) ... ]
```

n is the index of the monitor item.

BEpos()

midpoint, minimum or maximum position of a boundary element

Computes the midpoint, minimum or maximum of a boundary element of a fixed boundary, e.g. of a triangle.

```
[ ... BEpos(n, s) ... ]
```

n = 1,2,3; selects the x-, y-, z-component respectively
s = 0, positive value, negative value; specifies the position - midpoint, maximum or minimum respectively

This is useful when used together with INTEGRATION .

Example:

```
begin_equation{ $midpoint_x$ }
BEpos(1, 0)
end_equation
begin_equation{ $maximum_y$ }
BEpos(2, 1)
end_equation
begin_equation{ $minimum_z$ }
BEpos(3, -2)
end_equation
```


BEprincipalCurvatureEdge1()

first edge of principal curvature computation

Returns the n-th coordinate component of the vector pointing along the first edge used in principal curvature computations.

```
[ ... BEprincipalCurvatureEdge1(n) ... ]
```

n=1,2,3 the vector component

BEprincipalCurvatureEdge2()

second edge of principal curvature computation

Returns the n-th coordinate component of the vector pointing along the second edge used in principal curvature computations.

```
[ ... BEprincipalCurvatureEdge2(n) ... ]
```

n=1,2,3 the vector component

BEprincipalCurvatureEdge3()

third edge of principal curvature computation

Returns the n-th coordinate component of the vector pointing along the third edge used in principal curvature computations.

```
[ ... BEprincipalCurvatureEdge3(n) ... ]
```

n=1,2,3 the vector component

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · BEprincipalCurvatureEdge4()

BEprincipalCurvatureEdge4()

fourth edge of principal curvature computation

Returns the n-th coordinate component of the vector pointing along the fourth edge used in principal curvature computations.

```
[ ... BEprincipalCurvatureEdge4(n) ... ]
```

n=1,2,3 the vector component

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · BEprincipalCurvatureNormal()

BEprincipalCurvatureNormal()

normal for principal curvature computation

Returns the n-th coordinate component of the vector pointing along the normal used in principal curvature computations.

```
[ ... BEprincipalCurvatureNormal(n) ... ]
```

n=1,2,3 the vector component

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · BEsum()

BEsum()

summation over values given on boundary elements

For a node point of a fixed boundary all neighboring boundary elements, e.g. triangles, are collected.
The equation provided as argument is evaluated on each of these neighboring boundary elements
and finally summed up.

So far, BEsum only makes sense when used on nodes of the fixed boundary, e.g. in the context of a MOVE statement.

Example:

```
begin_equation{ $EqunName$ }
... BEsum( $eq_sum$ ) ...
end_equation
begin_equation{ $eq_sum$ }
...
end_equation
```

Warning: BEsum() can only have a reference to another equation. It is not possible to write down values or any mathematical expressions directly.

CID()

CuttingCurveCluster ID

```
begin_equation{ $EqunName$ }
... CID(0) ...
end_equation
```

The CuttingCurveCluster ID for points on the boundary is returned. For non-boundary points the result is 0.

Note: CID() needs a dummy argument (in the example, 0). So far, its value is ignored.

ChkNP()

check for attributes of node points of boundary elements

With the help of this function different attributes of the node points of the boundary elements belonging to a given alias can be checked.

Example:

```
begin_equation{ $EqunName$ }
... ChkNP("AliasName", attribute, component, type) ...
end_equation
```

"AliasName" specifies to which alias the boundary elements belong.

attribute specifies which attribute should be considered:
- 1 (position, X_BND)
- 2 (velocity, V_BND)
- 3 (acceleration, Vdot_BND)

component specifies which component of the given attribute should be considered:
- 1 (x-coordinate)
- 2 (y-coordinate)
- 3 (z-coordinate)

type specifies which type of check should be done:
- 1 (average with respect to the number of node points matching the given alias)
- 2 (minimum)
- 3 (maximum)

DtDom()

distance to a given alias-domain

```
begin_equation{ $EqunName$ }
... DtDom("AliasName") ...
end_equation
```

MESHFREE will compute the distance of a given point to the boundary elements (BE) attached to the alias **"AliasName"** .

Also the orientation of the BE plays a role, such that the distance can become negative,
if the point is logically outside of the domain.

Optionally, instead of computing the distance to MESHFREE points, compute the distance to
any given coordinate:

```
begin_equation{ $EqunName$ }
... DtDom("AliasName", x, y, z) ...
end_equation
```

The distance is computed with respect to the point (**x** , **y** , **z**), i.e. DtDom("AliasName") and
DtDom("AliasName", Y %ind_x(1)% , Y %ind_x(2)% , Y %ind_x(2)%) are equivalent.

Note: The algorithm is expensive, since MESHFREE compares the point with each BE given by "AliasName".
So, use this function with caution.

ExDom()

check if a point is outside a closed domain

Returns the opposite of InDom() , i.e. 0 if inside, 1 if outside.

FCOG()

integrated forces acting on the center of gravity for a given MOVE-flag

```
[ ... FCOG(i, $MOVEFlag$ , OPTIONAL: iWhat ) ... ]
```

- **i** = 1,2,3 yields the x-, y-, z-component of the forces for the given **$MOVEFlag$** , respectively.
- $MOVEFlag$ is directly associated to all boundary elements carrying this MOVE -flag.
- **IWhat** (DEFAULT=0) :
  0 => sum of EXTERNAL forces (given by %MOVE_Rigid% and/or RIGIDBODY_ExternalForces) + pressure/tension forces acting om body + gravity forces
  1 => sum of pressure/tension forces + gravity forces
  2 => simply sum of EXTERNAL forces
  3 => pressure and tension forces

InDom()

check if a point is inside a closed domain

```
begin_equation{ $EqunName$ }
... InDom("AliasName") ...
end_equation
```

For a MESHFREE point the InDom-check returns 1 if the point is inside the closed domain given by the boundary elements (BE) attached to the alias **"Alias name"** and 0 if it is outside.

To do the InDom-check, MESHFREE sends a ray from the point. If the ray cuts an even number of times the boundary, the point is outside, otherwise inside.

Optionally, instead of checking the MESHFREE points, any given coordinate can be checked:

```
begin_equation{ $EqunName$ }
... InDom("AliasName", x, y, z) ...
end_equation
```

The InDom-check is performed with respect to the point (**x** , **y** , **z**).

See also ExDom() .

LCOG()

integrated/rotated local coordinate system of a rigid body of a given MOVE-flag

```
[ ... LCOG(i,j, $MOVEFlag$ ) ... ]
```

i = 1,2,3; yields the x-, y-, z-component of the j-th unit vector of the given **$MOVEFlag$** , respectively.
j = 1,2,3; determines the indes of the local unit vector of the local coordinate system associated with the rigid body **$MOVEFlag$** movemebt flag of the rigid body.

REMARK: the original local coordinate system are the eigen vectors of the tensor of inertia to be given in the MOVE -declaration

MCOG()

moment about of the center of gravity for a given MOVE-flag

```
[ ... MCOG(i, $MOVEFlag$ , OPTIONAL: iWhat ) ... ]
```

- **i** = 1,2,3 yields the x-, y-, z-component of the rotational speed ω of the center of gravity for the given **$MOVEFlag$** , respectively.
- $MOVEFlag$ is directly associated to all boundary elements carrying this MOVE -flag.
- **IWhat** (DEFAULT=0) :
  0 => sum of EXTERNAL moments around the x C O G ( ) (given by %MOVE_Rigid% and/or RIGIDBODY_ExternalForces) + moments due to pressure/tension forces acting om body
  1 => sum of moments due to pressure/tension
  2 => sum moments due to all EXTERNAL forces
  3 => sum of moments due to pressure and tension forces (same as 1, but try to keep consistency with FCOG() .

RasterCircleX

x-coordinate of a random midpoint of a raster of squares with respect to a circle

```
[ ... RasterCircleX (r1,r2) ... ]
```

A rectangular raster of squares with edge length **r1** and a circle with radius **r2** are constructed.
The function returns the x-coordinate of a random midpoint of one of the squares that is fully contained in the circle.

RasterCircleY

y-coordinate of a random midpoint of a raster of squares with respect to a circle

```
[ ... RasterCircleY (r1,r2) ... ]
```

A rectangular raster of squares with edge length **r1** and a circle with radius **r2** are constructed.
The function returns the y-coordinate of a random midpoint of one of the squares that is fully contained in the circle.

Y0()

MESHFREE-entity

```
[ ... Y0(%ind_NameOfEntity%, OPTIONAL: iIndex ) ... ]
```

Without optional index, this is equivalent to [... Y%ind_NameOfEntity% ...].
WITH optional index, it returns the appropriate value of the MESHFREE point with the index iIndex.

Yopp()

MESHFREE-entity of the opposite MESHFREE point in contact problems

```
[ ... Yopp(%ind_NameOfEntity%) ... ]
```


abs()

absolute value

```
[ ... abs(a) ... ]
```

Computes the absolute value of **a** .

acos()

inverse cosine

[... acos(a) ...]

Computes the inverse cosine of **a** . The result is in radians.

approxY()

approximation of a MESHFREE-entity by the MESHFREE least squares operators

Approximation of given discrete function values by the MESHFREE least squares approximation,
i.e. MESHFREE uses the classical least-squares approximation stencil at the current (MESHFREE point) location,
or optionally at any user-provided location $\mathbf{x}$, in order to provide the following approximation:

$$\tilde{u}(\mathbf{x}) = \sum_j c_j^0(\mathbf{x}) \cdot u_j$$

The function approxY() , optionally, provides derivatives in the sense

$$\tilde{\partial}^* u(\mathbf{x}) = \sum_j c_j^*(\mathbf{x}) \cdot u_j$$

where '*' stands for x, y, or z derivatives.

See DOCUMATH_DifferentialOperators.pdf for a complete description of the least-squares idea,
especially refer to chapter 1.

[... approxY(%ind_u% , iChamber , OPTIONAL: iOrder ,
OPTIONAL: alphaKernel,
OPTIONAL: whatToApproximate,
OPTIONAL: xApprox, yApprox, zApprox,
OPTIONAL: factor_allowed_overshoot) ...]

- **%ind_u%** : index of the function to be approximated
- **iChamber** : approximation in what chamber; *default* : the chamber index of the current MESHFREE point Y %ind_cham%
- **iOrder** : order of approximation (1,2,3); *default* : the order given in ord_gradient
- **alphaKernel** : specify α in the kernel/weight function $W(\mathbf{x}_j, \mathbf{x}) = \exp\left(-\alpha \frac{\|\mathbf{x}_j - \mathbf{x}\|^2}{h(\mathbf{x}_j)^2}\right)$; *default* : given by DIFFOP_kernel_Gradient
- **whatToApproximate** : 0 (function), 1 (x-derivative), 2 (y-derivative), 3 (z-derivative); *default* : 0
- **xApprox, yApprox, zApprox** : define the location where to do the approximation; *default* : location of current MESHFREE point (Y %ind_x(1)% , Y %ind_x(2)% , Y %ind_x(3)%)
- **factor_allowed_overshoot** : activate and define the factor α for the allowed overshoot of the approximation: 0 (no limit for overshoot), $0 < \alpha \leq 1$ (internally programmed values), $\alpha > 1$ (user defined factor); *default* : 0

Further remarks:
- DIFFOP_Version triggers the approximation method
- The smoohting length / interaction radius $h(\mathbf{x}_j)$ is used from the appropriate SmoothingLength defintions set forth to chamber **iChamber**

Important Remark : given an approximation task in iChamber at the location $\boldsymbol{x}$, then MESHFREE will search for the closest neighbor point at location $\boldsymbol{x}_i$ in iChamber. The neighbors for the approximation task around $\boldsymbol{x}$ will be executed using the neighbor list of $\boldsymbol{x}_i$. Thus, the choice of the parameter NEIGHBOR_FilterMethod will have a big impact on the results of the approximation. We remember that, using NEIGHBOR_FilterMethod > 1, we prevent the
neighbor search from "looking through" thin walls.

Experts only : Two-digit mode for iOrder :
Instead of specifying a single digit for **iOrder** , there is the option to specify a two digit parameter that controls which points are considered for the approximation:

	Approximation Order 1	Approximation Order 2	Approximation Order 3
interior and free surface particles (Y%ind_kob%=%BND_none% or Y%ind_kob%=%BND_free%)	11	12	13
use only regular boundary particles (without free surface)	21	22	23
use only interior particles (Y%ind_kob%=%BND_none%)	31	32	33
use only boundary particles (including free surfaces)	41	42	43

asin()

inverse sine

```
[ ... asin(a) ... ]
```

Computes the inverse sine of **a** . The result is in radians.

atan()

inverse tangent

```
[ ... atan(a) ... ]
```

Computes the inverse tangent of **a** . The result is in radians.

binA()

step function for alias

The binA() function offers different options for retrieving alias related quantities on the pointcloud and the boundary elements.

- for a point of the pointcloud: evaluating if the boundary element of the MESHFREE point corresponds to a given alias or alias wildcard expression.
- for a point of the pointcloud: retrieving the ALIAS -index of the boundary element the MESHFREE point belongs to.
- for a boundary element: retrieving the ALIAS -index.

Evaluation on Alias Name

Syntax:

```
[ ... binA("AliasName") ... ]
```

The result of the evaluation is:

- 1, if the alias to the boundary element corresponding to MESHFREE point is the "AliasName"
- 0, if the alias to the boundary element corresponding to MESHFREE point is **not** the "AliasName"

Example: "AliasName" can also be a wildcard expression: The expression

```
[ ... binA("b*") ... ]
```

will return 1 for all aliasses matching the wildcard expression "b*". So aliasses "bottom" or "box" will be matched, but the alias "top" will not.

Alias Index to point

The construct

```
[ ... binA(0) ... ]
```

will deliver the ALIAS -index of the boundary element, the MESHFREE point belongs to.

Alias Index to boundary element

The construct

```
[ ... binA(-iBE) ... ]
```

will deliver the alias index of the boundary element with the index iBE.
HINT: if the point is a boundary point, then binA(0) and binA(-Y %ind_BE1%) would result in the same value

---

compareY()

compare function values between two given chambers

```
begin_equation{ $EqunName$ }
... compareY(%ind_f%, iChamber1, iChamber2) ...
end_equation
```

The function values represented by MESHFREE -index **%ind_f%** (see Indices) are compared between the two chambers **iChamber1** and **iChamber2** , i.e. the difference Y(%ind_f%,iChamber1) - Y(%ind_f%,iChamber2) is computed.

The chamber indices have to correspond to defined CHAMBER -flags.

---

cos()

cosine

```
[ ... cos(a) ... ]
```

Computes the cosine of **a** given in radians.

---

cosh()

hyperbolic cosine

```
[ ... cosh(a) ... ]
```

Computes the hyperbolic cosine of **a** .

cross()

flag if point crossed a BND_BlindAndEmpty boundary element in the current time step

```
[ ... cross( $PP_BlindAndEmpty_ID$ ) ... ]
```

This function determines whether a MESHFREE point has crossed a %BND_BlindAndEmpty% -boundary element with POSTPROCESS -flag **$PP_BlindAndEmpty_ID$** in the current time step.

Possible return values (per MESHFREE point):

- +1 if MESHFREE point has crossed from the inside to the outside
- 0 if MESHFREE point has not crossed
- -1 if MESHFREE point has crossed from the outside to the inside

curve()

incorporate curves in an equation

Definition of a curve (see Curves):

```
begin_curve{ $CrvName$ }, depvar_default{ %ind_Var%}
BodyOfCurve
end_curve
```

The result of this curve is used in an equation/arithmetic expression by:

```
[ ... curve( $CrvName$ ) ... ]
```

If the depvar_default{ }-information for the curve is not set, then the independent variable will be the simulation time. See also 1D_Curves .

dYdn()

normal derivative of MESHFREE-entity

derivative in the direction of the boundary normal by the actually installed (local) differential operators

```
[ ... dYdn(%ind_NameOfEntity%) ... ]
```

Note: The normal derivative is only valid for boundary points.

dYdx()

x-derivative of MESHFREE-entity

x-derivative by the actually installed (local) differential operators

```
[ ... dYdx(%ind_NameOfEntity%) ... ]
```


dYdy()

y-derivative of MESHFREE-entity

y-derivative by the actually installed (local) differential operators

```
[ ... dYdy(%ind_NameOfEntity%) ... ]
```


dYdz()

z-derivative of MESHFREE-entity

z-derivative by the actually installed (local) differential operators

```
[ ... dYdz(%ind_NameOfEntity%) ... ]
```


dcurv()

derivative of a given curve

This function numerically computes the derivative of a curve by a central difference.
The numerical differentiation is performed with respect to the depvar_default -variable:

```
begin_equation{ $EqunName$ }
... dcurv( $CurveName$ , ind_MFvariable, OPTIONAL:SizeOfInterval ) ...
end_equation
begin_curve{ $CurveName$ }, depvar_default {...}
...
end_curve
```

ind_MFvariable: If a positive value for ind_MFvariable is given (i.e. an item out of Indices), then the curve is numerically derived with respect to this variable. The standard case, however, is to set ind_MFvariable = -1. In this case, the curve is derived with respect to the default variable which is given in the depvar_default{ }-clause.

SizeOfInterval specifies the half-width of the central difference. The default value is 1.0e-4.

dequn()

derivative of a given equation

This function numerically computes the derivative of an equation by a central difference.
The numerical differentiation is performed with respect to a given MESHFREE -variable **%ind_MFvariable%** :

```
begin_equation{ $EqunName$ }
... dequn( $OtherEqunName$ , %ind_MFvariable%, OPTIONAL:SizeOfInterval ) ...
end_equation
begin_equation{ $OtherEqunName$ }
...
end_equation
```

SizeOfInterval specifies the half-width of the central difference. The default value is 1.0e-4.

---

dtBND()

(experimental) closest distance to boundary (free surface or regular) in the neighborhood of a MESHFREE point

Experimental only!

```
begin_equation{ $EqunName$ }
... dtBND(iArg) ...
end_equation
```

The distance is computed using a least-squares approximation of the distance functional:

- Points at the boundary have distance 0 and a gradient of 1 pointing in normal direction.
- Points in the interior are ignored for versions 0 and 1 (see below).

iArg:

- 0 (distance to free surface)
- 1 (distance to regular boundary based on boundary points only)
- 2 (distance to regular boundary based on %ind_dtb% -information)

---

eigen()

eigenvalues and eigenvectors of a symmetric 3x3 matrix

Produces the eigenvalues and eigenvectors of a symmetric 3x3 matrix.

Syntax:

```
[ ... eigen( M11, M22, M33, M12, M13, M23, iIndex ) ... ]
```

where:

- M11, M22, M33 :: the diagonal elements of the matrix to be considered
- M12, M13, M23 :: the off-diagonal elements of the matrix to be considered (as the matrix is assumed to be symmetric, no need to provide M21, M31, M32.)
- iIndex :: integer between 1 and 12 specifying the component to be returned, see below table.

iIndex	returned component
-1	first eigenvalue
1,2,3	x,y,z-components of the correspondig first eigenvector
-2	second eigenvalue
4,5,6	x,y,z-components of the correspondig second eigenvector
-3	third eigenvalue
7,8,9	x,y,z-components of the correspondig third eigenvector

Good to know:

- The functions in the equation parser generally provide only a **scalar real number** . As the eigen() function provides multiple return parameters, it has to be specified by iIndex, which one is required.
- There is a **caching mode** such that subsequent calls to this function will only recompute the eigenvalues and -vectors if required.
- If multiple eigen() -calls for the multiple matrices are present, then there are two possible strategies for performance optimization: the user can either order the equations appropriately to invoke the caching mode or store the computed values as intermediate result in UserDefinedIndices %indU_...% .

equn()

incorporate existing equations

Definition of an equation (see Equations):

```
begin_equation{ $EqnName$ }
BodyOfEquation
end_equation
```

The result of this equation is used in another equation/arithmetic expression by:

```
[ ... equn( $EqnName$ ) ... ]
```


exp()

exponential

```
[ ... exp(a) ... ]
```

Computes $\exp(a) = e^a$.

fABND()

function evaluation for monitor points relative to the area of the corresponding boundary element

```
[ ... fABND(%ind_f%) ... ]
```

In case of MONITORPOINTS perform the evaluation of a MESHFREE -index **%ind_f%** (see Indices) relative to the area of the boundary element, which the respective monitor point is attached to, in the following sense:

$$\mathrm{fABND}(f) = \frac{\sum_{i \in S_{\mathrm{monitor}}} f_i}{\sum_{i \in S_{\mathrm{monitor}}} A_i},$$

where S_{monitor} denotes the set of all monitor points attached to the same boundary element.
The area of the monitor points A_i is determined by

$$A_i = \frac{A_{iBE}}{np_{iBE}},$$

where A_{iBE} is the area of the boundary element monitor point i is attached to and np_{iBE} is the total number of monitor points attached to this boundary element.

if-then-else

logical branching in an equation

```
begin_equation{ $abs$ }
if(aaa) :: mathexpression1
elseif(bbb) :: mathexpression2
elseif(ccc) :: mathexpression3
else :: mathexpression4
endif
end_equation
```

The logical expressions **aaa** , **bbb** , **ccc** can be established using the logical operators ">", "Example: condition for the x-component of the point position

```
begin_equation{ $abs$ }
if ( Y %ind_x(1)% > 0 ) :: mathexpression1
elseif ( Y %ind_x(1)% > -0.5 ) :: mathexpression2
else :: mathexpression3
endif
end_equation
```

Also nesting is allowed:

```
begin_equation{ $abs$ }
if(aaa) ::
if(ddd) :: mathexpression1
else :: mathexpression2
endif
elseif(bbb) :: mathexpression3
elseif(ccc) :: mathexpression4
else ::
if(ddd) :: mathexpression5
else :: mathexpression6
endif
endif
end_equation
```


int()

integer part of a real value

```
begin_equation{ $EqunName$ }
... int(a) ...
end_equation
```

Computes the integer part of **a** .

integ()

incorporate integration results in an equation

Definition of an INTEGRATION :

```
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_...%, .....)
```

The result of this integration can be used inside of an equation by:

```
[ ... integ( $IntInd$ ) ... ]
```

$IntInd$ is the corresponding soft variable of the integration.

isCID()

characteristic function for a CuttingCurveCluster

```
begin_equation{ $EqunName$ }
... isCID(index_CCC) ...
end_equation
```

For all MESHFREE points on part of the boundary with CuttingCurveCluster ID **index_CCC** 1 is returned, 0 elsewhere.

joint()

provide general information of a given rigid body being in joint/link-contact with other bodies

```
begin_equation{ $EqunName$ }
... joint( iJNT, iItem, iMOVE ) ...
end_equation
```

- **iJNT** : then index of the joint/link between the rigid body and another body. There might be several, the number of which should be known to the user.
- **iItem** : what item to provide by this function; it can be on of the follwoing
  - %EQN_JOINT_x(1)% : x-position of the joint/link
  - %EQN_JOINT_x(2)% : y-position of the joint/link
  - %EQN_JOINT_x(3)% : z-position of the joint/link
  - %EQN_JOINT_F(1)% : x-component of force acting on the joint
  - %EQN_JOINT_F(2)% : y-component of force acting on the joint
  - %EQN_JOINT_F(3)% : z-component of force acting on the joint
  - %EQN_JOINT_M(1)% : x-component of moment acting on the joint
  - %EQN_JOINT_M(2)% : y-component of moment acting on the joint
  - %EQN_JOINT_M(3)% : z-component of moment acting on the joint
- **iMOVE** : move flag of the rigid body, i.e. the $moveName$ given in the MOVE ($moveName$) = (%MOVE_rigid% , ...) statement

Example

```
begin_timestepfile{ "myfile.timestep"}
INTEGRATION (1) = ( %PUBLICVALUE% , [real( %RealTimeSimulation% )], %INTEGRATION_Header%, "time" )
INTEGRATION (2) = ( %PUBLICVALUE% , [joint(1, %EQN_JOINT_x(1)% , $MOVE_RB1$ )],
%INTEGRATION_Header% , "location of joint" )
INTEGRATION (3) = ( %PUBLICVALUE% , [joint(1, %EQN_JOINT_F(1)% , $MOVE_RB1$ )],
%INTEGRATION_Header% , "force at of joint" )
end_timestepfile
```


lenA()

length of alias string

```
[ ... lenA("AliasName") ... ]
```

Determines the length of the given alias "AliasName". If "AliasName" = " ", 0 is returned.

log()

natural logarithm

```
[ ... log(a) ... ]
```

Computes the natural logarithm of **a** .

log10()

logarithm with basis 10

```
[ ... log10(a) ... ]
```

Computes the base 10 logarithm of **a** .

max()

maximum of two or more arguments

```
[ ... max(arg1, arg2, ..., argn) ... ]
```

Computes the maximum of **arg1** , **arg2** , ..., **argn** .

min()

minimum of two or more arguments

```
[ ... min(arg1, arg2, ..., argn) ... ]
```

Computes the minimum of **arg1** , **arg2** , ..., **argn** .

mod()

modulo operation

examples: mod(A,P)=A-FLOOR(A/P)*P , i.e. 1=mod(7,6), 6=mod(6,7), 0=mod(6,2)

nbsum()

sum over points in neighbor list

Sum up the values of a single index over all points in the neighbor list which have the same chamber, including the center point.

Syntax example:

```
[ ... nbsum( %ind_Vi% ) ... ]
```

Note:
By default, sums in DROPLETPHASE chambers are based on the full neighbor list of geometrical neighbor points, not the reduced one which is used in the stencil calculation and which is limited by max_N_stencil . On the other hand, in any other chamber, the "filtered" lists are used. To overwrite these defaults, one can specify the following constants as a

second argument:

- %EQN_nbsum_filtered%
  %EQN_nbsum_nonfiltered%

Syntax example:

```
[ ... nbsum( %ind_Vi% , %EQN_nbsum_filtered% ) ... ]
```


nrand()

random sample from a normal distribution

Produces a random sample from a normal distribution with mean mue and standard deviation sigma. Syntax:

```
[ ... nrand(mue, sigma) ... ]
```

Remark: Both arguments must be provided.

ode()

incorporate results of ODE solvers

Definition of ODE :

```
ODE (n) = ( A, B, Q, Finit )
```

The result of the ODE solver can be used inside of an equation by:

```
begin_equation{ $EqunName$ }
... ode(n) ...
end_equation
```

n is the index of the ODE .

omCOG()

rotational speed of the center of gravity for a given MOVE-flag

```
[ ... omCOG(i, $MOVEFlag$ ) ... ]
```

i = 1,2,3 yields the x-, y-, z-component of the rotational speed ω of the center of gravity for the given **$MOVEFlag$** , respectively.

$MOVEFlag$ is directly associated to all boundary elements carrying this MOVE -flag.

pmin()

minimum of all strictly positive values

The algorithm selects all strictly positive numbers and forms their minimum.

Example:

```
begin_equation{ $EqunName$ }
pmin( -0.0001, 10, -5, 0.1, -80, 6, ... )
end_equation
```

The result is 0.1.

projY()

projection of a MESHFREE-entity by smooth, Shepard-type approximation

The projection of a MESHFREE -entity is done by a smooth, least-squares approximation of Shepard-type. Depending on the
given parameters, the projection is done from a different chamber, only for specific types of points,
or with a specific kernel. The MESHFREE -used least squares approximation naturaly fall back to the Shepard
apprximation if order 1 is chosen. The explicit formulation is

$$\tilde{u}(\mathbf{x}) = \frac{\sum_{j}^{N(\mathbf{x})} W(\mathbf{x}_j, \mathbf{x}) \cdot u(\mathbf{x}_j)}{\sum_{j}^{N(\mathbf{x})} W(\mathbf{x}_j, \mathbf{x})}$$

with $W(\mathbf{x}_j, \mathbf{x}) = \exp\left(-\alpha \frac{\|\mathbf{x}_j - \mathbf{x}\|^2}{h(\mathbf{x}_j)^2}\right)$ where $\mathbf{x}_j$ are the neighbors and $h(\mathbf{x}_j)$ the smoothing length

The basic projection of the MESHFREE -entity **%ind_Entity%** (see Indices) is invoked as:

```
[ ... projY(%ind_Entity%) ... ]
```

The values of an entity from a different chamber with chamber index **iChamber** can be projected by:

```
[ ... projY(iChamber, %ind_Entity%, OPTIONAL:WhatPointsShouldBeUsed , alphaKernel ) ... ]
```

WhatPointsShouldBeUsed :

- %EQN_Proj_INT% (force the projection using only interior points)
- %EQN_Proj_BND% (force the projection using only boundary points)
- %EQN_Proj_ALL% (force the projection using all types of points, i.e. interior and boundary points)

The default is %EQN_Proj_ALL% .

alphaKernel : This option control the weight function by setting the parameter α (see above)

General Remark : given a projection task in iChamber at the location $\boldsymbol{x}$, then MESHFREE will search for
the closest neighbor point at location $\boldsymbol{x}_i$ in iChamber. The neighbors for the projection task around $\boldsymbol{x}$
will be executed using the neighbor list of $\boldsymbol{x}_i$. Thus, the choice of the parameter NEIGHBOR_FilterMethod
will have a big impact on the results of the projection. We remember that, using NEIGHBOR_FilterMethod > 1, we prevent the
neighbor search from "looking through" thin walls.

rand()

random number generator

```
[ ... rand(a, OPTIONAL: iReproducible ) ... ]
```

Options for **a** :

- a = 1 (this will produce a random number between 0 and 1)
- a is a positive real number (this will produce a random number between 0 and a)
- a is a negative real number (this will produce a random number between a and -a)

Optional parameter **iReproducible** :
iReproducible has to be an integer number > 0. This allows to generate reproducible random numbers, so rand(a,15508) will always represent the
same random number, no matter where and when applied (provided that a is the same in all cases).
This function can be helpful to generate random droplet sources in planes or similar tasks.

The following example shows how to setup random droplets create along a plane inclined in y-z-direction

```
####################################################
# set up the droplet source
####################################################
begin_equation{ "iDroplet"} # this equation returns the unique counter of the next droplet
real( %DropletSource_provideCounter% , 1) + 1 # add one as the counter actuially provides the droplet index of the provious droplet created
end_equation
DropletSource (1) = ( 0.05, [(1.7* &Hmin& )^3], [rand(-1)*2], [0.7357+rand(1,equn( $iDroplet$ ))*0.85028], [3.9705-rand(1,equn( $iDroplet$ ))*2.55923], 1, $Mat1$ ) # was y=0.9
```


real()

incorporate standard MESHFREE-postprocessing and statistics

Real function in MESHFREE -equations with either one or two arguments:

```
[ ... real(%MESHFREE_Variable%) ... ]
[ ... real(%MESHFREE_Variable%, Argument) ... ]
```

For details see below.

List of members:

OneArgument	real function in MESHFREE-equations with ONE parameter/argument
TwoArguments	real function in MESHFREE-equations with TWO parameters/arguments

OneArgument

real function in MESHFREE-equations with ONE parameter/argument

```
begin_equation{ $Name$}
real(%MF_Variable%)
end_equation
```

The options for %MF_Variable% are listed below.

List of members:

%BND_count_BE%	current number of boundary elements belonging to the geometry
%BND_count_NP%	current number of node points belonging to the geometry
%CLOCK_STATISTICS_TOTAL_FLIQUID%	CLOCK time summed over all MESHFREE points and the entire simulation time of the (pure) MESHFREE numerics
%CLOCK_STATISTICS_TOTAL_ORGANIZE%	CLOCK time summed over all MESHFREE points and the entire simulation time of the MESHFREE organization
%CLOCK_STATISTICS_TOTAL_SAMG%	CLOCK time for SAMG (BETA! USE WITH CAUTION!)
%CPU_STATISTICS_TOTAL_FLIQUID%	CPU time summed over all MESHFREE points and the entire simulation time of the (pure) MESHFREE numerics
%CPU_STATISTICS_TOTAL_ORGANIZE%	CPU time summed over all MESHFREE points and the entire simulation time of the MESHFREE organization
%ElapsedTimeIntegrationCycle%	elapsed CPU time for (pure) MESHFREE numerics
%ElapsedTimePointOrganization%	elapsed CPU time for MESHFREE organization
%FLIQUID_NbParticles%	current number of ACTIVE MESHFREE points
%MEM_STATISTICS_ALLOC%	currently allocated memory of the node with the highest workload
%MEM_STATISTICS_AVAIL%	currently available memory per node
%MEMORIZEDelete_NbParticles%	current number of MESHFREE points that are deleted due to MEMORIZE_Write statements
%MEMORIZEKeep_NbParticles%	current number of MESHFREE points that are kept due to MEMORIZE_Write statements
%MONITOR_NbParticles%	current number of MESHFREE monitor points
%MPI_NbProcesses%	current number of MPI processes
%NumberTimeStepsExecuted%	current number of time steps executed in general
%OMP_NbProcesses%	current number of openMP threads
%ORGANIZE_NbParticles%	current number of ALL MESHFREE points (inactive + active)
%RealTimeSimulation%	real simulation time
%SAVE_FreeUnit%	minimum number of available file units
%SAVE_FreeUnit100%	minimum number of available file units between 111 and 1000
%TIME_InitTime%	startup and initialization time in seconds

%TIME_StartTime%	timestamp at startup of MESHFREE
%TIME_StepStartTime%	timestamp at start of current time step
%TIME_StepWallTime%	walltime of current time step in seconds
%TIME_WallTime%	walltime in seconds
%VMEM_STATISTICS_ALLOC%	currently allocated virtual memory
%VMEM_STATISTICS_AVAIL%	currently available virtual memory

%BND_count_BE%

current number of boundary elements belonging to the geometry

Example:

```
begin_equation{ $boundary_elements$ }
real( %BND_count_BE% )
end_equation
```


%BND_count_NP%

current number of node points belonging to the geometry

Example:

```
begin_equation{ $node_points$ }
real( %BND_count_NP% )
end_equation
```


%CLOCK_STATISTICS_TOTAL_FLIQUID%

CLOCK time summed over all MESHFREE points and the entire simulation time of the (pure) MESHFREE numerics

Example:

```
begin_equation{ $clock_total_fliquid$ }
real( %CLOCK_STATISTICS_TOTAL_FLIQUID% )
end_equation
```


%CLOCK_STATISTICS_TOTAL_ORGANIZE%

CLOCK time summed over all MESHFREE points and the entire simulation time of the MESHFREE organization

Example:

```
begin_equation{ $clock_total_organize$ }
real( %CLOCK_STATISTICS_TOTAL_ORGANIZE% )
end_equation
```

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · OneArgument · %CLOCK_STATISTICS_TOTAL_SAMG%

%CLOCK_STATISTICS_TOTAL_SAMG%

CLOCK time for SAMG (BETA! USE WITH CAUTION!)

Example:

```
begin_equation{ $clock_total_samg$ }
real( %CLOCK_STATISTICS_TOTAL_SAMG% )
end_equation
```

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · OneArgument · %CPU_STATISTICS_TOTAL_FLIQUID%

%CPU_STATISTICS_TOTAL_FLIQUID%

CPU time summed over all MESHFREE points and the entire simulation time of the (pure) MESHFREE numerics

Example:

```
begin_equation{ $cpu_total_fliquid$ }
real( %CPU_STATISTICS_TOTAL_FLIQUID% )
end_equation
```

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · OneArgument · %CPU_STATISTICS_TOTAL_ORGANIZE%

%CPU_STATISTICS_TOTAL_ORGANIZE%

CPU time summed over all MESHFREE points and the entire simulation time of the MESHFREE organization

Example:

```
begin_equation{ $cpu_total_organize$ }
real( %CPU_STATISTICS_TOTAL_ORGANIZE% )
end_equation
```

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · OneArgument · %ElapsedTimeIntegrationCycle%

%ElapsedTimeIntegrationCycle%

elapsed CPU time for (pure) MESHFREE numerics

Example:

```
begin_equation{ $time_numerics$ }
real( %ElapsedTimeIntegrationCycle% )
end_equation
```


%ElapsedTimePointOrganization%

elapsed CPU time for MESHFREE organization

Example:

```
begin_equation{ $time_organize$ }
real( %ElapsedTimePointOrganization% )
end_equation
```


%FLIQUID_NbParticles%

current number of ACTIVE MESHFREE points

Example:

```
begin_equation{ $points_FLIQUID$ }
real( %FLIQUID_NbParticles% )
end_equation
```


%MEMORIZEDelete_NbParticles%

current number of MESHFREE points that are deleted due to MEMORIZE_Write statements

Example:

```
INTEGRATION ( $Int_MEMORIZEDelete$ ) = ( %PUBLICVALUE% , real( %MEMORIZEDelete_NbParticles% ) )
```


%MEMORIZEKeep_NbParticles%

current number of MESHFREE points that are kept due to MEMORIZE_Write statements

Example:

```
INTEGRATION ( $Int_MEMORIZEKeep$ ) = ( %PUBLICVALUE% , real( %MEMORIZEKeep_NbParticles% ) )
```

%MEM_STATISTICS_ALLOC%

currently allocated memory of the node with the highest workload

Example:

```
begin_equation{ $alloc_mem$ }
real( %MEM_STATISTICS_ALLOC% )
end_equation
```

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · OneArgument · %MEM_STATISTICS_AVAIL%

%MEM_STATISTICS_AVAIL%

currently available memory per node

Example:

```
begin_equation{ $avail_mem$ }
real( %MEM_STATISTICS_AVAIL% )
end_equation
```

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · OneArgument · %MONITOR_NbParticles%

%MONITOR_NbParticles%

current number of MESHFREE monitor points

Example:

```
begin_equation{ $points_monitor$ }
real( %MONITOR_NbParticles% )
end_equation
```

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · OneArgument · %MPI_NbProcesses%

%MPI_NbProcesses%

current number of MPI processes

Example:

```
begin_equation{ $mpi_procs$ }
real( %MPI_NbProcesses% )
end_equation
```

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · OneArgument · %NumberTimeStepsExecuted%

%NumberTimeStepsExecuted%

current number of time steps executed in general

Example:

```
begin_equation{ $nb_time_steps$ }
real( %NumberTimeStepsExecuted% )
end_equation
```


%OMP_NbProcesses%

current number of openMP threads

Example:

```
begin_equation{ $omp_threads$ }
real( %OMP_NbProcesses% )
end_equation
```


%ORGANIZE_NbParticles%

current number of ALL MESHFREE points (inactive + active)

Example:

```
begin_equation{ $points_ORGANIZE$ }
real( %ORGANIZE_NbParticles% )
end_equation
```


%RealTimeSimulation%

real simulation time

Example:

```
begin_equation{ $simulation_time$ }
real( %RealTimeSimulation% )
end_equation
```


%SAVE_FreeUnit%

minimum number of available file units

Example:

```
begin_equation{ $free_units$ }
real( %SAVE_FreeUnit% )
end_equation
```

%SAVE_FreeUnit100%

minimum number of available file units between 111 and 1000

Example:

```
begin_equation{ $free_units100$ }
real( %SAVE_FreeUnit100% )
end_equation
```


%TIME_InitTime%

startup and initialization time in seconds

Time in seconds from TIME_StartTime until right before ADMIN_TIME_INTEG. a# #b

Example:

```
begin_equation{ $init_time$ }
real( %TIME_InitTime% )
end_equation
```


%TIME_StartTime%

timestamp at startup of MESHFREE

Time in seconds from 1. January 1970 12:00 am (midnight GMT). a# #b

Example:

```
begin_equation{ $start_time$ }
real( %TIME_StartTime% )
end_equation
```


%TIME_StepStartTime%

timestamp at start of current time step

Time in seconds from 1. January 1970 12:00 am (midnight GMT). a# #b

Example:

```
begin_equation{ $step_start_time$ }
real( %TIME_StepStartTime% )
end_equation
```

%TIME_StepWallTime%

walltime of current time step in seconds

Time the software is running from TIME_StepStartTime given in seconds. a# #b
Example:

```
begin_equation{ $step_wall_time$ }
real( %TIME_StepWallTime% )
end_equation
```


%TIME_WallTime%

walltime in seconds

Time the software is running from TIME_StartTime given in seconds. a# #b
Example:

```
begin_equation{ $wall_time$ }
real( %TIME_WallTime% )
end_equation
```


%VMEM_STATISTICS_ALLOC%

currently allocated virtual memory

Example:

```
begin_equation{ $alloc_vmem$ }
real( %VMEM_STATISTICS_ALLOC% )
end_equation
```


%VMEM_STATISTICS_AVAIL%

currently available virtual memory

Example:

```
begin_equation{ $avail_vmem$ }
real( %VMEM_STATISTICS_AVAIL% )
end_equation
```

TwoArguments

real function in MESHFREE-equations with TWO parameters/arguments

```
begin_equation{ $Name$}
real( %MF_Variable%, Argument )
end_equation
```

The options for %MF_Variable% are listed below including details on the parameter Argument.

List of members:

%PUBLICVALUE_xValueOfBNDpoint%	x-coordinate of a BND_point carrying a certain POSTROCESS-flag
%PUBLICVALUE_yValueOfBNDpoint%	y-coordinate of a BND_point carrying a certain POSTROCESS-flag
%PUBLICVALUE_zValueOfBNDpoint%	z-coordinate of a BND_point which carries a certain POSTROCESS-flag
%CPU_STATISTICS_FLIQUID%	CPU time measured for the execution of the (pure) MESHFREE numerics at the current time step
%CLOCK_STATISTICS_FLIQUID%	CLOCK time measured for the execution of the (pure) MESHFREE numerics at the current time step
%CPU_STATISTICS_ORGANIZE%	CPU time measured for the execution of the MESHFREE organization (point cloud management, geometry operations) at the current time step
%CLOCK_STATISTICS_ORGANIZE%	CLOCK time measured for the execution of the MESHFREE organization (point cloud management, geometry operations) at the current time step
%FPM_VOLUME_TARGET%	target value of volume in a given chamber
%FPM_VOLUME_ACTUAL%	actual value of volume in a given chamber
%FPM_VOLUME_DeletedAtMetaplanes%	volume reduced by deletion of MESHFREE points at metaplanes AND by EVENT statements in the current time step
%FPM_RepMass_CreatedByInflowOutflow%	representative mass created by flow through %BND_inflow% and %BND_outflow% boundaries
%FPM_RepMass_DeletedAtMetaplanes%	representative mass reduced by deletion of MESHFREE points at metaplanes and EVENT-cuts
%FPM_RepMass_CreatedByDropletSource%	representative mass created by the droplet sources in a chamber or material
%FPM_KineticEnergy_DifferenceInOrganize%	change of kinetic energy in some chamber during MESHFREE organization
%FPM_KineticEnergy_DifferenceInOrganize2%	change of kinetic energy in some chamber during MESHFREE organization at the end of the time step
%FPM_KineticEnergy_DifferenceInTimeStep%	change of kinetic energy in some chamber during (pure) MESHFREE numerics

%FPM_KineticEnergy%	total kinetic energy of a given chamber
%FPM_KineticEnergy_Defect_gradPv%	first order defect of kinetic energy during time integration due to pressure
%FPM_KineticEnergy_Defect_rhogDv%	first order defect of kinetic energy during time integration due to gravity
%FPM_KineticEnergy_Defect_O2%	second order defect of kinetic energy during time integration
%DropletSource_provideCounter%	current status of the droplet counter of a given/defined DropletSource
%DropletSource_provideTargetVolume%	current status of the target volume of a given/defined DropletSource
%DropletSource_provideCurrentVolume%	current status of the actually injected volume by a given/defined DropletSource
%SurfaceTriangulation_NbStencil%	number of triangles/tetras established by free surface Delaunay triangulation
%BUBBLE_EQN_TruePressure%	true bubble pressure for given bubble index

%BUBBLE_EQN_TruePressure%

true bubble pressure for given bubble index

If the BubbleAlgorithm is switched on, the bubbles are tracked, and different ways of bubble pressure computation are used. The true pressure concept ist the original one and explained in BubbleTruePressure . Interrogate the true pressure of a given bubble index by

```
[ ... real( %BUBBLE_EQN_TruePressure% , iArgument) ... ]
```

iArgument is the index of the bubble under consideration, see %ind_bndBubble% .

Example:

```
[ ... real( %BUBBLE_EQN_TruePressure% , Y %ind_bndBubble% ) ... ]
```


%CLOCK_STATISTICS_FLIQUID%

CLOCK time measured for the execution of the (pure) MESHFREE numerics at the current time step

The CLOCK time for the execution of the pure MESHFREE numerics (no MESHFREE organization) is measured at the current time step.

```
[ ... real( %CLOCK_STATISTICS_FLIQUID% , Argument) ... ]
```

Argument:

- 1 (per MESHFREE point average CLOCK time over all MPI processes)
- 2 (number of MPI processes times MINIMUM CLOCK time elapsed at some MPI process, divided by the global

number of MESHFREE points)
- 3 (number of MPI processes times MAXIMUM CLOCK time elapsed at some MPI process, divided by the global number of MESHFREE points)
- 4 (summation of CLOCK time over all MPI processes)
- 5 (number of MPI processes times MINIMUM CLOCK time elapsed at some MPI process)
- 6 (number of MPI processes times MAXIMUM CLOCK time elapsed at some MPI process)

%CLOCK_STATISTICS_ORGANIZE%

CLOCK time measured for the execution of the MESHFREE organization (point cloud management, geometry operations) at the current time step

The CLOCK time for the execution of the MESHFREE organization (no pure MESHFREE numerics) is measured at the current time step.

```
[ ... real( %CLOCK_STATISTICS_ORGANIZE% , Argument) ... ]
```

Argument:
- 1 (per MESHFREE point average CLOCK time over all MPI processes)
- 2 (number of MPI processes times MINIMUM CLOCK time elapsed at some MPI process, divided by the global number of MESHFREE points)
- 3 (number of MPI processes times MAXIMUM CLOCK time elapsed at some MPI process, divided by the global number of MESHFREE points)
- 4 (summation of CLOCK time over all MPI processes)
- 5 (number of MPI processes times MINIMUM CLOCK time elapsed at some MPI process)
- 6 (number of MPI processes times MAXIMUM CLOCK time elapsed at some MPI process)

%CPU_STATISTICS_FLIQUID%

CPU time measured for the execution of the (pure) MESHFREE numerics at the current time step

The CPU time for the execution of the pure MESHFREE numerics (no MESHFREE organization) is measured at the current time step.

```
[ ... real( %CPU_STATISTICS_FLIQUID% , Argument) ... ]
```

Argument:
- 1 (per MESHFREE point average CPU time over all MPI processes)
- 2 (number of MPI processes times MINIMUM CPU time elapsed at some MPI process, divided by the global number of MESHFREE points)
- 3 (number of MPI processes times MAXIMUM CPU time elapsed at some MPI process, divided by the global number of MESHFREE points)
- 4 (summation of CPU time over all MPI processes)
- 5 (number of MPI processes times MINIMUM CPU time elapsed at some MPI process)
- 6 (number of MPI processes times MAXIMUM CPU time elapsed at some MPI process)

%CPU_STATISTICS_ORGANIZE%

CPU time measured for the execution of the MESHFREE organization (point cloud management, geometry operations) at the current time step

The CPU time for the execution of the MESHFREE organization (no pure MESHFREE numerics) is measured at the current time step.

[... real(%CPU_STATISTICS_ORGANIZE% , Argument) ...]

Argument:

- 1 (per MESHFREE point average CPU time over all MPI processes)
- 2 (number of MPI processes times MINIMUM CPU time elapsed at some MPI process, divided by the global number of MESHFREE points)
- 3 (number of MPI processes times MAXIMUM CPU time elapsed at some MPI process, divided by the global number of MESHFREE points)
- 4 (summation of CPU time over all MPI processes)
- 5 (number of MPI processes times MINIMUM CPU time elapsed at some MPI process)
- 6 (number of MPI processes times MAXIMUM CPU time elapsed at some MPI process)

%DropletSource_provideCounter%

current status of the droplet counter of a given/defined DropletSource

[... real(%DropletSource_provideCounter% , Argument) ...]

Argument is the index of the DropletSource .

%DropletSource_provideCurrentVolume%

current status of the actually injected volume by a given/defined DropletSource

[... real(%DropletSource_provideCurrentVolume% , Argument) ...]

Argument is the index of the DropletSource .

%DropletSource_provideTargetVolume%

current status of the target volume of a given/defined DropletSource

[... real(%DropletSource_provideTargetVolume% , Argument) ...]

Argument is the index of the DropletSource .

%FPM_KineticEnergy%

total kinetic energy of a given chamber

[... real(%FPM_KineticEnergy% , Argument) ...]

Argument is the chamber index as given in KOP() or as specified by the CHAMBER -flag in AliasForGeometryItems .

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · TwoArguments · %FPM_KineticEnergy_Defect_O2%

%FPM_KineticEnergy_Defect_O2%

second order defect of kinetic energy during time integration

[... real(%FPM_KineticEnergy_Defect_O2% , Argument) ...]

Argument is the chamber index as given in KOP() or as specified by the CHAMBER -flag in AliasForGeometryItems .

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · TwoArguments · %FPM_KineticEnergy_Defect_gradPv%

%FPM_KineticEnergy_Defect_gradPv%

first order defect of kinetic energy during time integration due to pressure

[... real(%FPM_KineticEnergy_Defect_gradPv% , Argument) ...]

Argument is the chamber index as given in KOP() or as specified by the CHAMBER -flag in AliasForGeometryItems .

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · TwoArguments · %FPM_KineticEnergy_Defect_rhogDv%

%FPM_KineticEnergy_Defect_rhogDv%

first order defect of kinetic energy during time integration due to gravity

[... real(%FPM_KineticEnergy_Defect_rhogDv% , Argument) ...]

Argument is the chamber index as given in KOP() or as specified by the CHAMBER -flag in AliasForGeometryItems .

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · TwoArguments · %FPM_KineticEnergy_DifferenceInOrganize%

%FPM_KineticEnergy_DifferenceInOrganize%

change of kinetic energy in some chamber during MESHFREE organization

[... real(%FPM_KineticEnergy_DifferenceInOrganize% , Argument) ...]

Argument is the chamber index as given in KOP() or as specified by the CHAMBER -flag in AliasForGeometryItems .

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · TwoArguments · %FPM_KineticEnergy_DifferenceInOrganize2%

%FPM_KineticEnergy_DifferenceInOrganize2%

change of kinetic energy in some chamber during MESHFREE organization at the end of the time step

This value should be strictly zero.

[... real(%FPM_KineticEnergy_DifferenceInOrganize2% , Argument) ...]

Argument is the chamber index as given in KOP() or as specified by the CHAMBER -flag in AliasForGeometryItems .

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · TwoArguments · %FPM_KineticEnergy_DifferenceInTimeStep%

%FPM_KineticEnergy_DifferenceInTimeStep%

change of kinetic energy in some chamber during (pure) MESHFREE numerics

[... real(%FPM_KineticEnergy_DifferenceInTimeStep% , Argument) ...]

Argument is the chamber index as given in KOP() or as specified by the CHAMBER -flag in AliasForGeometryItems .

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · TwoArguments · %FPM_RepMass_CreatedByDropletSource%

%FPM_RepMass_CreatedByDropletSource%

representative mass created by the droplet sources in a chamber or material

[... real(%FPM_RepMass_CreatedByDropletSource% , Argument) ...]

Argument is the chamber index as given in KOP() or as specified by the CHAMBER -flag in AliasForGeometryItems .

WARNING: this functionality will deliver reasonable values only if used for INTEGRATION statements with the %PUBLICVALUE% and %PUBLICVALUE_SUM% directives.
If used for boundary conditions, physical properties, etc., it will deliver 0 .

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · TwoArguments · %FPM_RepMass_CreatedByInflowOutflow%

%FPM_RepMass_CreatedByInflowOutflow%

representative mass created by flow through %BND_inflow% and %BND_outflow% boundaries

[... real(%FPM_RepMass_CreatedByInflowOutflow% , Argument) ...]

Argument is the chamber index as given in KOP() or as specified by the CHAMBER -flag in AliasForGeometryItems .

WARNING: this functionality will deliver reasonable values only if used for INTEGRATION statements with the %PUBLICVALUE% and %PUBLICVALUE_SUM% directives.
If used for boundary conditions, physical properties, etc., it will deliver 0 .

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · TwoArguments · %FPM_RepMass_DeletedAtMetaplanes%

%FPM_RepMass_DeletedAtMetaplanes%

representative mass reduced by deletion of MESHFREE points at metaplanes and EVENT-cuts

[... real(%FPM_RepMass_DeletedAtMetaplanes% , Argument) ...]

Argument is the chamber index as given in KOP() or as specified by the CHAMBER -flag in AliasForGeometryItems .

WARNING: this functionality will deliver reasonable values only if used for INTEGRATION statements with the %PUBLICVALUE% and %PUBLICVALUE_SUM% directives.
If used for boundary conditions, physical properties, etc., it will deliver 0 .

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · TwoArguments · %FPM_VOLUME_ACTUAL%

%FPM_VOLUME_ACTUAL%

actual value of volume in a given chamber

```
[ ... real( %FPM_VOLUME_ACTUAL% , Argument) ... ]
```

Argument is the chamber index as given in KOP() or as specified by the CHAMBER -flag in AliasForGeometryItems .

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · TwoArguments · %FPM_VOLUME_DeletedAtMetaplanes%

%FPM_VOLUME_DeletedAtMetaplanes%

volume reduced by deletion of MESHFREE points at metaplanes AND by EVENT statements in the current time step

volume reduced by deletion of MESHFREE points at metaplanes and deletion triggered by EVENT -statements (%EVENT_DeletePoint%) .

```
[ ... real( %FPM_VOLUME_DeletedAtMetaplanes% , Argument) ... ]
```

Argument is the chamber index as given in KOP() or as specified by the CHAMBER -flag in AliasForGeometryItems .

WARNING: this functionality will deliver reasonable values only if used for INTEGRATION statements with the %PUBLICVALUE% argument.
If used for boundary conditions, physical properties, etc., it will deliver 0 .

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · TwoArguments · %FPM_VOLUME_TARGET%

%FPM_VOLUME_TARGET%

target value of volume in a given chamber

```
[ ... real( %FPM_VOLUME_TARGET% , Argument) ... ]
```

Argument is the chamber index as given in KOP() or as specified by the CHAMBER -flag in AliasForGeometryItems .

MESHFREE · InputFiles · USER_common_variables · Equations · Functions · real() · TwoArguments · %PUBLICVALUE_xValueOfBNDpoint%

%PUBLICVALUE_xValueOfBNDpoint%

x-coordinate of a BND_point carrying a certain POSTROCESS-flag

```
[ ... real( %PUBLICVALUE_xValueOfBNDpoint% , $POSTPROCESS_flag$ ) ... ]
```

$POSTPROCESS_flag$ has to be given at the definition level of the desired BND_point .

%PUBLICVALUE_yValueOfBNDpoint%

y-coordinate of a BND_point carrying a certain POSTROCESS-flag

```
[ ... real( %PUBLICVALUE_yValueOfBNDpoint% , $POSTPROCESS_flag$ ) ... ]
```

$POSTPROCESS_flag$ has to be given at the definition level of the desired BND_point .

%PUBLICVALUE_zValueOfBNDpoint%

z-coordinate of a BND_point which carries a certain POSTROCESS-flag

```
[ ... real( %PUBLICVALUE_zValueOfBNDpoint% , $POSTPROCESS_flag$ ) ... ]
```

$POSTPROCESS_flag$ has to be given at the definition level of the desired BND_point .

%SurfaceTriangulation_NbStencil%

number of triangles/tetras established by free surface Delaunay triangulation

```
[ ... real( %SurfaceTriangulation_NbStencil% , Argument) ... ]
```

Argument is the index of the MESHFREE point.

Example:

```
[ ... real( %SurfaceTriangulation_NbStencil% , Y %ind_IN% ) ... ]
```


reduct()

incorporate results of PointCloudReduction operation

The result of a PointCloudReduction -definition can be used inside of an equation by:

```
begin_equation{ $EqunName$ }
... reduct( iPointCloudReduction, OPTIONAL:%EQN_Reduct_Accumulated% , OPTIONAL:%EQN_Reduct_iCluster% )
...
end_equation
```

iPointCloudReduction is the index of the desired PointCloudReduction -statement.

%EQN_Reduct_Accumulated% shows how much of the reduction quantity is represented by the marked point.

%EQN_Reduct_iCluster% is the cluster index which naturally turns out during the PointCloudReduction -procedure.

Example:

```
PointCloudReduction (1) = ( [1], [10] ) # mark every 10-th MESHFREE point
SAVE_ITEM = ( %SAVE_scalar% , [reduct(1,%EQN_Reduct_Accumulated%)], "nbPointsRepresented" ) # how many points are represented by the marked point
SAVE_ITEM = ( %SAVE_scalar% , [reduct(1,%EQN_Reduct_iCluster%)], "numberingClusteringIndex" ) # display the cluster index (index of fish scale) produced by the PointCloudReduction
PointCloudReduction (2) = ( [Y %ind_Vi% ], [ &Hmax& ^3] ) # mark MESHFREE points which represent a volume that is approximately equal to &Hmax& ^3
SAVE_ITEM = ( %SAVE_scalar% , [reduct(2,%EQN_Reduct_Accumulated%)], "volumeRepresented" ) # how many points are represented by the selected point
SAVE_ITEM = ( %SAVE_scalar% , [reduct(2,%EQN_Reduct_iCluster%)], "volumeClusteringIndex" ) # display the cluster index (index of fish scale) produced by the PointCloudReduction
PointCloudReduction (3) = ( [reduct(1,%EQN_Reduct_Accumulated%)>0], [10] ) # mark every 10-th MESHFREE point out of the PointCloudReduction (1), i.e. every 100-th point
SAVE_ITEM = ( %SAVE_scalar% , [reduct(3,%EQN_Reduct_Accumulated%)], "volumeRepresented" ) # how many points are represented by the marked point
SAVE_ITEM = ( %SAVE_scalar% , [reduct(3,%EQN_Reduct_iCluster%)], "volumeClusteringIndex" ) # display the cluster index (index of fish scale) produced by the PointCloudReduction
```


rot()

rotated vector

```
[ ... rot(i, x,y,z, p_rot_x,p_rot_y,p_rot_z, alpha_rot_x,alpha_rot_y,alpha_rot_z) ... ]
```

The point (**x** , **y** , **z**) is rotated according to a defined reference point
(**p_rot_x** , **p_rot_y** , **p_rot_z**) and rotation angle (**alpha_rot_x** , **alpha_rot_y** , **alpha_rot_z**).

i = 1,2,3 yield the x-, y-, z-component of the rotated point, respectively.

Details:
The vector α defines a rotation with angle $\theta = \|\alpha\|$ (in radians) around the unit vector $\mathbf{e} = \alpha/\|\alpha\|$.
It is calculated according to Rodrigues rotation formular, i.e.

$$\mathbf{v}_{rot} = [\mathbf{v} - (\mathbf{e}\cdot\mathbf{v})\mathbf{e}]\cos(\theta) + [\mathbf{e}\times\mathbf{v}]\sin(\theta) + (\mathbf{e}\cdot\mathbf{v})\mathbf{e}$$

with $\mathbf{v}$ denoting the vector from the reference point $\mathbf{p}$ to the point $\mathbf{x}$ which shall be rotated.
Clearly, the rotated point is then given by

$$\mathbf{x}_{rot} = \mathbf{p} + \mathbf{v}_{rot}$$

sin()

sine

```
[ ... sin(a) ... ]
```

Computes the sine of **a** given in radians.

sinh()

hyperbolic sine

```
[ ... sinh(a) ... ]
```

Computes the hyperbolic sine of **a** .

sodst()

provide solution to sods shock tube problem

The Sod shock tube problem is a 1-D benchmark for GASDYN solvers. The function

```
[ ... sodst(ID,x) ... ]
```

gives the the analytical solution for density (ID=1), pressure (ID=2) and velocity (ID=3) at position x. We assume the initial shock is at position 0 and the time of function evaluation is the current time Y %ind_time%
Example : Providing the analytical solution for Sod shock tube to user defined variables via CODI . The tube is oriented parallel to the x-axis:

```
CODI_eq ( $GAS$ ,%indU_rANA%) = [ sodst(1,Y %ind_x(1)% ) ] # density
CODI_eq ( $GAS$ ,%indU_pANA%) = [ sodst(2,Y %ind_x(1)% ) ] # pressure
CODI_eq ( $GAS$ ,%indU_uANA%) = [ sodst(3,Y %ind_x(1)% ) ] # velocity
```


sqrt()

square root

```
[ ... sqrt(a) ... ]
```

Computes the square root of **a** .

step()

(unit) step function

```
[ ... step(a) ... ]
```

- 1 if $\mathrm{abs}(a) \leq 1$
- 0 if $\mathrm{abs}(a) > 1$

tan()

tangent

```
[ ... tan(a) ... ]
```

Computes the tangent of **a** given in radians.

tanh()

hyperbolic tangent

```
[ ... tanh(a) ... ]
```

Computes the hyperbolic tangent of **a** .

vCOG()

velocity of the center of gravity for a given MOVE-flag

```
[ ... xCOG(i, $MOVEFlag$ ) ... ]
```

i = 1,2,3 yields the x-, y-, z-component of the velocity of the center of gravity for the given **$MOVEFlag$** , respectively.

$MOVEFlag$ is directly associated to all boundary elements carrying this MOVE -flag.

xCOG()

position of the center of gravity for a given MOVE-flag

```
[ ... xCOG(i, $MOVEFlag$ ) ... ]
```

i = 1,2,3 yields the x-, y-, z-component of the center of gravity for the given **$MOVEFlag$** , respectively.

$MOVEFlag$ is directly associated to all boundary elements carrying this MOVE -flag.

Operators

standard math operators

< : less than

```
a < b
```

Result is 1 if **a** is less than **b** and 0 otherwise.

> : greater than

```
a > b
```

Result is 1 if **a** is greater than **b** and 0 otherwise.

= : equal to

```
a = b
```

Result is 1 if **a** and **b** are equal and 0 otherwise.

! : not equal to

```
a ! b
```

Result is 1 if **a** and **b** are not equal and 0 otherwise.
+ : summation, addition
```
a + b
```

Adds the values of **a** and **b** . The result is the sum of **a** and **b** .
- : subtraction, difference
```
a - b
```

Subtracts **b** from **a** . The result is the difference of **a** and **b** .
* : multiplication, product
```
a * b
```

Multiplies **a** and **b** . The result is the product of **a** and **b** .
/ : division, quotient
```
a / b
```

Divides **a** by **b** . The result is the quotient of **a** and **b** .
^ or ** : power
```
a^b
a**b
```

Takes **a** to the power of **b** .

MESHFREE · InputFiles · USER_common_variables · INITDATA

3.1.14. INITDATA

prescribe initial data conditions

To define a transient simulation model properly, initial conditions must be provided for the quantities of interest. In MESHFREE , the initial condition is prescribed per material for each quantity by the following syntax:

```
INITDATA ( $MatTag$ ,%ind_quantity%)= RightHandSideExpression
```

where **$MatTag$** is the material tag, **%ind_quantity%** is the index of the quantity, and **RightHandSideExpression** is a (scalar) expression.

Note:

- If the initial value of a quantity is not defined, this value is defaulted to 0.
- There are no checks, whether the initial value is reasonable or not. The user has to make sure to provide appropriate initial values. For example, if k-epsilon turbulence modeling is turned on, k and epsilon must be initialized to positive values.
- There are also no checks regarding consistency of the initial conditions to boundary conditions, e.g. at an inflow boundary. The user should provide initial values consistent to the boundary conditions. This holds especially for the velocity. If there are inconsistencies, then you might observe instabilities in the first couple of iterations.

Example 1: Define the initial temperature of material Air to be 273.15 K in teh simulation domain

```
INITDATA ( $Air$ ,%ind_T%)= 273.15
```

Example 2: Define the initial turbulence quantities of material Air as constant positive values in the simulation domain

```
INITDATA ( $Air$ ,%ind_eps%) = 10
INITDATA ( $Air$ ,%ind_k%) = 1e-4
```

MESHFREE · InputFiles · USER_common_variables · INTEGRATION

3.1.15. INTEGRATION

integration of the simulation results

With the help of INTEGRATION statements in USER_common_variables , simulation quantities can be further analyzed. Application examples are monitoring conservation quantities such as the total mass in the simulation model or evaluating a quantity like pressure at a certain position corresponding to a sensor in experiments.

The result of an INTEGRATION statement is a scalar value. For each timestep, the INTEGRATION statement is evaluated and written to a so-called TimestepFile , that can be found in the result folder.

Optionally, each INTEGRATION statement can be supplemented with an %INTEGRATION_Header% to provide the column headers, see HeaderInfoOrComments .

Integration types

The following types of integrations are available:

- Volume and boundary integrals with respect to the point cloud
- Maximum, minimum, summation, average with respect to the point cloud
- Values and approximation for a BND_point
- Public values of the MESHFREE simulation
- Boundary integrals with respect to the boundary elements
- Maximum and minimum with respect to the boundary elements
- Assignment of function values to points (alternative to CODI and EVENT)

Volume and boundary integrals with respect to the point cloud

```
INTEGRATION ( $IntInd1$ ) = ( %INTEGRATION_INT% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ , ... )
INTEGRATION ( $IntInd2$ ) = ( %INTEGRATION_INT_TIME% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ , ... )

INTEGRATION ( $IntInd3$ ) = ( %INTEGRATION_BND_DIRECT% , ExpressionOfIntegrand , $PostprocessTag1$ , $PostprocessTag2$ , ... )
INTEGRATION ( $IntInd4$ ) = ( %INTEGRATION_BND_DIRECT_TIME% , ExpressionOfIntegrand , $PostprocessTag1$ , $PostprocessTag2$ , ... )

INTEGRATION ( $IntInd5$ ) = ( %INTEGRATION_FS_DIRECT% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ , ... )
INTEGRATION ( $IntInd6$ ) = ( %INTEGRATION_FS_DIRECT_TIME% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ , ... )

INTEGRATION ( $IntInd7$ ) = ( %INTEGRATION_BND% , ExpressionOfIntegrand , ExpressionOfIntegrand , ExpressionOfIntegrand , $PostprocessTag1$ , $PostprocessTag2$ , ... )
INTEGRATION ( $IntInd8$ ) = ( %INTEGRATION_BND_TIME% , ExpressionOfIntegrand , ExpressionOfIntegrand , ExpressionOfIntegrand , $PostprocessTag1$ , $PostprocessTag2$ , ... )

INTEGRATION ( $IntInd9$ ) = ( %INTEGRATION_FS% , ExpressionOfIntegrand , ExpressionOfIntegrand , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ , ... )
INTEGRATION ( $IntInd10$ ) = ( %INTEGRATION_FS_TIME% , ExpressionOfIntegrand , ExpressionOfIntegrand , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ , ... )

INTEGRATION ( $IntInd11$ ) = ( %INTEGRATION_FLUX% , ExpressionOfIntegrand , $PostprocessTag1$ , $PostprocessTag2$ , ... )
INTEGRATION ( $IntInd12$ ) = ( %INTEGRATION_FLUX_TIME% , ExpressionOfIntegrand , $PostprocessTag1$ , $PostprocessTag2$ , ... )

INTEGRATION ( $IntInd13$ ) = ( %INTEGRATION_ABSFLUX% , ExpressionOfIntegrand , $PostprocessTag1$ , $PostprocessTag2$ , ... )
INTEGRATION ( $IntInd14$ ) = ( %INTEGRATION_ABSFLUX_TIME% , ExpressionOfIntegrand , $PostprocessTag1$ , $PostprocessTag2$ , ... )

INTEGRATION ( $IntInd15$ ) = ( %INTEGRATION_FLUX_DROPLETPHASE% , ExpressionOfIntegrand , $PostprocessTag1$ , $PostprocessTag2$ , ... )
INTEGRATION ( $IntInd16$ ) = ( %MASSFLOW_DROPLETPHASE% , ExpressionOfIntegrand )
```

Application example: Monitoring conservation quantities such as mass or energy.

Maximum, minimum, summation, average with respect to the point cloud

```
INTEGRATION ( $IntInd17$ ) = ( %MAXIMUM_INT% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ , ... )
INTEGRATION ( $IntInd18$ ) = ( %MINIMUM_INT% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ , ... )
INTEGRATION ( $IntInd18$ ) = ( %SUMMATION_INT% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ , ...
)
INTEGRATION ( $IntInd19$ ) = ( %AVERAGE_INT% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ , ... )

INTEGRATION ( $IntInd20$ ) = ( %MAXIMUM_BND% , ExpressionOfIntegrand , $PostprocessTag1$ ,
$PostprocessTag2$ , ... )
INTEGRATION ( $IntInd21$ ) = ( %MINIMUM_BND% , ExpressionOfIntegrand , $PostprocessTag1$ ,
$PostprocessTag2$ , ... )
INTEGRATION ( $IntInd21$ ) = ( %SUMMATION_BND% , ExpressionOfIntegrand , $PostprocessTag1$ ,
$PostprocessTag2$ , ... )
INTEGRATION ( $IntInd22$ ) = ( %AVERAGE_BND% , ExpressionOfIntegrand , $PostprocessTag1$ ,
$PostprocessTag2$ , ... )

INTEGRATION ( $IntInd23$ ) = ( %MAXIMUM_FS% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ , ... )
INTEGRATION ( $IntInd24$ ) = ( %MINIMUM_FS% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ , ... )
INTEGRATION ( $IntInd25$ ) = ( %AVERAGE_FS% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ , ... )
```

Application example: Monitoring the range of quantities like pressure.
Values and approximation for a BND_point

```
INTEGRATION ( $IntInd26$ ) = ( %POINT_DIRECT% , ExpressionOfIntegrand , $PostprocessTag1$ ,
$PostprocessTag2$ , ... )
INTEGRATION ( $IntInd27$ ) = ( %POINT_APPROXIMATE% , ExpressionOfIntegrand , $PostprocessTag1$ ,
$PostprocessTag2$ , ... )
INTEGRATION ( $IntInd28$ ) = ( %POINT_APPROXIMATE_ProjBNDOnly% , ExpressionOfIntegrand ,
$PostprocessTag1$ , $PostprocessTag2$ , ... )
```

Application example: Evaluating quantities in the simulation at the position where they are also measured in experiment.
Public values of the MESHFREE simulation

```
INTEGRATION ( $IntInd29$ ) = ( %PUBLICVALUE% , Functional )
INTEGRATION ( $IntInd30$ ) = ( %PUBLICVALUE_TIME% , Functional )
INTEGRATION ( $IntInd31$ ) = ( %PUBLICVALUE_SUM% , Functional )

INTEGRATION ( $IntInd32$ ) = ( %PUBLICVALUE_CLOCKstatistics% , iArgument, "NameOfStopWatch" )
INTEGRATION ( $IntInd33$ ) = ( %PUBLICVALUE_CPUstatistics% , iArgument, "NameOfStopWatch" )
```

Application example: Monitor the total number of points or other internal quantities.
Boundary integrals with respect to the boundary elements

```
INTEGRATION ( $IntInd34$ ) = ( %BE_INTEGRATION_DIRECT% , ExpressionOfIntegrand , $PostprocessTag1$ ,
$PostprocessTag2$ , ... )
INTEGRATION ( $IntInd35$ ) = ( %BE_INTEGRATION_DIRECT_TIME% , ExpressionOfIntegrand , $PostprocessTag1$
, $PostprocessTag2$ , ... )
```

Maximum and minimum with respect to the boundary elements

```
INTEGRATION ( $IntInd36$ ) = ( %MINIMUM_BE%, ExpressionOfIntegrand , $PostprocessTag1$ , $PostprocessTag2$
, ... )
INTEGRATION ( $IntInd37$ ) = ( %MAXIMUM_BE%, ExpressionOfIntegrand , $PostprocessTag1$ ,
$PostprocessTag2$ , ... )
```

Maximum, minimum and sum with respect to the boundary nodes

```
INTEGRATION ( $IntInd38$ ) = ( %MINIMUM_BENP%, ExpressionOfIntegrand , $PostprocessTag1$ ,
$PostprocessTag2$ , ... )
INTEGRATION ( $IntInd39$ ) = ( %MAXIMUM_BENP%, ExpressionOfIntegrand , $PostprocessTag1$ ,
$PostprocessTag2$ , ... )
INTEGRATION ( $IntInd40$ ) = ( %SUM_BENP%, ExpressionOfIntegrand , $PostprocessTag1$ , $PostprocessTag2$ ,
... )
```

$IntInd...$: The user can (uniquely) choose these soft variables at will. There is no need for any further definition in USER_common_variables .
They can be used to incorporate the result of the corresponding integration statement into an equation with the help of the integ() -function.

Assign function values to points

```
INTEGRATION ( $IntInd41$ ) = ( %ASSIGN_FUNCTIONVALUE% , %ind_f%, [AssignedFunctionValue], $MaterialTag1$
, $MaterialTag2$ , ... )
```

Good to know:

- The soft variables are optional. If none is given, then MESHFREE counts the number of integration statements by their appearance in USER_common_variables .
  Warning: The syntax with and without soft variables must not be mixed.
- Instead of a soft variable $IntInd$ also the legacy syntax with natural number n is possible. In this case all integration statements in USER_common_variables have to be numbered consecutively to prevent overwriting.
- Example for setting the **$PostprocessTag$** can be found under POSTPROCESS .
- Example for setting the **$MaterialTag$** can be found under MAT .
- The scalar ExpressionOfIntegrand is a typical RightHandSideExpression .
- The evaluated statement can be also incorporated into equations by using the function integ() .

For details on the specific statements, the SelectionFeatures , and HeaderInfoOrComments see below.

List of members:

ExpressionOfIntegrand	scalar expression to integrate with respect to a given region
Skip	additional options to skip computation of integrations
TimestepFile	Results of INTEGRATION statements per timestep
SequentialFiltering	generate writeout to timestep files due to simple sequential filters
SelectionFeatures	additional options to further select MESHFREE integration points for integration
HeaderInfoOrComments	add comments for integration
AppendDataToExistingFiles	append INTEGRATION results to an existing .timestep file
%ASSIGN_FUNCTIONVALUE%	assign a function value to selected MESHFREE points
%INTEGRATION_INT%	volume integration of a functional with respect to a given material
%INTEGRATION_INT_TIME%	volume and time integration of a functional with respect to a given material
%INTEGRATION_BND_DIRECT%	surface integration of a scalar value along pieces of boundary

%INTEGRATION_BND_DIRECT_TIME%	surface and time integration of a scalar value along pieces of boundary
%INTEGRATION_FS_DIRECT%	surface integration of a scalar value along the free surface
%INTEGRATION_FS_DIRECT_TIME%	surface and time integration of a scalar value along the free surface
%INTEGRATION_BND%	surface integration of a vector valued function along pieces of boundary
%INTEGRATION_BND_TIME%	surface and time integration of a vector valued function along pieces of boundary
%INTEGRATION_FS%	surface integration of a vector valued function along the free surface
%INTEGRATION_FS_TIME%	surface and time integration of a vector valued function along the free surface
%INTEGRATION_FLUX%	flux integration of a functional by counting the MESHFREE points that slip over a given control surface
%INTEGRATION_FLUX_TIME%	time and flux integration of a functional by counting the MESHFREE points that slip over a given control surface
%INTEGRATION_ABSFLUX%	flux integration of a functional by counting the MESHFREE points that slip over a given control surface independent of the direction
%INTEGRATION_ABSFLUX_TIME%	time and flux integration of a functional by counting the MESHFREE points that slip over a given control surface independent of the direction
%INTEGRATION_FLUX_DROPLETPHASE%	flux integration of a functional by counting the DROPLETPHASE points that slip over a given control surface
%MASSFLOW_DROPLETPHASE%	mass flux integration of a functional by counting the DROPLETPHASE points that are injected at all inflow surfaces
%MAXIMUM_INT%	maximum of a functional based on all MESHFREE points with respect to given material flags
%MINIMUM_INT%	minimum of a functional based on all MESHFREE points with respect to given material flags
%SUMMATION_INT%	summation of given function values based on all MESHFREE points with respect to given material flags
%AVERAGE_INT%	average of a functional based on all MESHFREE points with respect to given material flags
%MAXIMUM_BND%	maximum of a functional based on all MESHFREE boundary points with respect to given boundary elements
%MINIMUM_BND%	minimum of a functional based on all MESHFREE boundary points with respect to given boundary elements
%SUMMATION_BND%	summation of given function values based on all MESHFREE boundary points with respect to given boundary elements
%AVERAGE_BND%	average of a functional based on all MESHFREE boundary points with respect to given boundary elements

%MAXIMUM_FS%	maximum of a functional based on all MESHFREE free surface points with respect to given material flags
%MINIMUM_FS%	minimum of a functional based on all MESHFREE free surface points with respect to given material flags
%AVERAGE_FS%	average of a functional based on all MESHFREE free surface points with respect to given material flags
%POINT_DIRECT%	write simple values like position, chamber index etc. of a BND_point to file
%POINT_APPROXIMATE%	approximation of a functional at a BND_point by MESHFREE interpolation
%POINT_APPROXIMATE_ProjBNDOnly%	approximation of a functional at a BND_point by MESHFREE interpolation with respect to neighboring boundary points
%PUBLICVALUE%	public value of MESHFREE simulation
%PUBLICVALUE_TIME%	time-integrated public value of MESHFREE simulation
%PUBLICVALUE_SUM%	summed public value of MESHFREE simulation
%PUBLICVALUE_CLOCKstatistics%	CLOCK value of given stop watch
%PUBLICVALUE_CPUstatistics%	CPU value of given stop watch
%BE_INTEGRATION_DIRECT%	surface integration of a scalar value on boundary elements
%BE_INTEGRATION_DIRECT_TIME%	surface and time integration of a scalar value on boundary elements

%ASSIGN_FUNCTIONVALUE%

assign a function value to selected MESHFREE points

This function is rather not a typical INTEGRATION , as it does not reduce values of MESHFREE points to a scalar.
On the other hand, it is useful to assign values within the INTEGRATION -sequence, in order to use previous integration results and to use assigned values in later integations.

```
INTEGRATION ( $IntInd29$ ) = ( %ASSIGN_FUNCTIONVALUE% , %ind_f%, [AssignedFunctionValue], $MaterialTag1$ , $MaterialTag2$ , ... )
```

%ind_f% -> where to save the assigned values
[AssignedFunctionValue] -> what function value to assign (as usual, this can be anything in the framework of RightHandSideExpression)

The assignment is restricted to the MESHFREE points belonging to the given MaterialTags, and can be further restricted by the SelectionFeatures .
Neverthless, also this intergratin item will produce an entry in the timestep-file, which shall usually be zero.

```
INTEGRATION ( $INT5$ ) = ( %ASSIGN_FUNCTIONVALUE% , %indU_1%, [Y %ind_IN_glob% ], $MAT$ , %INTEGRATION_Header%, "assign global point index to indU_1" ) # test
```

%AVERAGE_BND%

average of a functional based on all MESHFREE boundary points with respect to given boundary elements

Average of a given functional f (ExpressionOfIntegrand) with respect to the set P_{BND} of all MESHFREE boundary points with given POSTPROCESS -flags:

$$I_{\mathrm{AvgBND}} = \frac{1}{\#P_{\mathrm{BND}}} \sum_{i \in P_{\mathrm{BND}}} f_i,$$

where $\#P_{\mathrm{BND}}$ denotes the number of points in P_{BND} .

Example:

```
begin_alias{ }
"Alias1" = " ... POSTPROCESS$PostprocessTag1$ ... " # definition of Alias1
"Alias2" = " ... POSTPROCESS$PostprocessTag2$ ... " # definition of Alias2
end_alias
INTEGRATION ( $IntInd$ ) = ( %AVERAGE_BND% , ExpressionOfIntegrand , $PostprocessTag1$ ,
$PostprocessTag2$ )
```

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %AVERAGE_FS%

%AVERAGE_FS%

average of a functional based on all MESHFREE free surface points with respect to given material flags

Average of a given functional f (ExpressionOfIntegrand) with respect to the set P_{FS} of all MESHFREE free surface points with given material flags:

$$I_{\mathrm{AvgFS}} = \frac{1}{\#P_{\mathrm{FS}}} \sum_{i \in P_{\mathrm{FS}}} f_i,$$

where $\#P_{\mathrm{FS}}$ denotes the number of points in P_{FS} .

Example:

```
begin_alias{ }
"Alias1" = " ... MAT$MaterialTag1$ ... " # definition of Alias1
"Alias2" = " ... MAT$MaterialTag2$ ... " # definition of Alias2
end_alias
INTEGRATION ( $IntInd$ ) = ( %AVERAGE_FS% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ )
```

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %AVERAGE_INT%

%AVERAGE_INT%

average of a functional based on all MESHFREE points with respect to given material flags

Average of a given functional f (ExpressionOfIntegrand) with respect to the set P of all MESHFREE points with given material flags:

$$I_{\mathrm{Avg}} = \frac{1}{\#P} \sum_{i \in P} f_i,$$

where $\#P$ denotes the number of points in P .

Example:

```
begin_alias{ }
"Alias1" = " ... MAT$MaterialTag1$ ... " # definition of Alias1
"Alias2" = " ... MAT$MaterialTag2$ ... " # definition of Alias2
end_alias
INTEGRATION ( $IntInd$ ) = ( %AVERAGE_INT% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ )
```

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %BE_INTEGRATION_DIRECT%

%BE_INTEGRATION_DIRECT%

surface integration of a scalar value on boundary elements

```
INTEGRATION ( $IntInd$ ) = ( %BE_INTEGRATION_DIRECT% , ExpressionOfIntegrand , $PostprocessTag1$ , $PostprocessTag2$ , ... )
```

The POSTPROCESS -flags **$PostprocessTag1$** , **$PostprocessTag2$** , ... define the IntegrationArea . Their number is not limited.

This computes the integral of a functional f (ExpressionOfIntegrand) with respect to the region $\partial\Omega$ identified by the POSTPROCESS -flags

$$I_{\mathrm{BEDirect}} = \int_{\partial\Omega} f dA$$

by a sum approximation

$$I_{\mathrm{BEDirect}} \approx \sum_{i \in BE} f_i \cdot A_i,$$

where BE is the set of all boundary elements with the given postprocess flags.
f_i is the function value and A_i is the area of the i-th boundary element.

Example:

```
INTEGRATION ( $area_PostprocessTag1$ ) = ( %BE_INTEGRATION_DIRECT% , [1.0], $PostprocessTag1$ )
```

Note: In contrast to %INTEGRATION_BND_DIRECT% , ExpressionOfIntegrand is defined and evaluated on the boundary elements and not on the MESHFREE point cloud!

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %BE_INTEGRATION_DIRECT_TIME%

%BE_INTEGRATION_DIRECT_TIME%

surface and time integration of a scalar value on boundary elements

```
INTEGRATION ( $IntInd$ ) = ( %BE_INTEGRATION_DIRECT_TIME% , ExpressionOfIntegrand , $PostprocessTag1$ , $PostprocessTag2$ , ... )
```

The POSTPROCESS -flags **$PostprocessTag1$** , **$PostprocessTag2$** , ... define the IntegrationArea . Their number is not limited.

This computes the integral of a functional f (ExpressionOfIntegrand) with respect to the region $\partial\Omega$ identified by the POSTPROCESS -flags

$$I_{\mathrm{BEDirectTime}} = \int_{t_0}^{t_{n+1}} \int_{\partial\Omega} f(t) dA dt$$

by a preliminary approximation

$$I_{\mathrm{BEDirect}} \approx \sum_{i \in BE} f_i\left(t_{n+1}\right) \cdot A_i\left(t_{n+1}\right)$$

and a subsequent time integration:

$$I_{\mathrm{BEDirectTime}}\left(t_{n+1}\right) = I_{\mathrm{BEDirectTime}}\left(t_n\right) + \left(t_{n+1} - t_n\right) \cdot I_{\mathrm{BEDirect}}$$

BE is the set of all boundary elements with the given postprocess flags.
f_i is the function value and A_i is the area of the i-th boundary element.

Example:

```
INTEGRATION ( $time_area_PostprocessTag1$ ) = ( %BE_INTEGRATION_DIRECT_TIME% , [1.0], $PostprocessTag1$ )
```

Note: In contrast to %INTEGRATION_BND_DIRECT_TIME% , ExpressionOfIntegrand is defined and evaluated on the boundary elements and not on the MESHFREE point cloud!

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %INTEGRATION_ABSFLUX%

%INTEGRATION_ABSFLUX%

flux integration of a functional by counting the MESHFREE points that slip over a given control surface independent of the direction

```
begin_alias{ }
"AliasOmega" = " ... IDENT%BND_BlindAndEmpty% ... POSTPROCESS$PostprocessTag$ ... " # definition of AliasOmega
end_alias
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_ABSFLUX% , ExpressionOfIntegrand , $PostprocessTag$ )
```

Warning: %INTEGRATION_ABSFLUX% as well as %INTEGRATION_ABSFLUX_TIME% work only for boundary elements marked with %BND_BlindAndEmpty% .

It computes the flux of a functional f (ExpressionOfIntegrand) across a control surface in the sense:

$$I_{\mathrm{AbsFlux}} = \int_{\partial\Omega} f \cdot \left| \boldsymbol{v}^T \boldsymbol{n} \right| dA$$

This integral is approximated by summing up the MESHFREE points which are currently penetrating through the control surface $\partial\Omega$:

$$I_{\mathrm{AbsFlux}} \approx \sum_{i \in P_{\mathrm{slipped}}} f_i \cdot \frac{V_i}{\Delta t}$$

P_{slipped} is the set of all MESHFREE points which slipped over $\partial\Omega$ in this time step.

Here, the direction of penetration of a MESHFREE point does not matter.

Note: Skip is not recommended for this type of integration statement.

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %INTEGRATION_ABSFLUX_TIME%

%INTEGRATION_ABSFLUX_TIME%

time and flux integration of a functional by counting the MESHFREE points that slip over a given control surface independent of the direction

This is the time integration of %INTEGRATION_ABSFLUX% :

$$I_{\mathrm{AbsFluxTime}} = \int I_{\mathrm{AbsFlux}} dt \approx \sum_{i=\mathrm{AllTimeSteps}} I_{\mathrm{AbsFlux},i} \cdot \Delta t_i$$

Note: Skip is not recommended for this type of integration statement.

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %INTEGRATION_BND%

%INTEGRATION_BND%

surface integration of a vector valued function along pieces of boundary

```
begin_alias{ }
"Alias1" = " ... IDENT%BND_wall% ... MAT$MaterialTag$ ... BC$BCindex$ ... POSTPROCESS$PostprocessTag1$ ... "
# definition of Alias1
"Alias2" = " ... IDENT%BND_wall% ... MAT$MaterialTag$ ... BC$BCindex$ ... POSTPROCESS$PostprocessTag2$ ... "
# definition of Alias2
end_alias
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_BND% , Integrand_x, Integrand_y, Integrand_z, $PostprocessTag1$ ,
$PostprocessTag2$ , ... )
```

This computes the integral with respect to the region $\partial\Omega$ identified by the POSTPROCESS -flags

$$I_{\mathrm{Bnd}} = \int_{\partial\Omega} \boldsymbol{u} \cdot \boldsymbol{n} dA$$

by a sum approximation

$$I_{\mathrm{Bnd}} \approx \sum_{i \in P} (\boldsymbol{u}_i \cdot \boldsymbol{n}_i) A_i,$$

where $\boldsymbol{n}$ represents the local boundary normal. The integrand $\boldsymbol{u}$ is given by the vector (**Integrand_x** , **Integrand_y** , **Integrand_z**), whose components are all of type ExpressionOfIntegrand . P is the set of all boundary points with the given postprocess flags and A_i is the area of the i-th point.

The POSTPROCESS -flags **$PostprocessTag1$** , **$PostprocessTag2$** , ... define the IntegrationArea . Their number is not limited.

Example:

```
INTEGRATION ( $pressure_x$ ) = ( %INTEGRATION_BND% , [Y %ind_p% +Y %ind_p_dyn% ], [0], [0],
$PostprocessTag1$ , $PostprocessTag2$ , $PostprocessTag3$ )
```

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %INTEGRATION_BND_DIRECT%

%INTEGRATION_BND_DIRECT%

surface integration of a scalar value along pieces of boundary

```
begin_alias{ }
"Alias1" = " ... IDENT%BND_wall% ... MAT$MaterialTag$ ... BC$BCindex$ ... POSTPROCESS$PostprocessTag1$ ... "
# definition of Alias1
"Alias2" = " ... IDENT%BND_wall% ... MAT$MaterialTag$ ... BC$BCindex$ ... POSTPROCESS$PostprocessTag2$ ... "
# definition of Alias2
end_alias
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_BND_DIRECT% , ExpressionOfIntegrand , $PostprocessTag1$ ,
$PostprocessTag2$ , ... )
```

The POSTPROCESS -flags **$PostprocessTag1$** , **$PostprocessTag2$** , ... define the IntegrationArea . Their number is not limited.

This computes the integral of a functional f (ExpressionOfIntegrand) with respect to the region $\partial\Omega$ identified by the POSTPROCESS -flags

$$I_{\mathrm{BndDirect}} = \int_{\partial\Omega} f dA$$

by a sum approximation

$$I_{\mathrm{BndDirect}} \approx \sum_{i \in P_{Bnd}} f_i \cdot A_i,$$

where P_{Bnd} is the set of all boundary points with the given POSTPROCESS -flags and A_i is the area of the i-th point.

Example:

```
INTEGRATION ( $IntInd1$ ) = ( %INTEGRATION_BND_DIRECT% , [Y %ind_p% +Y %ind_p_dyn% ],
$PostprocessTag1$ , $PostprocessTag2$ , $PostprocessTag3$ )
INTEGRATION ( $IntInd2$ ) = ( %INTEGRATION_BND_DIRECT% , equn{ $EqnName$ }, $PostprocessTag1$ ,
$PostprocessTag2$ , $PostprocessTag3$ )
INTEGRATION ( $IntInd3$ ) = ( %INTEGRATION_BND_DIRECT% , curve{ $CrvName$ }depvar{%ind_DepVar%},
$PostprocessTag1$ , $PostprocessTag2$ , $PostprocessTag3$ )
```

List of members:

IntegrationArea	list of flags taggig the region with respect to which the integration is performed

IntegrationArea

list of flags taggig the region with respect to which the integration is performed

List of flags $PostprocessFlag1$, $PostprocessFlag2$, ... which have to be defined in the alias section (see AliasForGeometryItems)
by attributes of the form POSTPROCESS $PostprocessFlag1$, POSTPROCESS $PostprocessFlag2$, ...

%INTEGRATION_BND_DIRECT_TIME%

surface and time integration of a scalar value along pieces of boundary

```
begin_alias{ }
"Alias1" = " ... IDENT%BND_wall% ... MAT$MaterialTag$ ... BC$BCindex$ ... POSTPROCESS$PostprocessTag1$ ... "
# definition of Alias1
"Alias2" = " ... IDENT%BND_wall% ... MAT$MaterialTag$ ... BC$BCindex$ ... POSTPROCESS$PostprocessTag2$ ... "
# definition of Alias2
end_alias
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_BND_DIRECT_TIME% , ExpressionOfIntegrand , $PostprocessTag1$
, $PostprocessTag2$ , ... )
```

The POSTPROCESS -flags **$PostprocessTag1$** , **$PostprocessTag2$** , ... define the IntegrationArea . Their number is not limited.

This computes the integral of a functional f (ExpressionOfIntegrand) with respect to the region $\partial\Omega$ identified by the POSTPROCESS -flags

$$I_{\mathrm{BndDirectTime}} = \int_{t_0}^{t_{n+1}} \int_{\partial\Omega} f(t) dA dt$$

by a preliminary approximation

$$I_{\mathrm{BndDirect}} \approx \sum_{i \in P_{Bnd}} f_i\left(t_{n+1}\right) \cdot A_i\left(t_{n+1}\right)$$

and a subsequent time integration:

$$I_{\text{BndDirectTime}}(t_{n+1}) = I_{\text{BndDirectTime}}(t_n) + (t_{n+1} - t_n) \cdot I_{\text{BndDirect}}$$

P_{Bnd} is the set of all boundary points with the given postprocess flags and A_i is the area of the i-th point.

Example:

```
INTEGRATION ( $IntInd1$ ) = ( %INTEGRATION_BND_DIRECT_TIME% , [Y %ind_p% +Y %ind_p_dyn% ],
$PostprocessTag1$ , $PostprocessTag2$ , $PostprocessTag3$ )
INTEGRATION ( $IntInd2$ ) = ( %INTEGRATION_BND_DIRECT_TIME% , equn{ $EqnName$ }, $PostprocessTag1$ ,
$PostprocessTag2$ , $PostprocessTag3$ )
INTEGRATION ( $IntInd3$ ) = ( %INTEGRATION_BND_DIRECT_TIME% , curve{ $CrvName$
}depvar{%ind_DepVar%}, $PostprocessTag1$ , $PostprocessTag2$ , $PostprocessTag3$ )
```


%INTEGRATION_BND_TIME%

surface and time integration of a vector valued function along pieces of boundary

```
begin_alias{ }
"Alias1" = " ... IDENT%BND_wall% ... MAT$MaterialTag$ ... BC$BCindex$ ... POSTPROCESS$PostprocessTag1$ ... "
# definition of Alias1
"Alias2" = " ... IDENT%BND_wall% ... MAT$MaterialTag$ ... BC$BCindex$ ... POSTPROCESS$PostprocessTag2$ ... "
# definition of Alias2
end_alias
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_BND_TIME% , Integrand_x, Integrand_y, Integrand_z,
$PostprocessTag1$ , $PostprocessTag2$ , ... )
```

This computes the integral with respect to the region $\partial\Omega$ identified by the POSTPROCESS -flags

$$I_{\text{BndTime}} = \int_{t_0}^{t_{n+1}} \int_{\partial\Omega} \boldsymbol{u}(t) \cdot \boldsymbol{n}(t) dA dt$$

by a preliminary approximation

$$I_{\text{Bnd}} \approx \sum_{i \in P} \left(\boldsymbol{u}_i(t_{n+1}) \cdot \boldsymbol{n}_i(t_{n+1}) \right) A_i(t_{n+1})$$

and a subsequent time integration:

$$I_{\text{BndTime}}(t_{n+1}) = I_{\text{BndTime}}(t_n) + (t_{n+1} - t_n) \cdot I_{\text{Bnd}}$$

$\boldsymbol{n}$ represents the local boundary normal. The integrand $\boldsymbol{u}$ is given by the vector (**Integrand_x** , **Integrand_y** , **Integrand_z**), whose components are all of type ExpressionOfIntegrand . P is the set of all boundary points with the given postprocess flags and A_i is the area of the i-th point.

The POSTPROCESS -flags **$PostprocessTag1$** , **$PostprocessTag2$** , ... define the IntegrationArea . Their number is not limited.

Example:

```
INTEGRATION ( $pressure_x$ ) = ( %INTEGRATION_BND_TIME% , [Y %ind_p% +Y %ind_p_dyn% ], [0], [0],
$PostprocessTag1$ , $PostprocessTag2$ , $PostprocessTag3$ )
```

%INTEGRATION_FLUX%

flux integration of a functional by counting the MESHFREE points that slip over a given control surface

```
begin_alias{ }
"AliasOmega" = " ... IDENT%BND_BlindAndEmpty% ... POSTPROCESS$PostprocessTag$ ... " # definition of
AliasOmega
end_alias
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_FLUX% , ExpressionOfIntegrand , $PostprocessTag$ )
```

Warning: %INTEGRATION_FLUX% as well as %INTEGRATION_FLUX_TIME% work only for boundary elements marked with IDENT %BND_BlindAndEmpty% .

It computes the flux of a functional f (ExpressionOfIntegrand) across a control surface in the sense:

$$I_{\text{Flux}} = \int_{\partial\Omega} f \cdot (\boldsymbol{v}^T \boldsymbol{n}) dA$$

This integral is approximated by summing up the MESHFREE points which are currently penetrating through the control surface $\partial\Omega$:

$$I_{\text{Flux}} \approx \sum_{i \in P_{\text{slipped}}} f_i \cdot \text{sgn}(\boldsymbol{v}_i^T \cdot \boldsymbol{n}_i) \frac{V_i}{\Delta t}$$

P_{slipped} is the set of all MESHFREE points which slipped over $\partial\Omega$ in this time step.

The term $\text{sgn}(\boldsymbol{v}_i^T \cdot \boldsymbol{n}_i)$ accounts for the direction the MESHFREE point goes through the control surface.

If the dependency from the direction should be ignored, the net value can be integrated by:

```
begin_alias{ }
"AliasOmega" = " ... IDENT%BND_BlindAndEmpty% ... POSTPROCESS$PostprocessTag$ ... " # definition of
AliasOmega
end_alias
begin_construct{ }
"nOmega" = CONSTRUCT ( %CONSTRUCT_Normal% , "AliasOmega" ) # definition of nOmega
end_construct
begin_equation{ $LeftOrRight$ }
if ( Y %ind_v(1)% * &nOmega(1)& + Y %ind_v(2)% * &nOmega(2)& + Y %ind_v(3)% * &nOmega(3)& > 0 ) :: 1.0
else :: -1.0
endif
end_equation
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_FLUX% , [equn( $LeftOrRight$ )*(Functional)], $PostprocessTag$ )
```

Integration without dependency of the direction of passage through the control surface is given by %INTEGRATION_ABSFLUX% and %INTEGRATION_ABSFLUX_TIME% .

Note: Skip is not recommended for this type of integration statement.

DOWNLOAD COMPREHENSIVE EXAMPLE

%INTEGRATION_FLUX_DROPLETPHASE%

flux integration of a functional by counting the DROPLETPHASE points that slip over a given control surface

```
begin_alias{ }
"AliasOmega" = " ... IDENT%BND_BlindAndEmpty% ... POSTPROCESS$PostprocessTag$ ... " # definition of
AliasOmega
end_alias
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_FLUX_DROPLETPHASE% , ExpressionOfIntegrand ,
$PostprocessTag$ )
```

Warning: %INTEGRATION_FLUX_DROPLETPHASE% only works for boundary elements marked with %BND_BlindAndEmpty% .

It computes the flux of a functional f (ExpressionOfIntegrand) across a control surface in the sense:

$$I_{\mathrm{FluxDrops}} = \int_{\partial\Omega} f \cdot (\boldsymbol{v}^T \boldsymbol{n}) dA$$

This integral is approximated by summing up the DROPLETPHASE points which are currently penetrating through the control surface $\partial\Omega$:

$$I_{\mathrm{FluxDrops}} \approx \sum_{i \in P_{\mathrm{slippedDrops}}} f_i \cdot \mathrm{sgn}(\boldsymbol{v}_i^T \cdot \boldsymbol{n}_i) \frac{V_i}{\Delta t}$$

$P_{\mathrm{slippedDrops}}$ is the set of all DROPLETPHASE points which slipped over $\partial\Omega$ in this time step.

The term $\mathrm{sgn}(\boldsymbol{v}_i^T \cdot \boldsymbol{n}_i)$ accounts for the direction the DROPLETPHASE point goes through the control surface.

The current volume of a DROPLETPHASE point is determined by:

$$V_i = \frac{\pi}{6} d_i^3,$$

where d_i is the mean diameter of the DROPLETPHASE (see %ind_d30%).

If the dependency from the direction should be ignored, the net value can be integrated by:

```
begin_alias{ }
"AliasOmega" = " ... IDENT%BND_BlindAndEmpty% ... POSTPROCESS$PostprocessTag$ ... " # definition of
AliasOmega
end_alias
begin_construct{ }
"nOmega" = CONSTRUCT ( %CONSTRUCT_Normal% , "AliasOmega" ) # definition of nOmega
end_construct
begin_equation{ $LeftOrRight$ }
if ( Y %ind_v(1)% * &nOmega(1)& + Y %ind_v(2)% * &nOmega(2)& + Y %ind_v(3)% * &nOmega(3)& > 0 ) :: 1.0
else :: -1.0
endif
end_equation
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_FLUX_DROPLETPHASE% , [equn( $LeftOrRight$ )*(Functional)],
$PostprocessTag$ )
```

Note:

- This integration is analogous to %INTEGRATION_FLUX% . However, the DROPLETPHASE points are used instead of the classical LIQUID points. For details on the Solvers , see KindOfProblem and the respective links.
- Skip is not recommended for this type of integration statement.

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %INTEGRATION_FLUX_TIME%

%INTEGRATION_FLUX_TIME%

time and flux integration of a functional by counting the MESHFREE points that slip over a given control surface

This is the time integration of %INTEGRATION_FLUX% :

$$I_{\mathrm{FluxTime}} = \int I_{\mathrm{Flux}} dt \approx \sum_{i=\mathrm{AllTimeSteps}} I_{\mathrm{Flux},i} \cdot \Delta t_i$$

Note: Skip is not recommended for this type of integration statement.

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %INTEGRATION_FS%

%INTEGRATION_FS%

surface integration of a vector valued function along the free surface

```
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_FS% , Integrand_x, Integrand_y, Integrand_z, $MaterialTag$ )
```

This computes the integral with respect to the free surface $\partial\Omega_{\mathrm{FS}}$ identified by the material flag

$$I_{\mathrm{FS}} = \int_{\partial\Omega_{\mathrm{FS}}} \boldsymbol{u} \cdot \boldsymbol{n} dA$$

by a sum approximation

$$I_{\mathrm{FS}} \approx \sum_{i \in P_{\mathrm{FS}}} (\boldsymbol{u}_i \cdot \boldsymbol{n}_i) A_i,$$

where $\boldsymbol{n}$ represents the local free surface normal. The integrand $\boldsymbol{u}$ is given by the vector (**Integrand_x** , **Integrand_y** , **Integrand_z**), whose components are all of type ExpressionOfIntegrand . P_{FS} is the set of all boundary points with the given material flag and A_i is the area of the i-th point.

The material flag $MaterialTag$ defines the integration area (analogous to the POSTPROCESS -flags for %INTEGRATION_BND%).

Note: Analogous to %INTEGRATION_INT% , a list of material flags can be used to specify the integration area. The number of flags is not limited.

Example:

```
INTEGRATION ( $pressure_x$ ) = ( %INTEGRATION_FS% , [Y %ind_p% +Y %ind_p_dyn% ], [0], [0], $MaterialTag$ )
```

Note: In case of multiphase simulations with detection of interface connections (see PHASE_distinction), the interface points are treated like free surface points.

%INTEGRATION_FS_DIRECT%

surface integration of a scalar value along the free surface

```
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_FS_DIRECT% , ExpressionOfIntegrand , $MaterialTag$ )
```

The material flag $MaterialTag$ defines the integration area (analogous to the POSTPROCESS -flags for %INTEGRATION_BND_DIRECT%).

This computes the integral of a functional f (ExpressionOfIntegrand) with respect to the free surface $\partial\Omega_{\mathrm{FS}}$ identified by the material flag

$$I_{\mathrm{FSDirect}} = \int_{\partial\Omega_{\mathrm{FS}}} f dA$$

by a sum approximation

$$I_{\mathrm{FSDirect}} \approx \sum_{i \in P_{\mathrm{FS}}} f_i \cdot A_i,$$

where P_{FS} is the set of all free surface points with the given material flag and A_i is the area of the i-th point.

Note: Analogous to %INTEGRATION_INT% , a list of material flags can be used to specify the integration area. The number of flags is not limited.

Example:

```
INTEGRATION ( $IntInd1$ ) = ( %INTEGRATION_FS_DIRECT% , [Y %ind_p% +Y %ind_p_dyn% ], $MaterialTag$ )
INTEGRATION ( $IntInd2$ ) = ( %INTEGRATION_FS_DIRECT% , equn{ $EqnName$ }, $MaterialTag1$ ,
$MaterialTag2$ , $MaterialTag3$ )
INTEGRATION ( $IntInd3$ ) = ( %INTEGRATION_FS_DIRECT% , curve{ $CrvName$ }depvar{%ind_DepVar%},
$MaterialTag$ )
```

Note: In case of multiphase simulations with detection of interface connections (see PHASE_distinction), the interface points are treated like free surface points.

%INTEGRATION_FS_DIRECT_TIME%

surface and time integration of a scalar value along the free surface

```
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_FS_DIRECT_TIME% , ExpressionOfIntegrand , $MaterialTag$ )
```

The material flag $MaterialTag$ defines the integration area (analogous to the POSTPROCESS -flags for %INTEGRATION_BND_DIRECT_TIME%).

This computes the integral of a functional f (ExpressionOfIntegrand) with respect to the free surface $\partial\Omega_{\mathrm{FS}}$ identified by the material flag

$$I_{\mathrm{FSDirectTime}} = \int_{t_0}^{t_{n+1}} \int_{\partial\Omega_{\mathrm{FS}}} f(t) dA dt$$

by a preliminary approximation

$$I_{\mathrm{FSDirect}} \approx \sum_{i \in P_{\mathrm{FS}}} f_i\left(t_{n+1}\right) \cdot A_i\left(t_{n+1}\right)$$

and a subsequent time integration:

$$I_{\mathrm{FSDirectTime}}\left(t_{n+1}\right) = I_{\mathrm{FSDirectTime}}\left(t_n\right) + \left(t_{n+1} - t_n\right) \cdot I_{\mathrm{FSDirect}}$$

P_{FS} is the set of all free surface points with the given material flag and A_i is the area of the i-th point.

Note: Analogous to %INTEGRATION_INT_TIME% , a list of material flags can be used to specify the integration area. The number of flags is not limited.

Example:

```
INTEGRATION ( $IntInd1$ ) = ( %INTEGRATION_FS_DIRECT_TIME% , [Y %ind_p% +Y %ind_p_dyn% ], $MaterialTag$ )
INTEGRATION ( $IntInd2$ ) = ( %INTEGRATION_FS_DIRECT_TIME% , equn{ $EqnName$ }, $MaterialTag1$ , $MaterialTag2$ , $MaterialTag3$ )
INTEGRATION ( $IntInd3$ ) = ( %INTEGRATION_FS_DIRECT_TIME% , curve{ $CrvName$ }depvar{%ind_DepVar%}, $MaterialTag$ )
```

Note: In case of multiphase simulations with detection of interface connections (see PHASE_distinction), the interface points are treated like free surface points.

%INTEGRATION_FS_TIME%

surface and time integration of a vector valued function along the free surface

```
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_FS_TIME% , Integrand_x, Integrand_y, Integrand_z, $MaterialTag$ )
```

This computes the integral with respect to the free surface $\partial\Omega_{\mathrm{FS}}$ identified by the material flag

$$I_{\mathrm{FSTime}} = \int_{t_0}^{t_{n+1}} \int_{\partial\Omega_{\mathrm{FS}}} \boldsymbol{u}(t) \cdot \boldsymbol{n}(t) dA dt$$

by a preliminary approximation

$$I_{\mathrm{FS}} \approx \sum_{i \in P_{\mathrm{FS}}} \left(\boldsymbol{u}_i\left(t_{n+1}\right) \cdot \boldsymbol{n}_i\left(t_{n+1}\right)\right) A_i\left(t_{n+1}\right)$$

and a subsequent time integration:

$$I_{\mathrm{FSTime}}\left(t_{n+1}\right) = I_{\mathrm{FSTime}}\left(t_n\right) + \left(t_{n+1} - t_n\right) \cdot I_{\mathrm{FS}}$$

$\boldsymbol{n}$ represents the local free surface normal. The integrand $\boldsymbol{u}$ is given by the vector (**Integrand_x** , **Integrand_y** , **Integrand_z**), whose components are all of type ExpressionOfIntegrand . P_{FS} is the set of all free surface points with the given material flag and A_i is the area of the i-th point.

The material flag $MaterialTag$ defines the integration area (analogous to the POSTPROCESS -flags for %INTEGRATION_BND%).

Note: Analogous to %INTEGRATION_INT_TIME% , a list of material flags can be used to specify the integration area. The number of flags is not limited.

Example:

```
INTEGRATION ( $pressure_x$ ) = ( %INTEGRATION_FS_TIME% , [Y %ind_p% +Y %ind_p_dyn% ], [0], [0], $PostprocessTag1$ , $PostprocessTag2$ , $PostprocessTag3$ )
```

Note: In case of multiphase simulations with detection of interface connections (see PHASE_distinction), the interface points are treated like free surface points.

%INTEGRATION_INT%

volume integration of a functional with respect to a given material

```
begin_alias{ }
"AliasOmega" = " ... IDENT%BND_wall% ... MAT$MaterialTag$ ... BC$BCindex$ ... " # definition of AliasOmega
end_alias
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_INT% , ExpressionOfIntegrand , $MaterialTag$ )
```

This computes the integral of a functional f (ExpressionOfIntegrand) with respect to the region Ω identified by the material flag $MaterialTag$

$$I = \int_{\Omega} f dV$$

by a sum approximation

$$I \approx \sum_{i \in P} f_i \cdot V_i,$$

where P is the set of all points with the given material flag and V_i is the volume of the i-th point.

Note: Analogous to %INTEGRATION_BND% , a list of material flags can be used to specify the integration region. The number of flags is not limited.

Example:

- volume of a material

  ```
  INTEGRATION ( $volume$ ) = ( %INTEGRATION_INT% , [1], $MaterialTag$ )
  ```

- kinetic energy of a material

  ```
  INTEGRATION ( $energy$ ) = ( %INTEGRATION_INT% , [0.5*Y %ind_r% *(Y %ind_v(1)% ^2 + Y %ind_v(2)% ^2 + Y %ind_v(3)% ^2)], $MaterialTag$ )
  ```

%INTEGRATION_INT_TIME%

volume and time integration of a functional with respect to a given material

```
begin_alias{ }
"AliasOmega" = " ... IDENT%BND_wall% ... MAT$MaterialTag$ ... BC$BCindex$ ... " # definition of AliasOmega
end_alias
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_INT_TIME% , ExpressionOfIntegrand , $MaterialTag$ )
```

This computes the integral of a functional f (ExpressionOfIntegrand) with respect to the region Ω identified by the material flag $MaterialTag$

$$I_{\text{Time}} = \int_{t_0}^{t_{n+1}} \int_{\Omega} f(t) dV dt$$

by a preliminary approximation

$$I \approx \sum_{i \in P} f_i \left(t_{n+1}\right) \cdot V_i \left(t_{n+1}\right)$$

and a subsequent time integration:

$$I_{\text{Time}} \left(t_{n+1}\right) = I_{\text{Time}} \left(t_n\right) + \left(t_{n+1} - t_n\right) \cdot I$$

P is the set of all points with the given material flag and V_i is the volume of the i-th point.

Note: Analogous to %INTEGRATION_BND_TIME% , a list of material flags can be used to specify the integration region. The number of flags is not limited.

Example: total turbulent dissipation of some material

```
INTEGRATION ( $dissipation$ ) = ( %INTEGRATION_INT_TIME% , [Y %ind_eps% ], $MaterialTag$ )
```


%MASSFLOW_DROPLETPHASE%

mass flux integration of a functional by counting the DROPLETPHASE points that are injected at all inflow surfaces

```
INTEGRATION ( $IntInd$ ) = ( %MASSFLOW_DROPLETPHASE% , ExpressionOfIntegrand )
```

It computes the flux of a functional f (ExpressionOfIntegrand) across all inflow surfaces in the sense:

$$I_{\text{Flow}} = \int_{\partial\Omega} f dA$$

This integral is approximated by summing up the DROPLETPHASE points which are currently injected at all inflow surfaces $\partial\Omega$:

$$I_{\text{Flow}} \approx \sum_{i \in P_{\text{injected}}} f_i \cdot \frac{V_i}{\Delta t}$$

P_{injected} is the set of all DROPLETPHASE points which are injected at $\partial\Omega$ in this time step.

The current volume of a DROPLETPHASE point is determined by:

$$V_i = \frac{\pi}{6} d_i^3,$$

where d_i is the mean diameter of the DROPLETPHASE (see %ind_d30%).

Example: massflow of DROPLETPHASE through all inflows

```
INTEGRATION ( $massflow$ ) = ( %MASSFLOW_DROPLETPHASE% , [Y %ind_r% ] )
```

Note:

- Details on the DROPLETPHASE can be found in the section Solvers , see KindOfProblem and the respective link.
- Skip is not recommended for this type of integration statement.

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %MAXIMUM_BND%

%MAXIMUM_BND%

maximum of a functional based on all MESHFREE boundary points with respect to given boundary elements

Maximum of a given functional f (ExpressionOfIntegrand) with respect to the set P_{BND} of all MESHFREE boundary points with given POSTPROCESS -flags:

$$I_{\text{MaxBND}} = \max_{i \in P_{\text{BND}}} f_i$$

Example:

```
begin_alias{ }
"Alias1" = " ... POSTPROCESS$PostprocessTag1$ ... " # definition of Alias1
"Alias2" = " ... POSTPROCESS$PostprocessTag2$ ... " # definition of Alias2
end_alias
INTEGRATION ( $IntInd$ ) = ( %MAXIMUM_BND% , ExpressionOfIntegrand , $PostprocessTag1$ ,
$PostprocessTag2$ )
```

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %MAXIMUM_FS%

%MAXIMUM_FS%

maximum of a functional based on all MESHFREE free surface points with respect to given material flags

Maximum of a given functional f (ExpressionOfIntegrand) with respect to the set P_{FS} of all MESHFREE free surface points with given material flags:

$$I_{\mathrm{MaxFS}} = \max_{i \in P_{\mathrm{FS}}} f_i$$

Example:

```
begin_alias{ }
"Alias1" = " ... MAT$MaterialTag1$ ... " # definition of Alias1
"Alias2" = " ... MAT$MaterialTag2$ ... " # definition of Alias2
end_alias
INTEGRATION ( $IntInd$ ) = ( %MAXIMUM_FS% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ )
```

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %MAXIMUM_INT%

%MAXIMUM_INT%

maximum of a functional based on all MESHFREE points with respect to given material flags

Maximum of a given functional f (ExpressionOfIntegrand) with respect to the set P of all MESHFREE points with given material flags:

$$I_{\mathrm{Max}} = \max_{i \in P} f_i$$

Example:

```
begin_alias{ }
"Alias1" = " ... MAT$MaterialTag1$ ... " # definition of Alias1
"Alias2" = " ... MAT$MaterialTag2$ ... " # definition of Alias2
end_alias
INTEGRATION ( $IntInd$ ) = ( %MAXIMUM_INT% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ )
```

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %MINIMUM_BND%

%MINIMUM_BND%

minimum of a functional based on all MESHFREE boundary points with respect to given boundary elements

Minimum of a given functional f (ExpressionOfIntegrand) with respect to the set P_{BND} of all MESHFREE boundary points with given POSTPROCESS -flags:

$$I_{\mathrm{MinBND}} = \min_{i \in P_{\mathrm{BND}}} f_i$$

Example:

```
begin_alias{ }
"Alias1" = " ... POSTPROCESS$PostprocessTag1$ ... " # definition of Alias1
"Alias2" = " ... POSTPROCESS$PostprocessTag2$ ... " # definition of Alias2
end_alias
INTEGRATION ( $IntInd$ ) = ( %MINIMUM_BND% , ExpressionOfIntegrand , $PostprocessTag1$ , $PostprocessTag2$
)
```

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %MINIMUM_FS%

%MINIMUM_FS%

minimum of a functional based on all MESHFREE free surface points with respect to given material flags

Minimum of a given functional f (ExpressionOfIntegrand) with respect to the set P_{FS} of all MESHFREE free surface points with given material flags:

$$I_{\mathrm{MinFS}} = \min_{i \in P_{\mathrm{FS}}} f_i$$

Example:

```
begin_alias{ }
"Alias1" = " ... MAT$MaterialTag1$ ... " # definition of Alias1
"Alias2" = " ... MAT$MaterialTag2$ ... " # definition of Alias2
end_alias
INTEGRATION ( $IntInd$ ) = ( %MINIMUM_FS% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ )
```

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %MINIMUM_INT%

%MINIMUM_INT%

minimum of a functional based on all MESHFREE points with respect to given material flags

Minimum of a given functional f (ExpressionOfIntegrand) with respect to the set P of all MESHFREE points with given material flags:

$$I_{\mathrm{Min}} = \min_{i \in P} f_i$$

Example:

```
begin_alias{ }
"Alias1" = " ... MAT$MaterialTag1$ ... " # definition of Alias1
"Alias2" = " ... MAT$MaterialTag2$ ... " # definition of Alias2
end_alias
INTEGRATION ( $IntInd$ ) = ( %MINIMUM_INT% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ )
```

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %POINT_APPROXIMATE%

%POINT_APPROXIMATE%

approximation of a functional at a BND_point by MESHFREE interpolation

If a BND_point is active throughout the simulation and has a POSTPROCESS -flag, the user can approximate any given function (ExpressionOfIntegrand) at this point by interpolation of MESHFREE points in its neighborhood:

```
INTEGRATION ( $IntInd$ ) = ( %POINT_APPROXIMATE% , ExpressionOfIntegrand , $PostprocessTag$ )
```

If the BND_point has no neighbors, the result is zero.

Example:

```
begin_boundary_elements{ }
BND_point ACTIVE%ACTIVE_always% CHAMBER1 POSTPROCESS $PostprocessTag$ x y z
end_boundary_elements {}

INTEGRATION ( $IntInd$ ) = ( %POINT_APPROXIMATE% , [Y %ind_p% +Y %ind_p_dyn% ], $PostprocessTag$ )
```

See also %POINT_APPROXIMATE_ProjBNDOnly% and %POINT_DIRECT% .

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %POINT_APPROXIMATE_ProjBNDOnly%

%POINT_APPROXIMATE_ProjBNDOnly%

approximation of a functional at a BND_point by MESHFREE interpolation with respect to neighboring boundary points

If a BND_point is active throughout the simulation and has a POSTPROCESS -flag, the user can approximate any given function (ExpressionOfIntegrand) at this point by interpolation of MESHFREE boundary points in its neighborhood:

```
INTEGRATION ( $IntInd$ ) = ( %POINT_APPROXIMATE_ProjBNDOnly% , ExpressionOfIntegrand , $PostprocessTag$ )
```

If the BND_point has no neighbors, the result is zero.

Example:

```
begin_boundary_elements{ }
BND_point ACTIVE%ACTIVE_always% CHAMBER1 POSTPROCESS $PostprocessTag$ x y z
end_boundary_elements {}

INTEGRATION ( $IntInd$ ) = ( %POINT_APPROXIMATE_ProjBNDOnly% , [Y %ind_p% +Y %ind_p_dyn% ],
$PostprocessTag$ )
```

See also %POINT_APPROXIMATE% and %POINT_DIRECT% .

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %POINT_DIRECT%

%POINT_DIRECT%

write simple values like position, chamber index etc. of a BND_point to file

```
INTEGRATION ($IntInd$) = ( %POINT_DIRECT% , ExpressionOfIntegrand , $PostprocessTag1$, $PostprocessTag2$, ...
)
```

Note: The *only* values this integration has access to are

Y %ind_x(1)% , Y %ind_x(2)% , Y %ind_x(3)% , Y %ind_time% , Y %ind_cham% , Y %ind_h% .

For more complicated expressions, including the simulation result, please use %POINT_APPROXIMATE% or %POINT_APPROXIMATE_ProjBNDOnly% .

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %PUBLICVALUE%

%PUBLICVALUE%

public value of MESHFREE simulation

```
INTEGRATION ( $IntInd$ ) = ( %PUBLICVALUE% , Functional )
```

Functional: equation based on public values of a MESHFREE simulation, i.e.
indirect point cloud and boundary element attributes

Example:

```
INTEGRATION ( $pressure$ ) = ( %INTEGRATION_BND_DIRECT% , [Y %ind_p% +Y %ind_p_dyn% ],
$PostprocessTag$ )
INTEGRATION ( $area$ ) = ( %INTEGRATION_BND_DIRECT% , [1.0], $PostprocessTag$ )

INTEGRATION ( $normalized_pressure$ ) = ( %PUBLICVALUE% , [integ( $pressure$ )/integ( $area$ )] )
INTEGRATION ( $allocated_memory$ ) = ( %PUBLICVALUE% , [real( %MEM_STATISTICS_ALLOC% )] )
```

Note: If Functional has different values on the MPI processes, the standard behavior is that the maximum across all processes is used to evalulate the integration statement.

Warning:
Acessing direct point cloud attributes such as Y%ind_...% together with %PUBLICVALUE% means that the attribute of the point with index 1 is taken. Thus, this combination can lead to unexpected results for varying point attributes or empty MPI processes. Only for the following indices, it is explicitly ensured that the correct point-independent variable is used:

- %ind_time% : Current (at the time of evaluating the expression) simulation time
- %ind_dt% : Current (at the time of evaluating the expression) simulation time step

---

%PUBLICVALUE_CLOCKstatistics%

CLOCK value of given stop watch

The time values given by this option refer to the current time cycle.

```
INTEGRATION ( $IntInd$ ) = ( %PUBLICVALUE_CLOCKstatistics% , iArgument, "NameOfStopWatch" )
```

iArgument:

- 1 (average per-point-values of CLOCK time measured by the indicated stop watch)
- 2 (minimum per-point-values of CLOCK time measured by the indicated stop watch: N_MPI*min(CLOCK(1...N_MPI)/N_MFpoints)
- 3 (maximum per-point-values of CLOCK time measured by the indicated stop watch: N_MPI*max(CLOCK(1...N_MPI)/N_MFpoints)
- 4 (sum of the CLOCK-times over all MPI processes)
- 5 (minimum CLOCK-time: N_MPI*min(CLOCK(1...N_MPI))
- 6 (maximum CLOCK-time: N_MPI*max(CLOCK(1...N_MPI))

NameOfStopWatch: see NamesOfStopWatches .

Example:

```
begin_timestepfile{ "TimeStatistics"}
INTEGRATION ( $IntInd1$ ) = ( %PUBLICVALUE% , [real( %RealTimeSimulation% )]) # this puts the time into the first column
INTEGRATION ( $IntInd2$ ) = ( %PUBLICVALUE_CLOCKstatistics% , 1, "ADMIN_TIME_INTEG.ORGANIZE" )
INTEGRATION ( $IntInd3$ ) = ( %PUBLICVALUE_CLOCKstatistics% , 1, "ADMIN_TIME_INTEG.FLIQUID" )
INTEGRATION ( $IntInd4$ ) = ( %PUBLICVALUE_CLOCKstatistics% , 2, "ADMIN_TIME_INTEG.ORGANIZE" )
INTEGRATION ( $IntInd5$ ) = ( %PUBLICVALUE_CLOCKstatistics% , 2, "ADMIN_TIME_INTEG.FLIQUID" )
INTEGRATION ( $IntInd6$ ) = ( %PUBLICVALUE_CLOCKstatistics% , 3, "ADMIN_TIME_INTEG.ORGANIZE" )
INTEGRATION ( $IntInd7$ ) = ( %PUBLICVALUE_CLOCKstatistics% , 3, "ADMIN_TIME_INTEG.FLIQUID" )
end_timestepfile
```


%PUBLICVALUE_CPUstatistics%

CPU value of given stop watch

The time values given by this option refer to the current time cycle.

```
INTEGRATION ( $IntInd$ ) = ( %PUBLICVALUE_CPUstatistics% , iArgument, "NameOfStopWatch" )
```

iArgument:

- 1 (average per-point-values of CPU time measured by the indicated stop watch)
- 2 (minimum per-point-values of CPU time measured by the indicated stop watch: N_MPI*min(CPU(1...N_MPI)/N_MFpoints)
- 3 (maximum per-point-values of CPU time measured by the indicated stop watch: N_MPI*max(CPU(1...N_MPI)/N_MFpoints)
- 4 (sum of the CPU-times over all MPI processes)
- 5 (minimum CPU-time: N_MPI*min(CPU(1...N_MPI))
- 6 (maximum CPU-time: N_MPI*max(CPU(1...N_MPI))

NameOfStopWatch: see NamesOfStopWatches .

Example:

```
begin_timestepfile{ "TimeStatistics"}
INTEGRATION ( $IntInd1$ ) = ( %PUBLICVALUE% , [real( %RealTimeSimulation% )]) # this puts the time into the first column
INTEGRATION ( $IntInd2$ ) = ( %PUBLICVALUE_CPUstatistics% , 1, "ADMIN_TIME_INTEG.ORGANIZE" )
INTEGRATION ( $IntInd3$ ) = ( %PUBLICVALUE_CPUstatistics% , 1, "ADMIN_TIME_INTEG.FLIQUID" )
INTEGRATION ( $IntInd4$ ) = ( %PUBLICVALUE_CPUstatistics% , 2, "ADMIN_TIME_INTEG.ORGANIZE" )
INTEGRATION ( $IntInd5$ ) = ( %PUBLICVALUE_CPUstatistics% , 2, "ADMIN_TIME_INTEG.FLIQUID" )
INTEGRATION ( $IntInd6$ ) = ( %PUBLICVALUE_CPUstatistics% , 3, "ADMIN_TIME_INTEG.ORGANIZE" )
INTEGRATION ( $IntInd7$ ) = ( %PUBLICVALUE_CPUstatistics% , 3, "ADMIN_TIME_INTEG.FLIQUID" )
end_timestepfile
```


%PUBLICVALUE_SUM%

summed public value of MESHFREE simulation

```
INTEGRATION ( $IntInd$ ) = ( %PUBLICVALUE_SUM% , Functional )
```

Functional: equation based on public values of a MESHFREE simulation, i.e.
indirect point cloud and boundary element attributes

This is the time summation of %PUBLICVALUE% :

$$I_{\mathrm{PublicSum}} \approx \sum_{i=\mathrm{AllTimeSteps}} I_{\mathrm{Public},i}$$

Example:

```
INTEGRATION ( $sum_monitor$ ) = ( %PUBLICVALUE_SUM% , [real( %MONITOR_NbParticles% )] )
```

Note: If Functional has different values on the MPI processes, the standard behavior is that the maximum across all processes is used to evalulate the integration statement.

Warning: The same warning as for %PUBLICVALUE% applies here.

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %PUBLICVALUE_TIME%

%PUBLICVALUE_TIME%

time-integrated public value of MESHFREE simulation

```
INTEGRATION ( $IntInd$ ) = ( %PUBLICVALUE_TIME% , Functional )
```

Functional: equation based on public values of a MESHFREE simulation, i.e.
indirect point cloud and boundary element attributes

This is the time integration of %PUBLICVALUE% :

$$I_{\mathrm{PublicTime}} \approx \sum_{i=\mathrm{AllTimeSteps}} I_{\mathrm{Public},i} \cdot \Delta t_i$$

Example:

```
INTEGRATION ( $pressure$ ) = ( %INTEGRATION_BND_DIRECT% , [Y %ind_p% +Y %ind_p_dyn% ],
$PostprocessTag$ )
INTEGRATION ( $time_pressure$ ) = ( %PUBLICVALUE_TIME% , [integ( $pressure$ )] )
INTEGRATION ( $integ_monitor$ ) = ( %PUBLICVALUE_TIME% , [real( %MONITOR_NbParticles% )] )
```

Note: If Functional has different values on the MPI processes, the standard behavior is that the maximum across all processes is used to evalulate the integration statement.

Warning: The same warning as for %PUBLICVALUE% applies here.

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · %SUMMATION_BND%

%SUMMATION_BND%

summation of given function values based on all MESHFREE boundary points with respect to given boundary elements

Summation of a given functional f (ExpressionOfIntegrand) with respect to the set P_{BND} of all MESHFREE boundary points with given POSTPROCESS -flags:

$$I_{\mathrm{SumBND}} = \sum_{i \in P_{\mathrm{BND}}} f_i$$

Example:

```
begin_alias{ }
"Alias1" = " ... POSTPROCESS$PostprocessTag1$ ... " # definition of Alias1
"Alias2" = " ... POSTPROCESS$PostprocessTag2$ ... " # definition of Alias2
end_alias
INTEGRATION ( $IntInd$ ) = ( %SUMMATION_BND% , ExpressionOfIntegrand , $PostprocessTag1$ ,
$PostprocessTag2$ )
```


%SUMMATION_INT%

summation of given function values based on all MESHFREE points with respect to given material flags

Summation of a given functional f (ExpressionOfIntegrand) with respect to the set P of all MESHFREE points with given material flags:

$$I_{\mathrm{Sum}} = \sum_{i \in P} f_i$$

Example:

```
begin_alias{ }
"Alias1" = " ... MAT$MaterialTag1$ ... " # definition of Alias1
"Alias2" = " ... MAT$MaterialTag2$ ... " # definition of Alias2
end_alias
INTEGRATION ( $IntInd$ ) = ( %SUMMATION_INT% , ExpressionOfIntegrand , $MaterialTag1$ , $MaterialTag2$ )
```


AppendDataToExistingFiles

append INTEGRATION results to an existing .timestep file

For some purposes, it might be favorable to append the data to an existing file of the same column structure:

```
begin_timestepfile{ "MyFile"} append{ }
INTEGRATION (...) = ( ... )
end_timestepfile
```


ExpressionOfIntegrand

scalar expression to integrate with respect to a given region

The integrand expression is a typical RightHandSideExpression in the scope of USER_common_variables .

HeaderInfoOrComments

HeaderInfoOrComments

add comments for integration

Enhance a classical integration statement by header information by appending the identifier %INTEGRATION_Header% and the header text **"comment"** to the argument list of the integration statement:

```
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_...%, ExpressionOfIntegrand , ..., $PostprocessTag1$ ,
$PostprocessTag2$ , ..., %INTEGRATION_Header% , "comment" )
```

The header text will be written in the appropriate timestep-file. So, if the integration will be written in the $IntInd$-th column of xyz.timestep, then the header information will appear in a file with the name xyz.timestep.header in the $IntInd$-th line.

Example:

```
begin_timestepfile{ "MyFile"}
INTEGRATION ( $time$ ) = ( %PUBLICVALUE% , [real( %RealTimeSimulation% )], %INTEGRATION_Header% ,
"current simulation time" )
INTEGRATION ( $Wkin$ ) = ( %INTEGRATION_INT% , [Y %ind_v(1)% ^2+Y %ind_v(2)% ^2+Y %ind_v(3)% ^2],
$MaterialTag$ , %INTEGRATION_Header% , "kinetic energy" )
INTEGRATION ( $mass$ ) = ( %INTEGRATION_INT% , [Y %ind_r% ], $MaterialTag$ , %INTEGRATION_Header% ,
"total mass" )
INTEGRATION ( $Wint$ ) = ( %INTEGRATION_INT% , [Y %ind_r% *Y %ind_CV% *Y %ind_T% ], $MaterialTag$ ,
%INTEGRATION_Header% , "internal energy" )
end_timestepfile
```

This will create the file "MyFile.timestep.header" with the following contents:

```
current simulation time
kinetic energy
total mass
internal energy
```

Note: This option always needs to be the *last* one in an integration statement.

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · SelectionFeatures

SelectionFeatures

additional options to further select MESHFREE integration points for integration

A regular integration statement is given by:

```
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_INT_...%, ExpressionOfIntegrand , $MaterialTag$ , $MaterialTag2$ , ...
)
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_BND_...%, ExpressionOfIntegrand , $BoundaryTag$ ,
$BoundaryTag2$ , ... )
```

This statement integrates over all MESHFREE points with the material flag $MaterialTag$ or with the boundary flags $BoundaryTag$ with no further selection of integration points.

If a more distinct selection is needed, use either or both of

- SelectBySwitchOffFunctional
- SelectByPercentileBounds

This is the selection order:
1.) first select by the given $MaterialTag$ or $BoundaryTag$
2.) on top of this, select by SelectBySwitchOffFunctional , if invoked
3.) on top of this, select by SelectByPercentileBounds , if invoked

List of members:

SelectBySwitchOffFunctional	further selection MESHFREE integration points by switch-off-functional
SelectByPercentileBounds	further selection of MESHFREE integration points by percentile restrictions

SelectByPercentileBounds

further selection of MESHFREE integration points by percentile restrictions

```
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_INT_...%, ExpressionOfIntegrand , %INTEGRATION_Percentile% , p_MIN, p_MAX, [f_TEST], [f_WEIGHT], $MaterialTag$ )
INTEGRATION ( $IntInd$ ) = ( %INTEGRATION_BND_...%, ExpressionOfIntegrand , %INTEGRATION_Percentile% , p_MIN, p_MAX, [f_TEST], [f_WEIGHT], $BoundaryTag$ )
```

The percentile ideas is as follows.

1. Define the values
$W_{\mathrm{MIN}} = \sum_{i \in N, f_i^{\mathrm{TEST}} < f_{\mathrm{MIN}}} f_i^{\mathrm{WEIGHT}}$ i.e. the collected weights of all points whose test-function-value is smaller than f_{MIN}
$W_{\mathrm{MAX}} = \sum_{i \in N, f_i^{\mathrm{TEST}} < f_{\mathrm{MAX}}} f_i^{\mathrm{WEIGHT}}$ i.e. the collected weights of all points whose test-function-value is smaller than f_{MAX}
$W_{\mathrm{ALL}} = \sum_{i \in N} f_i^{\mathrm{WEIGHT}}$ i.e. the collected weights of all considered points

2. find $f_{\mathrm{MIN}}, f_{\mathrm{MAX}}$ such that
$W_{\mathrm{MIN}} \approx p_{\mathrm{MIN}} \cdot W_{\mathrm{ALL}}$
$W_{\mathrm{MAX}} \approx p_{\mathrm{MAX}} \cdot W_{\mathrm{ALL}}$
3. select all those points for which we have
$f_{\mathrm{MIN}} \leq f_i^{\mathrm{TEST}} \leq f_{\mathrm{MAX}}$

Example: find maximum global index of MESHFREE points with restrictions

```
INTEGRATION (1) = ( %MAXIMUM_INT% , [Y %ind_IN_glob% ], $MAT$ , %INTEGRATION_Header%, "maximum global index" )
INTEGRATION (2) = ( %MAXIMUM_INT% , [Y %ind_IN_glob% ],
%INTEGRATION_Percentile%, 0, 0.90, [Y %ind_IN_glob% ], 1,
$MAT$ ,
%INTEGRATION_Header%, "maximum global index in the 90-percentile-range" )
INTEGRATION (3) = (-%MAXIMUM_INT%, [Y %ind_IN_glob% ], [Y%ind_proc%<2],
%INTEGRATION_Percentile%, 0, 0.90, [Y %ind_IN_glob% ], 1,
$GLASS$ ,
%INTEGRATION_Header%, "maximum global index in the 90-percentile-range restricted to the first two MPI-procs" )
```

SelectBySwitchOffFunctional

further selection MESHFREE integration points by switch-off-functional

```
INTEGRATION ( $IntInd$ ) = ( - %INTEGRATION_INT% , ExpressionOfIntegrand , SelectionFunctionalIntegral , $MaterialTag$ )
```

Rules:

- Put a minus sign (**-**) in front of the %INTEGRATION_...%-identifier.
- SelectionFunctionalIntegral has to be placed at the end of all mathematical integration functionals. If SelectionFunctionalIntegral > 0 for a MESHFREE point, the point will be considered for the integration, otherwise it is ignored.

Warning: This feature does not (yet) apply for %POINT_...%, %INTEGRATION_FLUX...%, and %PUBLICVALUE...%.

Note: If this feature is used for %BE_INTEGRATION_DIRECT% , %BE_INTEGRATION_DIRECT_TIME% , %MINIMUM_BE% or %MAXIMUM_BE%, SelectionFunctionalIntegral is defined and evaluated on the boundary elements and not on the MESHFREE point cloud!

Example:

```
INTEGRATION ( $IntInd1$ ) = ( - %AVERAGE_INT% , ExpressionOfIntegrand , SelectionFunctionalIntegral , $MaterialTag$ )
INTEGRATION ( $IntInd2$ ) = ( - %MINIMUM_BND% , ExpressionOfIntegrand , SelectionFunctionalIntegral , $PostprocessTag$ )
INTEGRATION ( $IntInd3$ ) = ( - %INTEGRATION_BND% , ExpressionOfIntegrand , ExpressionOfIntegrand , ExpressionOfIntegrand , SelectionFunctionalIntegral , $PostprocessTag$ )
```

Note: This is an experimental solution. In the future, the syntax of the selective integration will be improved and made consistent.

List of members:

SelectionFunctionalIntegral	scalar expression to select or switch off specific points for integration

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · SelectionFeatures · SelectBySwitchOffFunctional · SelectionFunctionalIntegral

SelectionFunctionalIntegral

scalar expression to select or switch off specific points for integration

The selection functional is a typical RightHandSideExpression in the scope of USER_common_variables . If SelectionFunctionalIntegral > 0 for an MESHFREE point, the point will be considered for the integration, otherwise it is ignored.

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · SequentialFiltering

SequentialFiltering

generate writeout to timestep files due to simple sequential filters

In many cases, the user wishes to reduce the data produced by the .timestep files. If INTEGRATION data are explicitly written to a dedicated .timestep-file by the begin_timestepfile{ - clause, then one can define time filters.

Example: additinal sequential filtering

```
begin_timestepfile{ "TimeStatistics"} filter{ %INTEGRATION_FilterBy...%, filterThreshold }
INTEGRATION ( $IntInd1$ ) = ( %PUBLICVALUE% , [real( %RealTimeSimulation% )]) # this puts the time into the first column
INTEGRATION ( $IntInd2$ ) = ( %PUBLICVALUE_CLOCKstatistics% , 1, "ADMIN_TIME_INTEG.ORGANIZE" )
INTEGRATION ( $IntInd3$ ) = ( %PUBLICVALUE_CLOCKstatistics% , 1, "ADMIN_TIME_INTEG.FLIQUID" )
end_timestepfile
```

Over a number of time cycles, the filtered integration results are averaged by the following way:

$$u_{average}^{integ} = \frac{1}{2}\frac{\sum_{i\in N}(u_i^{integ} + u_{i-1}^{integ})(t_i - t_{i-1})}{\sum_{i\in N}(t_i - t_{i-1})}$$

that means it is a weighted average with the time step size to be the weight. In this way, we can guarantee conservation properties of some variables like momentum etc.

List of members:

%INTEGRATION_FilterByTime%	trigger the writeouts time .timestep files based on intervals of simulation time
%INTEGRATION_FilterByTimestepCounter%	trigger the writeouts time .timestep files based on intervals of number of time steps executed

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · SequentialFiltering · %INTEGRATION_FilterByTime%

%INTEGRATION_FilterByTime%

trigger the writeouts time .timestep files based on intervals of simulation time

Example: filtering by simulation time passed

```
begin_timestepfile{ "TimeStatistics"} filter{ %INTEGRATION_FilterByTime% , timeInterval }
INTEGRATION ( $IntInd1$ ) = ( %PUBLICVALUE% , [real( %RealTimeSimulation% )]) # this puts the time into the first column
INTEGRATION ( $IntInd2$ ) = ( %PUBLICVALUE_CLOCKstatistics% , 1, "ADMIN_TIME_INTEG.ORGANIZE" )
INTEGRATION ( $IntInd3$ ) = ( %PUBLICVALUE_CLOCKstatistics% , 1, "ADMIN_TIME_INTEG.FLIQUID" )
end_timestepfile
```

here we force MESHFREE to write out the INTEGRATION results in time intervals of the given value timeInterval .

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · SequentialFiltering · %INTEGRATION_FilterByTimestepCounter%

%INTEGRATION_FilterByTimestepCounter%

trigger the writeouts time .timestep files based on intervals of number of time steps executed

Example: filtering by time steps executed

```
begin_timestepfile{ "TimeStatistics"} filter{ %INTEGRATION_FilterByTimestepCounter% , stepInterval }
INTEGRATION ( $IntInd1$ ) = ( %PUBLICVALUE% , [real( %RealTimeSimulation% )]) # this puts the time into the first column
INTEGRATION ( $IntInd2$ ) = ( %PUBLICVALUE_CLOCKstatistics% , 1, "ADMIN_TIME_INTEG.ORGANIZE" )
INTEGRATION ( $IntInd3$ ) = ( %PUBLICVALUE_CLOCKstatistics% , 1, "ADMIN_TIME_INTEG.FLIQUID" )
end_timestepfile
```

here we force MESHFREE to write out the INTEGRATION results always after a number of stepInterval time cycles has passed.

Skip

additional options to skip computation of integrations

If the integration results are not required in every time step, they can be skipped for a number of time steps or after a certain interval of the simulation time has passed to save computation time. The last computed value is written to file whenever the computation is skipped.

```
INTEGRATION ( $IntInd1$ ) = ( Type, ExpressionOfIntegrand , MaterialOrBoundaryTags,
%INTEGRATION_SkipByTimestepCounter% , TimeStepThreshold )
INTEGRATION ( $IntInd2$ ) = ( Type, ExpressionOfIntegrand , MaterialOrBoundaryTags,
%INTEGRATION_SkipByTime% , TimeThreshold )
```

Example:

```
INTEGRATION ( $volume$ ) = ( %INTEGRATION_INT% , [1.0], $MaterialTag$ ,
%INTEGRATION_SkipByTimestepCounter% , 5 )
INTEGRATION ( $freesurface$ ) = ( %INTEGRATION_FS% , [1.0], $MaterialTag$ , %INTEGRATION_SkipByTime% ,
0.025 )
```

Warning: Skipping is highly discouraged for flux or massflow computations (%...FLUX...%, %..._DROPLETPHASE%) and any types with %..._TIME% and %..._SUM% if the integrand is highly variable in time.

Note: For %..._TIME% and %..._SUM%, the newly computed value is multiplied with the interval just passed. If the integrand is computed in a separate integration statement, with skip, but then integrated over time with %PUBLICVALUE_TIME% or %PUBLICVALUE_SUM% , this uses the old computed value over the interval and thus may lead to a slightly different value.

List of members:

%INTEGRATION_SkipByTime%	skip computation of integrations for a given time interval
%INTEGRATION_SkipByTimestepCounter%	skip computation of integrations for a number of timesteps

DOWNLOAD COMPREHENSIVE EXAMPLE

%INTEGRATION_SkipByTime%

skip computation of integrations for a given time interval

See Skip .

%INTEGRATION_SkipByTimestepCounter%

skip computation of integrations for a number of timesteps

See Skip .

MESHFREE · InputFiles · USER_common_variables · INTEGRATION · TimestepFile

TimestepFile

Results of INTEGRATION statements per timestep

MESHFREE stores the result of the INTEGRATION statements in so-called timestep files. These are pure ASCII files with the ending .timestep in the result folder and they contain the INTEGRATION evaluation (column) for each timestep (rows).

Default Timestep File

The default timestep file is always created (unless suppressed by certain choices of SAVE_type) and contains at least two columns at the beginning: the simulation time of the timestep and the timestep size of the timestep. All INTEGRATION statements not defined within a begin_timestepfile{ environment (see below) will be evaluated into the default timestep file.

Additional Timestep Files

Additional timestep files contain precisely the columns that the user defines within the environment enclosed by begin_timestepfile{ and end_timestepfile .
Example: In order to have the simulation time in the first column it can be specified by:

```
begin_timestepfile{ "myOwnTimestepFile"}
INTEGRATION = ( %PUBLICVALUE% , [real( %RealTimeSimulation% )], %INTEGRATION_Header%, "Simulation Time")
INTEGRATION = ...
...
end_timestepfile
```

For more examples, see the links at begin_timestepfile{ .

Header Files

It is good practice to declare an %INTEGRATION_Header% for all INTEGRATION statements. These headers are found in the corresponding file with the ending .timestep.header, see HeaderInfoOrComments .

MESHFREE · InputFiles · USER_common_variables · KindOfProblem

3.1.16. KindOfProblem

Solver Selection for a simulation chamber

For each simulation CHAMBER the KindOfProblem (or KOP) selects the numerical solver to be used for numerical integration.
All parameters need to be specified per chamber (i.e. per flow phase).

The general form of the statement is

```
KOP(iChamber) = Solvers IntegrationType TimeIntegration MotionOfPointcloud TurbulenceModel
```

Example:

```
KOP(1) = LIQUID LAGRANGE IMPLICIT v-- TURBULENCE:k-epsilon
```

This selects:

- the **LIQUID solver**
- applies **Lagrangian** movement of the point cloud and
- solves the equations **implicitely** using the segregated **v--** solver.
- Additionally, the **k-epsilon turbulence model** is turned on, see KepsilonAlgorithm .

Except for the turbulence these are the default parameters which will be assumed if one parameter is not specified. The

order of the parameters is not relevant.

List of members:

Solvers	Select the solver base on a physical model
MotionOfPointcloud	Movement of point-cloud
TimeIntegration	Order of time integration
IntegrationType	Numerical Scheme used for time integration
TurbulenceModel	Selection of turbulence model

IntegrationType

Numerical Scheme used for time integration

Defines the Scheme to be used for solving the system of equations for velocity and pressure.

Note:
This only applies to LIQUID .

Available Schemes:

- v--

  Segregated solver for incompressible flow.
- vp-

  Coupled implicit solver with penalty formulation for incompressible flow.

MotionOfPointcloud

Movement of point-cloud

This selects the point of view for the description of the flow equations. At the same time this also describes if a fixed point-cloud (EULER) or a moving point-cloud (LAGRANGE) is used.

List of members:

LAGRANGE	Lagrangian motion
EULER	implicit Eulerian or ALE motion (1st order)
EULERIMPL	Higher order implicit Eulerian or ALE motion (recommended among the Euler implementations)

EULER

implicit Eulerian or ALE motion (1st order)

Eulerian formulation of flow equations. The point-cloud is fixed except for moving boundaries. All quantities are transported numerically from one point to the other.

EULERIMPL

Higher order implicit Eulerian or ALE motion (recommended among the Euler implementations)

Transport terms are approximated with a second order accurate scheme. It uses upwinding in combination with MUSCL-reconstruction schemes to
prevent high numerical diffusion and uses an implicit time integration scheme of second order.
It is the Singly Diagonally Implicit Runge-Kutta(SDIRK) method of Alexander. We use the abbreviation SDIRK2 .
Example:

```
KOP(1) = LIQUID EULERIMPL T:EXPIMP(1.0) V:IMPLICIT v-- TURBULENCE:k-epsilon
```

ATTENTION: Please note the remark for the velocity boundary condition BC_v !!!
List of parameters:

- LIMITER
- BETA_FOR_LIMITER
- NB_OF_ACCEPTED_REPETITIONS
- SUBSTEPS_IMPL
- SpecialBNDtreatmentEULERIMPL (experimental)
- StencilOrderReductionNearBND_forEULERIMPL (experimental)
- SkipMarkingPointsLayer2 (experimental)
- TOL_T (control of time step size)
- TOL_keps (control of time step size)
- TOL_v (control of time step size)
- TRANSPORT_ODE_fct_evaluation (experimental)
- additionalPoint_approximation (experimental)
- pure_TRANSPORT (experimental)
- time_integration_impl
- time_integration_impl_solve_v

LAGRANGE

Lagrangian motion

Lagrangian equations with moving point-cloud. Points move with the
local velocity of the flow. Advective properties are carried through this movement.

Solvers

Select the solver base on a physical model

MESHFREE provides a set of different physical models (see Numerics). The LIQUID solver
is the most prominent one used for various kinds of simulations; even for air when it is assumed to be incompressible or weakly-compressible.

Set of keywords:

-

LIQUID

Solver for incompressible and weakly compressible flow.

- GASDYN

  Solver for compressible flow.

- SHALLOWWATER

  Solver for shallow water equations to simulate thin water sheets in 2D.
  Usually coupled to a LIQUID phase.

- POPBAL

  Population balance equations. Bubbles of a secondary phase are represented as local stochastic distribution of droplet sizes.

- DROPLETPHASE

  Explicit solver for droplets which may interact and collect as water films along boundaries

TimeIntegration

Order of time integration

Choose the order of time integration for the temporal discretization. Default behavior is implicit time integration. However, it is possible to switch to an explicit or semi-implicit (a.k.a. implicit-excplicit) scheme for velocity and temperature separately.

Note:
This only applies to LIQUID .

Available orders of time integration:

- IMPLICIT

  Implicit time integration.

- V:EXPLICIT

  Use fully explicit time integration for the velocity.

- V:EXPIMP(0.5)

  Mixed integration scheme for velocities. Any parameter value between 0 and 1 is allowed, where 0 is fully explicit and 1 is fully implicit.

- T:EXPLICIT

  Use fully explicit time integration for temperature.

- T:EXPIMP(0.5)

  Mixed integration scheme for temperature. Any parameter value between 0 and 1 is allowed, where 0 is fully ecplicit and 1 is fully implicit.

- T:NONE

  Turn off solving of temperature equations.

TurbulenceModel

Selection of turbulence model

Select turbulence model to turn it on. Or do not provide any turbulence model to turn it off. So far, only the k-epsilon model for turbulence is supported.

Supported turbulence models:

- [empty]

  Do not provide any turbulence keyword to turn turbulence off.
- TURBULENCE:k-epsilon

  Use the k-epsilon model for turbulence, see KepsilonAlgorithm .

3.1.17. Loops

loop over a block of lines in the input file

Use (nested loops) in the input file.

```
begin_loop{ "LoopVariable_i", iBegin, iStep, iEnd}
line that might contain &LoopVariable_i&
begin_loop{ "LoopVariable_j", jBegin, jStep, jEnd}
line that might contain &LoopVariable_i& and &LoopVariable_j&
begin_loop{ "LoopVariable_k", kBegin, kStep, kEnd}
line that might contain &LoopVariable_i& and &LoopVariable_j& and &LoopVariable_k&
end_loop
end_loop
end_loop
```

The names **&LoopVariable_i&** , **&LoopVariable_j&** , and **&LoopVariable_k&** are free to be chosen by the user.
The values **iBegin** , **iStep** , **iEnd** , etc have to be integers.

Example: Place a raster of cubes in the geometry.

```
begin_boundary_elements{ }

begin_loop{ "iLoop",1,1,18}
begin_loop{ "jLoop",-2,1,2}
BND_cube &AliasForTheCubes& -1 -1 -1 1 1 1 rotate{ 0,0,0,[3*rand(1)],[3*rand(1)],[3*rand(1)]} scale{ &H_min& } offset{ [
&iLoop& *2* &H_min& ],[ &jLoop& *2* &H_min& ],[0.6]}
end_loop
end_loop
end_boundary_elements
```

The cubes are randomly rotated and given a regular offset.

3.1.18. MEMORIZE

memorize functionality

This functionality consists of writing memorize information and, in a subsequent simulation run, reading the saved memorize information. Memorize information can only be generated for the point cloud (see MEMORIZE_Write). With the help of a corresponding MEMORIZE_Read statement, the saved information can be read from the MEMORIZE_File and the MEMORIZE_Header for different cycling modes.

A representative scenario: *fill water from a bottle into different glass shapes and study the different splashing behavior.* Regradless of the shape of the glass, the water always comes out of the bottle in the same way (assuming perfect, repeatable conditions on the way how the bottle is inclined in order to empty out). So, the way to run variations of the glass geometry would be:

- do ONE simulation of the emptying process of the bottle,
- MEMORIZE the MESHFREE points at a defined reference plane/surface below the bottle (i.e. record the time sequence of points going through the reference surface),
- in several SUBSEQUENT simulations (with varying glass geometries), ignore the bottle, instead read in the memorized data such that they practically act as an inflow,
- in this way, save computation time on repeating numerics for any geometrical/parametrical variation.

In order to retain the results of the first simulation including the results of the MEMORIZE_Write statements, the results folder is changed to "MySavePath___MEMORIZERead" for subsequent simulation runs using MEMORIZE_Read statements. The saved memorize information (MEMORIZE_File and MEMORIZE_Header) is copied to this folder.

Note:

- Using both MEMORIZE_Write and MEMORIZE_Read statements in a simulation based on a previous MEMORIZE_Write statement is only possible if they have different indices. In case the indices are identical, the MEMORIZE_Write statement with this identical index is ignored.
- A valid MEMORIZE_Read simulation run with index-differing MEMORIZE_Write statement can not automatically be used in a "third" simulation run with a new MEMORIZE_Read statement. For this to work, the SAVE_path in USER_common_variables has to be adapted accordingly to "MySavePath___MEMORIZERead". With this, a series of simulations using writing-subsequent-reading of information is realizable.
- In case of RESTART , reload (see ComputationalSteering), or resuming (see checkpoint), the current memorize configuration is compared to the previous one. If it does not agree in the necessary characteristics, the simulation is aborted!
- Does currently not work in combination with the begin_save{ -environment. Please use only the standard saving definitions, see SAVE .
- In case of an active MEMORIZE_Read statement, the restart_path is automatically set to "MySavePath___MEMORIZERead" disregarding the definition in USER_common_variables .

List of members:

MEMORIZE_Read	MEMORIZE_Read statements defined for the memorize files and headers
MEMORIZE_Write	MEMORIZE_Write statements defined for the point cloud
MEMORIZE_File	memorize file
MEMORIZE_Header	memorize header file

MEMORIZE_File

memorize file

Writing of memorize files is triggered by MEMORIZE_Write statements. Each statement generates a MEMORIZE_File and the corresponding MEMORIZE_Header with the following naming convention: "MyFileName.memorize_n.dat" and "MyFileName.memorize_n.header", where **n** is the reference number of the MEMORIZE_Write statement.

The MEMORIZE_File is a human readable ascii file. The information defined by the corresponding MEMORIZE_Write statement is saved line by line for each point that was triggered. In the first column the time is saved automatically, in the subsequent columns the additional values for the defined indices are saved.

Example:

```
MEMORIZE_Write (1) = ( equn{ $memorize_trigger$ }, %MEMORIZE_DeletePoint% , %ind_x(1)% , [Y
%ind_x_displaced(1)% ], %ind_x(2)% , [Y %ind_x_displaced(2)% ], %ind_x(3)% , [Y %ind_x_displaced(3)% ] )

begin_equation{ $memorize_trigger$ }
if ( Y %ind_x(1)% < -0.005 ) :: 1.0
else :: 0.0
endif
end_equation
```

This generates a MEMORIZE_File of the following form.

```
0.512202E-03, -0.505818E-02, 0.000000E+00, -0.663885E-03
0.513439E-03, -0.502678E-02, 0.000000E+00, -0.877450E-03
0.513439E-03, -0.500344E-02, 0.000000E+00, 0.237234E-03
0.514677E-03, -0.513243E-02, 0.000000E+00, -0.610120E-03
0.514677E-03, -0.510678E-02, 0.000000E+00, -0.807786E-03
0.514677E-03, -0.504446E-02, 0.640051E-04, -0.418219E-03
0.514677E-03, -0.502776E-02, 0.000000E+00, -0.968895E-04
0.515915E-03, -0.505785E-02, 0.000000E+00, -0.102377E-02
0.515915E-03, -0.509953E-02, 0.000000E+00, -0.516220E-03
0.515915E-03, -0.505135E-02, 0.589291E-04, -0.704640E-03
0.515915E-03, -0.505183E-02, 0.000000E+00, 0.558770E-03
0.515915E-03, -0.511330E-02, 0.000000E+00, 0.180125E-03
..., ..., ..., ...
```

The corresponding MEMORIZE_Header reads as follows (integers may vary for different MESHFREE versions!).

```
32
6
7
8
```

Reading of previously generated memorize files is triggered by MEMORIZE_Read statements.

MEMORIZE_Header

memorize header file

Writing of memorize header files is triggered by MEMORIZE_Write statements. Each statement generates a MEMORIZE_File and the corresponding MEMORIZE_Header with the following naming convention: "MyFileName.memorize_n.dat" and "MyFileName.memorize_n.header", where **n** is the reference number of the MEMORIZE_Write statement.

The MEMORIZE_Header is a human readable ascii file. The point cloud indices defined by the corresponding MEMORIZE_Write statement including the time index are saved line by line.

Example:

```
MEMORIZE_Write (1) = ( equn{ $memorize_trigger$ }, %MEMORIZE_DeletePoint% , %ind_x(1)% , [Y
%ind_x_displaced(1)% ], %ind_x(2)% , [Y %ind_x_displaced(2)% ], %ind_x(3)% , [Y %ind_x_displaced(3)% ] )

begin_equation{ $memorize_trigger$ }
if ( Y %ind_x(1)% < -0.005 ) :: 1.0
else :: 0.0
endif
end_equation
```

This generates a MEMORIZE_File of the following form.

```
0.512202E-03, -0.505818E-02, 0.000000E+00, -0.663885E-03
0.513439E-03, -0.502678E-02, 0.000000E+00, -0.877450E-03
0.513439E-03, -0.500344E-02, 0.000000E+00, 0.237234E-03
0.514677E-03, -0.513243E-02, 0.000000E+00, -0.610120E-03
0.514677E-03, -0.510678E-02, 0.000000E+00, -0.807786E-03
0.514677E-03, -0.504446E-02, 0.640051E-04, -0.418219E-03
0.514677E-03, -0.502776E-02, 0.000000E+00, -0.968895E-04
0.515915E-03, -0.505785E-02, 0.000000E+00, -0.102377E-02
0.515915E-03, -0.509953E-02, 0.000000E+00, -0.516220E-03
0.515915E-03, -0.505135E-02, 0.589291E-04, -0.704640E-03
0.515915E-03, -0.505183E-02, 0.000000E+00, 0.558770E-03
0.515915E-03, -0.511330E-02, 0.000000E+00, 0.180125E-03
..., ..., ..., ...
```

The corresponding MEMORIZE_Header reads as follows (integers may vary for different MESHFREE versions!). The integers represent the MESHFREE internal integers of the Indices %ind...% stated in the corresponding MEMORIZE_Write statement. The time is automatically written in the first column of the MEMORIZE_File and, thus, the corresponding integer in the first line of the MEMORIZE_Header . Decoding of the integers can either be performed by manually counting in the MEMORIZE_Write statement or by comparing with the information in the file "List_of_indices.log" in the hidden folder ".FPM_log___FPM_ID=*".

```
32
6
7
8
```

Reading of previously generated memorize header files is triggered by MEMORIZE_Read statements.

MEMORIZE_Read

MEMORIZE_Read statements defined for the memorize files and headers

In USER_common_variables , the definition of a statement looks as follows:

```
MEMORIZE_Read (n) = ( %MEMORIZE_Cycle% , m_cycle, t_cycle, %MEMORIZE_AdditionalFunctionManipulation% ,
OPTIONAL: %ind_xyz%, expression_xyz [, %ind_abc%, expression_abc ...] )
```

The MEMORIZE_File and MEMORIZE_Header with reference number **n** are read line by line in each time step. If the check time is inside the allowed time frame, the corresponding line in the MEMORIZE_File generates a new MESHFREE point with the saved values. Thereby, the check time is defined by the MEMORIZE_Cycle configuration, i.e. **m_cycle** and **t_cycle** .

m_cycle defines the number of cycles for reading the memorize information: 0 - infinite cycles, 1 - only one cycle, 2 - two cycles, ...

t_cycle defines the cycling time, i.e. which time has to be added to the saved time in case of multiple cycles for reading the memorize information. Its value has to be larger than 0.

After generation of a new point according to the saved memorize information, the %MEMORIZE_AdditionalFunctionManipulation% definitions are evaluated. If present, the given expressions (**expression_xyz** , **expression_abc** , ...) are saved for the indices (**%ind_xyz%** , **%ind_abc%** , ...). Equations are used to define the expressions.

List of members:

%MEMORIZE_Cycle%	cycle configuration MEMORIZE_Read handle
%MEMORIZE_AdditionalFunctionManipulation%	additional function manipulation MEMORIZE_Read handle

%MEMORIZE_AdditionalFunctionManipulation%

additional function manipulation MEMORIZE_Read handle

```
MEMORIZE_Read (n) = ( %MEMORIZE_Cycle% , m_cycle, t_cycle, %MEMORIZE_AdditionalFunctionManipulation% , OPTIONAL: %ind_xyz%, expression_xyz [, %ind_abc%, expression_abc ...] )
```

After generation of a new point according to the saved memorize information, the %MEMORIZE_AdditionalFunctionManipulation% definitions are evaluated. If present, the given expressions (**expression_xyz** , **expression_abc** , ...) are saved for the indices (**%ind_xyz%** , **%ind_abc%** , ...). Equations are used to define the expressions.

%MEMORIZE_Cycle%

cycle configuration MEMORIZE_Read handle

```
MEMORIZE_Read (n) = ( %MEMORIZE_Cycle% , m_cycle, t_cycle, %MEMORIZE_AdditionalFunctionManipulation% , OPTIONAL: %ind_xyz%, expression_xyz [, %ind_abc%, expression_abc ...] )
```

m_cycle defines the number of cycles for reading the memorize information:

- 0 - infinite cycles
- 1 - only one cycle
- 2 - two cycles
- ...

t_cycle defines the cycling time, i.e. which time has to be added to the saved time in case of multiple cycles for reading the memorize information. Its value has to be larger than 0.

m_cycle and t_cycle define the check time.

MEMORIZE_Write

MEMORIZE_Write statements defined for the point cloud

Types of statements are:

1.) Deletion of points
2.) Retention of points

In USER_common_variables , the definition of a statement looks as follows:

```
MEMORIZE_Write (n) = ( memorize_trigger_expression, %MEMORIZE_DeletePoint% , OPTIONAL: %ind_xyz%, expression_xyz [, %ind_abc%, expression_abc ...] )
MEMORIZE_Write (n) = ( memorize_trigger_expression, %MEMORIZE_KeepPoint% , OPTIONAL: %ind_xyz%, expression_xyz [, %ind_abc%, expression_abc ...] )
```

For each MESHFREE point, the **memorize_trigger_expression** is evaluated. If it is larger than zero for the considered point, the statement is triggered. In this case, the given expressions (**expression_xyz** , **expression_abc** , ...) are saved with reference to the indices (**%ind_xyz%** , **%ind_abc%** , ...) in the MEMORIZE_File and MEMORIZE_Header with

reference number **n** , respectively. The current time is saved automatically as first index. Equations are used to define the memorize_trigger_expression as well as the expressions for saving the indices.

List of members:

%MEMORIZE_DeletePoint%	deletion of point MEMORIZE_Write handle
%MEMORIZE_KeepPoint%	retention of point MEMORIZE_Write handle

MESHFREE · InputFiles · USER_common_variables · MEMORIZE · MEMORIZE_Write · %MEMORIZE_DeletePoint%

%MEMORIZE_DeletePoint%

deletion of point MEMORIZE_Write handle

```
MEMORIZE_Write (n) = ( memorize_trigger_expression, %MEMORIZE_DeletePoint% , OPTIONAL: %ind_xyz%, expression_xyz [, %ind_abc%, expression_abc ...] )
```

For each MESHFREE point, the **memorize_trigger_expression** is evaluated. If it is larger than zero for the considered point, the statement is triggered, the specified information of the point is saved wrt the given indices in the MEMORIZE_File as well as MEMORIZE_Header and the point is deleted afterwards. Equations are used to define the memorize_trigger_expression as well as the expressions for saving the indices.

MESHFREE · InputFiles · USER_common_variables · MEMORIZE · MEMORIZE_Write · %MEMORIZE_KeepPoint%

%MEMORIZE_KeepPoint%

retention of point MEMORIZE_Write handle

```
MEMORIZE_Write (n) = ( memorize_trigger_expression, %MEMORIZE_KeepPoint% , OPTIONAL: %ind_xyz%, expression_xyz [, %ind_abc%, expression_abc ...] )
```

For each MESHFREE point, the **memorize_trigger_expression** is evaluated. If it is larger than zero for the considered point, the statement is triggered, the specified information of the point is saved wrt the given indices in the MEMORIZE_File as well as MEMORIZE_Header and the point is retained as it is. Equations are used to define the memorize_trigger_expression as well as the expressions for saving the indices.

MESHFREE · InputFiles · USER_common_variables · MONITORPOINTS

3.1.19. MONITORPOINTS

monitor points due to user-defined conditions

Pure postprocessing points can be created by user-defined conditions in order to better understand the computed flow. These monitorpoints do not take part in the numerics of the simulation, they are simply attached to the solution and carry useful results.

Monitor points can be created by MONITORPOINTS_CREATION , stopped by MONITORPOINTS_STOP , and deleted by MONITORPOINTS_DELETION . The latter is important regarding the performance of a simulation.

Information of monitor points can be saved by SAVE_MONITOR_ITEM .

Information of monitor points, that have been created at boundary elements by %MONITORPOINTS_CREATION_AtBoundary% , can be mapped onto the corresponding boundary elements by

BE_MONITOR_ITEM or directly mapped and saved by SAVE_BE_MONITOR_ITEM .

List of members:	
BE_MONITOR_ITEM	BE monitor item
MONITORPOINTS_CREATION	create monitor points due to user-defined conditions
MONITORPOINTS_DELETION	delete existing monitor points by user-defined conditions
MONITORPOINTS_STOP	stop existing monitor points by user-defined conditions
SAVE_BE_MONITOR_ITEM	monitor item to be saved per BE element for visualization (MP)
SAVE_MONITOR_ITEM	monitor item to be saved for visualization (MP)

DOWNLOAD COMPREHENSIVE EXAMPLE

MESHFREE · InputFiles · USER_common_variables · MONITORPOINTS · BE_MONITOR_ITEM

BE_MONITOR_ITEM

BE monitor item

The syntax of BE_MONITOR_ITEM is analogous to the one of SAVE_BE_MONITOR_ITEM , just omit the "DescriptionText". It allows to evaluate monitor points per boundary element, but does not save the results. However, it can be referenced by BEmon() in equations.

The syntax is equivalent to SAVE_BE_MONITOR_ITEM .

MESHFREE · InputFiles · USER_common_variables · MONITORPOINTS · MONITORPOINTS_CREATION

MONITORPOINTS_CREATION

create monitor points due to user-defined conditions

Create a monitor point out of an existing MESHFREE point with MAT -flag **$Material$** if a given functional (or a sequence of functionals) is positive.
Each functional is a typical RightHandSideExpression in the scope of USER_common_variables .

```
MONITORPOINTS_CREATION ( $Material$ ) = ( %MONITORPOINTS_CREATION_AtBoundary% , Functional1, OPTIONAL:{%AND%,%OR%}, Functional2, ... )
MONITORPOINTS_CREATION ( $Material$ ) = ( %MONITORPOINTS_CREATION_Inside% , Functional1, OPTIONAL:{%AND%,%OR%}, Functional2, ... )
MONITORPOINTS_CREATION ( $Material$ ) = ( %MONITORPOINTS_CREATION_IrreducibleFPMpoint% , Functional1, OPTIONAL:{%AND%,%OR%}, Functional2, ... )
MONITORPOINTS_CREATION ( $Material$ ) = ( %MONITORPOINTS_CREATION_PenetrationOfBlindAndEmptyBoundary% , $iPostprocessFlag$ , OPTIONAL:{%AND%,%OR%}, Functional2, ... )
```

Each monitor point obtains a unique marker in %ind_MARKER% . Their creation time is reported in %ind_st% .
Furthermore, they inherit the values of the creating MESHFREE points.

Note: By default, the monitor points only store a reduced number of Indices . However, all Indices occurring in defined SAVE_MONITOR_ITEM , BE_MONITOR_ITEM , or SAVE_BE_MONITOR_ITEM are stored additionally.

At creation time, dedicated function values can be provided to the monitor points by

MONITORPOINTS_CREATION_FunctionEvaluation .

If a sequence of functionals is used, the functionals can either be combined by %AND% or by %OR% .
A monitor point is created, if the logical convolution of the functionals is true. A functional is true, if it delivers a positive value.

Monitor points are not supported by the SAVE_formats ASCII and ERFHDF5 .

List of members:

%MONITORPOINTS_CREATION_AtBoundary%	create monitor points at the boundary due to user-defined conditions
%MONITORPOINTS_CREATION_Inside%	create monitor points not attached to a boundary due to user-defined conditions
%MONITORPOINTS_CREATION_IrreducibleFPMpoint%	mark MESHFREE points to be irreducible
%MONITORPOINTS_CREATION_PenetrationOfBlindAndEmptyBoundary%	create monitor points if MESHFREE points penetrate BND_BlindAndEmpty boundary
MONITORPOINTS_CREATION_FunctionEvaluation	provide dedicated function values at creation time to the monitor point

%MONITORPOINTS_CREATION_AtBoundary%

create monitor points at the boundary due to user-defined conditions

Create a monitor point out of an existing MESHFREE point with MAT -flag **$Material$** , if a given functional (or a sequence of functionals) is positive,
and attach it to the corresponding boundary.

```
MONITORPOINTS_CREATION ( $Material$ ) = ( %MONITORPOINTS_CREATION_AtBoundary% , Functional1, OPTIONAL:{%AND%,%OR%}, Functional2, ... )
```

Note: The monitor points are attached to the boundary and will also move with it if the boundary moves.

If a sequence of functionals is used, the functionals can either be combined by %AND% or by %OR% .
A monitor point is created, if the logical convolution of the functionals is true. A functional is true, if it delivers a positive value.

Example: Create a monitor point if a MESHFREE point supercedes a pressure criterion or a temperature criterion.

```
MONITORPOINTS_CREATION ( $Material$ ) = ( %MONITORPOINTS_CREATION_AtBoundary% , [Y %ind_p% +Y %ind_p_dyn% > 10000], %OR%, [Y %ind_T% > 100] )
```


%MONITORPOINTS_CREATION_Inside%

create monitor points not attached to a boundary due to user-defined conditions

Create a monitor point out of an existing MESHFREE point with MAT -flag **$Material$** , if a given functional (or a sequence of functionals) is positive,

but do not attach it to the boundary.

```
MONITORPOINTS_CREATION ( $Material$ ) = ( %MONITORPOINTS_CREATION_Inside% , Functional1, OPTIONAL:{%AND%,%OR%}, Functional2, ... )
```

If a sequence of functionals is used, the functionals can either be combined by %AND% or by %OR% .
A monitor point is created, if the logical convolution of the functionals is true. A functional is true, if it delivers a positive value.

Example: Create a monitor point if a MESHFREE point superceeds a pressure criterion and is an interior point.

```
MONITORPOINTS_CREATION ( $Material$ ) = ( %MONITORPOINTS_CREATION_Inside% , [Y %ind_p% +Y %ind_p_dyn% > 10000], %AND%, [Y%ind_kob%<2] )
```

MESHFREE · InputFiles · USER_common_variables · MONITORPOINTS · MONITORPOINTS_CREATION · %MONITORPOINTS_CREATION_IrreducibleFPMpoint%

%MONITORPOINTS_CREATION_IrreducibleFPMpoint%

mark MESHFREE points to be irreducible

A MESHFREE point with MAT -flag **$Material$** is flagged such that MESHFREE cannot cluster it with another MESHFREE point if the point cloud becomes dense.
However, all other reduction operations are executed such as:

- removal after boundary crossing (see %ind_dtb%)
- removal due to isolation status invoked by COMP_IsolatedParticles_MinNbOfNeigh and COMP_IsolatedParticles_MinNbOfInteriorNeigh

```
MONITORPOINTS_CREATION ( $Material$ ) = ( %MONITORPOINTS_CREATION_IrreducibleFPMpoint% , Functional1, OPTIONAL:{%AND%,%OR%}, Functional2, ... )
```

The MESHFREE point is flagged with a random positive value found in %ind_MARKER% . The point can be unflagged by the MONITORPOINTS_DELETION statement.
The user can define a dedicated flag using the MONITORPOINTS_CREATION_FunctionEvaluation statement.

If a sequence of functionals is used, the functionals can either be combined by %AND% or by %OR% .
A monitor point is created, if the logical convolution of the functionals is true. A functional is true, if it delivers a positive value.

Example: Flag MESHFREE points which were just injected at the inflow at time smaller than 0.1 seconds.

```
MONITORPOINTS_CREATION ( $Material$ ) = ( %MONITORPOINTS_CREATION_IrreducibleFPMpoint% , [Y %ind_OrganizePC(1)% = 6], %AND%, [Y %ind_time% < 0.1] )
```

MESHFREE · InputFiles · USER_common_variables · MONITORPOINTS · MONITORPOINTS_CREATION · %MONITORPOINTS_CREATION_PenetrationOfBlindAndEmptyBoundary%

%MONITORPOINTS_CREATION_PenetrationOfBlindAndEmptyBoundary%

create monitor points if MESHFREE points penetrate BND_BlindAndEmpty boundary

It can be useful to monitor MESHFREE point penetrations through a %BND_BlindAndEmpty% boundary since this provides a nice visualization of the impact locations.
Usually, the user would have to create an %INTEGRATION_FLUX% around this boundary. The current option, however, does not sum up but localize the impact events.

```
MONITORPOINTS_CREATION ( $Material$ ) = (
%MONITORPOINTS_CREATION_PenetrationOfBlindAndEmptyBoundary% , $iPostprocessFlag$ , OPTIONAL:
{%AND%,%OR%}, Functional2, ... )
```

A monitor point is created at the location of penetration of an existing MESHFREE point with MAT -flag **$Material$** through the %BND_BlindAndEmpty%
boundary with POSTPROCESS -flag **$iPostprocessFlag$** . The monitor point is mapped to the penetrated boundary element and further moved with it.

Note: The boundary has to be flagged by IDENT%BND_BlindAndEmpty% and by a POSTPROCESS flag which is referenced in the
MONITORPOINTS_CREATION statement.

Additional functionals can either be added by %AND% or by %OR% . A monitor point is created, if the logical convolution of the functionals
and the POSTPROCESS -flag is true. A functional is true, if it delivers a positive value.

Example: Create a monitor point if a MESHFREE point penetrates the %BND_BlindAndEmpty% boundary with POSTPROCESS -flag **iPP** at time larger than 10 seconds.

```
MONITORPOINTS_CREATION ( $Material$ ) = (
%MONITORPOINTS_CREATION_PenetrationOfBlindAndEmptyBoundary% , $iPP$ , %AND%, [Y %ind_time% > 10.0] )
```


MONITORPOINTS_CREATION_FunctionEvaluation

provide dedicated function values at creation time to the monitor point

At the moment of creation of a monitor point, give values to some predefined indices of the monitor point. This is optional. In general, the values of the mother-MESHFREE point will be inherited to the monitor point.
Syntax:

```
MONITORPOINTS_CREATION_FunctionEvaluation ($Material$) = ( %ind_xyz%, expression [,%ind_abc%, expression2
...] )
```

The indices %ind_abc% and %ind_xyz% are classical MESHFREE -index variables as given in Indices .

MONITORPOINTS_DELETION

delete existing monitor points by user-defined conditions

Delete a monitor point with MAT -flag **$Material$** if a given functional (or a sequence of functionals) is positive.

```
MONITORPOINTS_DELETION ( $Material$ ) = ( Functional1, OPTIONAL:{%AND%,%OR%}, Functional2, ... )
```

If a sequence of functionals is used, the functionals can either be combined by %AND% or by %OR% .
A monitor point is deleted, if the logical convolution of the functionals is true. A functional is true, if it delivers a positive value.

MONITORPOINTS_STOP

stop existing monitor points by user-defined conditions

Stop a monitor point with MAT -flag **$Material$** if a given functional (or a sequence of functionals) is positive.

```
MONITORPOINTS_STOP ( $Material$ ) = ( Functional1, OPTIONAL:{%AND%,%OR%}, Functional2, ... )
```

Note: If a monitor point is stopped, there is yet no way to let it move again.

If a sequence of functionals is used, the functionals can either be combined by %AND% or by %OR% .
A monitor point is created, if the logical convolution of the functionals is true. A functional is true, if it delivers a positive value.

MESHFREE · InputFiles · USER_common_variables · MONITORPOINTS · SAVE_BE_MONITOR_ITEM

SAVE_BE_MONITOR_ITEM

monitor item to be saved per BE element for visualization (MP)

See SAVE_BE_MONITOR_ITEM .

MESHFREE · InputFiles · USER_common_variables · MONITORPOINTS · SAVE_MONITOR_ITEM

SAVE_MONITOR_ITEM

monitor item to be saved for visualization (MP)

See SAVE_MONITOR_ITEM .

MESHFREE · InputFiles · USER_common_variables · MOVE

3.1.20. MOVE

move parts of the boundary by an explicit statement

The movement of parts of the boundary is defined by explicit statements. For details see below.

```
MOVE ( $MOVE_index1$ ) = ( %MOVE_position% , xPosition, yPosition, zPosition )
MOVE ( $MOVE_index2$ ) = ( %MOVE_rotation% , xCenter, yCenter, zCenter, xOmega, yOmega, zOmega )
MOVE ( $MOVE_index3$ ) = ( %MOVE_velocity% , xVelocity, yVelocity, zVelocity )
MOVE ( $MOVE_index4$ ) = ( %MOVE_translation% , xDiff, yDiff, zDiff )
MOVE ( $MOVE_index5$ ) = ( %MOVE_rigid% ,
xCenterInit, yCenterInit, zCenterInit,
Mass,
xxInertia, xyInertia, xzInertia, yxInertia, yyInertia, yzInertia, zxInertia, zyInertia, zzInertia,
xVelocityInit, yVelocityInit, zVelocityInit,
xOmegaInit, yOmegaInit, zOmegaInit,
xForce, yForce, zForce,
xMomentum, yMomentum, zMomentum)
MOVE ( $MOVE_index6$ ) = ( %MOVE_TranslationRotation% , xCenterInit, yCenterInit, zCenterInit,
$MOVE_IndexForCenter$ , xOmega, yOmega, zOmega )
MOVE ( $MOVE_index7$ ) = ( %MOVE_ProjectionOfMovementOfAnotherPart% , $MOVE_RefMove$ )
MOVE ( $MOVE_index8$ ) = ( %MOVE_concat% , $MOVE_index_first$ , $MOVE_index_second$ )
MOVE ( $MOVE_index9$ ) = ( %MOVE_vertuschka% , aExtension, bExtension, omega )
MOVE ( $MOVE_index10$ ) = ( %MOVE_ElasticBeam% , $MOVE_FromWhereToTakeForces$ )
MOVE ( $MOVE_index11$ ) = ( %MOVE_ReducedModel% , PressureTerm )
```

Note:

- The number of MOVE statements is currently limited to 100.
- In many cases, data caching can be invoked by the optional parameter %MOVE_InvokeDataCaching% yielding a huge performance boost.

List of members:

%MOVE_concat%	combine two MOVE-statements
%MOVE_ElasticBeam%	special setting for a beam-like structure that moves like a damped elastic beam
%MOVE_InvokeDataCaching%	Data Caching for Move Statements (optional, but recommended)
%MOVE_position%	movement based on a sequence of positions
%MOVE_ProjectionOfMovementOfAnotherPart%	follow the movement of another geometry part
%MOVE_ReducedModel%	special setting for a reduced model (such as rings, beams, etc)
%MOVE_rigid%	rigid body movement (translation and rotation) due to acting forces of the flow
%MOVE_rotation%	rotation movement
%MOVE_translation%	movement by given translation
%MOVE_TranslationRotation%	movement by given translation and rotation
%MOVE_velocity%	movement by given velocity
%MOVE_vertuschka%	special setting for VERTUSCHKA (specific scientific laboratory test in geomechanics)
RIGIDBODY	rigid body movement (translation and rotation) due to acting forces of the flow

%MOVE_ElasticBeam%

special setting for a beam-like structure that moves like a damped elastic beam

```
MOVE ( $MOVE_index$ ) = ( %MOVE_ElasticBeam% , $MOVE_FromWhereToTakeForces$ )
```

All MESHFREE points which belong to a geometry with MOVE -flag **$MOVE_FromWhereToTakeForces$** will contribute to the force
computation/projection of the elastic beam driving the movement of the geometry with MOVE -flag **$MOVE_index$** .


```
MOVE ( $MOVE_index$ ) = ( %MOVE_ElasticBeam% , $MOVE_FromWhereToTakeForces$ )
```

All MESHFREE points which belong to a geometry with MOVE -flag **$MOVE_FromWhereToTakeForces$** will contribute to the force
computation/projection of the elastic beam driving the movement of the geometry with MOVE -flag **$MOVE_index$** .

```
MOVE ( $MOVE_index$ ) = ( %MOVE_ElasticBeam% , $MOVE_FromWhereToTakeForces$ )
```

All MESHFREE points which belong to a geometry with MOVE -flag **$MOVE_FromWhereToTakeForces$** will contribute to the force
computation/projection of the elastic beam driving the movement of the geometry with MOVE -flag **$MOVE_index$** .

%MOVE_InvokeDataCaching%

Data Caching for Move Statements (optional, but recommended)

With the introduction of ORGANIZE_USER_update_boundary_particles_Version = 3 we compute the rotation matrix $\mathbf{M}_{rot}^{n+1}$ and the translation vector $\mathbf{b}_{trans}^{n+1}$ such that the movement from the old to the new position of a geometry node is computed by

$$\mathbf{x}_i^{n+1} = \mathbf{M}_{rot}^{n+1} \cdot \mathbf{x}_i^n + \mathbf{b}_{trans}^{n+1}$$

For any rigid body movement, the translation and rotation items are unique, so the matrix and vector have to be computed only once per timestep for all geometry points.

Thus, it is recommended to also apply the option %MOVE_InvokeDataCaching% to the Move statement in order to avoid unnecessary recomputation of $\mathbf{M}_{rot}^{n+1}$ and $\mathbf{b}_{trans}^{n+1}$. This is possible if the movement is not dependent on space variables and only dependent on time.

Note: MESHFREE does not check for space dependence because it would mean to check every node point of the geometry in every time cycle. This check could be costly depending on the geometry model.

See the definition of the individual MOVE Statements on how to incorporate %MOVE_InvokeDataCaching% .

%MOVE_ProjectionOfMovementOfAnotherPart%

follow the movement of another geometry part

```
MOVE ( $MOVE_index$ ) = ( %MOVE_ProjectionOfMovementOfAnotherPart% , $MOVE_RefMove$ )
```

The movement of the geometry with MOVE -flag **$MOVE_RefMove$** is mapped to the geometry
with MOVE -flag **$MOVE_index$** by a perpendicular projection.

Example:

```
begin_alias{ }
' "A1" = " ... MOVE$MOVE_A1$ ... " ' # definition of alias A1
' "A2" = " ... MOVE$MOVE_A2$ ... " ' # definition of alias A2
end_alias
MOVE ( $MOVE_A1$ ) = ( %MOVE_TranslationRotation% , ... )
MOVE ( $MOVE_A2$ ) = ( %MOVE_ProjectionOfMovementOfAnotherPart% , $MOVE_A1$ )
```

```
MOVE ( $MOVE_index$ ) = ( %MOVE_ProjectionOfMovementOfAnotherPart% , $MOVE_RefMove$ )
```

The movement of the geometry with MOVE -flag **$MOVE_RefMove$** is mapped to the geometry with MOVE -flag **$MOVE_index$** by a perpendicular projection.

Example:

```
begin_alias{ }
' "A1" = " ... MOVE$MOVE_A1$ ... " ' # definition of alias A1
' "A2" = " ... MOVE$MOVE_A2$ ... " ' # definition of alias A2
end_alias
MOVE ( $MOVE_A1$ ) = ( %MOVE_TranslationRotation% , ... )
MOVE ( $MOVE_A2$ ) = ( %MOVE_ProjectionOfMovementOfAnotherPart% , $MOVE_A1$ )
```

```
MOVE ( $MOVE_index$ ) = ( %MOVE_ProjectionOfMovementOfAnotherPart% , $MOVE_RefMove$ )
```

The movement of the geometry with MOVE -flag **$MOVE_RefMove$** is mapped to the geometry with MOVE -flag **$MOVE_index$** by a perpendicular projection.

Example:

```
begin_alias{ }
' "A1" = " ... MOVE$MOVE_A1$ ... " ' # definition of alias A1
' "A2" = " ... MOVE$MOVE_A2$ ... " ' # definition of alias A2
end_alias
MOVE ( $MOVE_A1$ ) = ( %MOVE_TranslationRotation% , ... )
MOVE ( $MOVE_A2$ ) = ( %MOVE_ProjectionOfMovementOfAnotherPart% , $MOVE_A1$ )
```


%MOVE_ReducedModel%

special setting for a reduced model (such as rings, beams, etc)

All MESHFREE points which belong to a geometry with MOVE -flag **$MOVE_index$** will be moved due to the dynamics of the reduced model.
The reduced model needs forces which are interpolated at the nodes of the reduced model geometry.

```
MOVE ( $MOVE_index$ ) = ( %MOVE_ReducedModel% , PressureTerm )
```

PressureTerm: any type of RightHandSideExpression that tells what term to interprete as the pressure to be projected onto the structure of the reduced model

All MESHFREE points which belong to a geometry with MOVE -flag **$MOVE_index$** will be moved due to the dynamics of the reduced model.
The reduced model needs forces which are interpolated at the nodes of the reduced model geometry.

```
MOVE ( $MOVE_index$ ) = ( %MOVE_ReducedModel% , PressureTerm )
```

PressureTerm: any type of RightHandSideExpression that tells what term to interprete as the pressure to be projected onto the structure of the reduced model

All MESHFREE points which belong to a geometry with MOVE -flag **$MOVE_index$** will be moved due to the dynamics of the reduced model.
The reduced model needs forces which are interpolated at the nodes of the reduced model geometry.

```
MOVE ( $MOVE_index$ ) = ( %MOVE_ReducedModel% , PressureTerm )
```

PressureTerm: any type of RightHandSideExpression that tells what term to interprete as the pressure to be projected onto the structure of the reduced model

---

%MOVE_TranslationRotation%

movement by given translation and rotation

```
MOVE ( $MOVE_index$ ) = ( %MOVE_TranslationRotation% , xCenterInit, yCenterInit, zCenterInit, $MOVE_Center$ ,
xOmega, yOmega, zOmega, OPTIONAL:%MOVE_VirtualRotation% , OPTIONAL:%MOVE_InvokeDataCaching% )
MOVE ( $MOVE_Center$ ) = ( %MOVE_...%, AnythingCanBeHere, ... )
```

The vector (**xCenterInit** , **yCenterInit** , **zCenterInit**) represents the initial center of rotation of the geometry with MOVE_Flag **$MOVE_index$** .
This center of rotation is then translated by the movement described in an additional MOVE -statement (here: **$MOVE_Center$**) which is compulsory.

The geometry is translated with the movement defined in $MOVE_Center$. On top of it, a rigid rotation about the current center
of rotation with rotation vector (**xOmega** , **yOmega** , **zOmega**) is applied.

Example: A rolling wheel can be modeled by

```
MOVE ( $MOVE_index$ ) = ( %MOVE_TranslationRotation% , 0, 0, 1, $MOVE_Center$ , 0, 6.2831852, 0 )
MOVE ( $MOVE_Center$ ) = ( %MOVE_velocity% , 6.2831852, 0, 0 )
```

A wheel with radius 1m, the center of which is originally at (0, 0, 1), moves forward in x-direction with a speed of 6.2831852m/s.
The rotation is put accordingly to (0, 6.2831852, 0) such that it turns out to be a rolling movement with 1 rotation per second.

Further options:

1.) In some cases, for instance for deformed tyres, the user does not actually want to rotate the tyre, but only apply the
rotation boundary condition to the flow. If the tyre would rotate, the deformation would rotate as well, such that
it would have to be recomputed in every time cycle.

```
MOVE ( $MOVE_index$ ) = ( %MOVE_TranslationRotation% , 0, 0, 1, $MOVE_Center$ , 0, 6.2831852, 0,
%MOVE_VirtualRotation% )
MOVE ( $MOVE_Center$ ) = ( %MOVE_velocity% , 6.2831852, 0, 0 )
```

%MOVE_VirtualRotation% lets the tyre translate with **$MOVE_Center$** but not rotate. For the velocity boundary conditions, however,
the rotational speed is provided. That especially refers to the following BC_v -conditions:

- %BND_wall_nosl%
- %BND_wall%
- %BND_slip%

2.) For big data sets (geometry) whose movement depends on time only (and not on space) the performance can be improved by data caching.

```
MOVE ( $MOVE_index$ ) = ( %MOVE_TranslationRotation% , 0, 0, 1, $MOVE_Center$ , 0, 6.2831852, 0, ... ,
%MOVE_InvokeDataCaching% )
```

%MOVE_InvokeDataCaching% : Data caching is recommended for performance reasons if the movement is not dependent

on space variables and only dependent on time.

Note: MESHFREE does not check for space dependence because it would mean to check every node point of the geometry
in every time cycle. This check could be costly depending on the geometry model.

%MOVE_concat%

combine two MOVE-statements

```
MOVE ( $MOVE_index_first$ ) = ...
MOVE ( $MOVE_index_second$ ) = ...
MOVE ( $MOVE_combined$ ) = ( %MOVE_concat% , $MOVE_index_first$ , $MOVE_index_second$ )
```

The geometry with MOVE -flag **$MOVE_combined$** moves based on the combination of the movement for MOVE -flag **$MOVE_index_first$** and the one with MOVE -flag **$MOVE_index_second$** . The order of the MOVE -statements does not matter as the computation of displacements and the actual changes in position are decoupled. So, based on the current positions the predicted movements are summed and at the beginning of the next timestep they are executed.

%MOVE_position%

movement based on a sequence of positions

```
MOVE ( $MOVE_index$ ) = ( %MOVE_position% , xPosition, yPosition, zPosition,
OPTIONAL:%MOVE_InvokeDataCaching% )
```

xPosition, yPosition, zPosition: The time-dependent sequences of x-, y-, and z-coordinates
form a sample path along which the whole geometry with MOVE -flag **$MOVE_index$** moves along.

Example 1:

```
MOVE ( $MOVE_index$ ) = ( %MOVE_position% , [sin(Y %ind_time% )], [-cos(Y %ind_time% )], 0 )
```

This forms a movement of the geometry with MOVE -flag $MOVE_index$ on a circular curve.

Example 2:

```
MOVE ( $MOVE_index$ ) = ( %MOVE_position% , curve{ $CURVE_xPos$ }, curve{ $CURVE_yPos$ }, curve{
$CURVE_zPos$ } )
```

The curves implement the time sequence of the x-, y-, and z-component of the movement.

%MOVE_InvokeDataCaching% : Data caching is recommended for performance reasons if the movement is not dependent
on space variables and only dependent on time.

Note: MESHFREE does not check for space dependence because it would mean to check every node point of the geometry
in every time cycle. This check could be costly depending on the geometry model.

%MOVE_rigid%

rigid body movement (translation and rotation) due to acting forces of the flow

See RIGIDBODY .

MESHFREE · InputFiles · USER_common_variables · MOVE · %MOVE_rotation%

%MOVE_rotation%

rotation movement

Rotation of the geometry with MOVE -flag **$MOVE_index$** :

```
MOVE ( $MOVE_index$ ) = ( %MOVE_rotation% , xCenter, yCenter, zCenter, xOmega, yOmega, zOmega,
OPTIONAL:%MOVE_InvokeDataCaching% )
```

(**xCenter** , **yCenter** , **zCenter**) is the rotation center which can be
modeled as typical RightHandSideExpression , i.e. Equations as well as Curves . Its unit is meters.

The rotation vector is given by (**xOmega** , **yOmega** , **zOmega**). Its unit is 1/s (radians per second).
If the magnitude of this vector takes a value of 6.2831852, then one revolution per second is prescribed.

The direction of (**xOmega** , **yOmega** , **zOmega**) represents the rotation axis.

%MOVE_InvokeDataCaching% : Data caching is recommended for performance reasons if the movement is not dependent
on space variables and only dependent on time.

Note: MESHFREE does not check for space dependence because it would mean to check every node point of the geometry
in every time cycle. This check could be costly depending on the geometry model.

MESHFREE · InputFiles · USER_common_variables · MOVE · %MOVE_translation%

%MOVE_translation%

movement by given translation

```
MOVE ( $MOVE_index$ ) = ( %MOVE_translation% , xDiff, yDiff, zDiff, OPTIONAL:%MOVE_InvokeDataCaching% )
```

The vector (**xDiff** , **yDiff** , **zDiff**) represents the current translation and direction of the geometry
with MOVE -flag **$MOVE_index$** . The unit is meters. The components can be modeled as typical RightHandSideExpression , i.e.
Equations as well as Curves .

%MOVE_InvokeDataCaching% : Data caching can be invoked if the given velocity is not dependent on space variables, but only on time.

Note: MESHFREE does not check for space dependence because it would mean to check every node point of the geometry
in every time cycle. This check could be costly depending on the geometry model.

MESHFREE · InputFiles · USER_common_variables · MOVE · %MOVE_velocity%

%MOVE_velocity%

movement by given velocity

```
MOVE ( $MOVE_index$ ) = ( %MOVE_velocity% , xVelocity, yVelocity, zVelocity,
OPTIONAL:%MOVE_InvokeDataCaching% )
```

The vector (**xVelocity** , **yVelocity** , **zVelocity**) represents the current translation speed and direction of the geometry with MOVE -flag **$MOVE_index$** . The unit is m/s. The components can be modeled as typical RightHandSideExpression , i.e.
Equations as well as Curves .

%MOVE_InvokeDataCaching% : Data caching is recommended for performance reasons if the rotation is not dependent on space variables and only dependent on time.

Note: MESHFREE does not check for space dependence because it would mean to check every node point of the geometry
in every time cycle. This check could be costly depending on the geometry model.

%MOVE_vertuschka%

special setting for VERTUSCHKA (specific scientific laboratory test in geomechanics)

The geometry with MOVE -flag $MOVE_index$ is moved according to an ellipsoidal deformation.

```
MOVE ( $MOVE_index$ ) = ( %MOVE_vertuschka% , aExtension, bExtension, omega )
```

Formulation for original ellipsoid:

$$x = a_{\mathrm{Extension}} \cdot e_x \cdot \cos(gamma) + b_{\mathrm{Extension}} \cdot e_y \cdot \sin(gamma)$$

aExtension = $a_{\mathrm{Extension}}$
bExtension = $b_{\mathrm{Extension}}$
omega = rotation speed in 1/s

The geometry with MOVE -flag $MOVE_index$ is moved according to an ellipsoidal deformation.

```
MOVE ( $MOVE_index$ ) = ( %MOVE_vertuschka% , aExtension, bExtension, omega )
```

Formulation for original ellipsoid:

$$x = a_{\mathrm{Extension}} \cdot e_x \cdot \cos(gamma) + b_{\mathrm{Extension}} \cdot e_y \cdot \sin(gamma)$$

aExtension = $a_{\mathrm{Extension}}$
bExtension = $b_{\mathrm{Extension}}$
omega = rotation speed in 1/s

The geometry with MOVE -flag $MOVE_index$ is moved according to an ellipsoidal deformation.

```
MOVE ( $MOVE_index$ ) = ( %MOVE_vertuschka% , aExtension, bExtension, omega )
```

Formulation for original ellipsoid:

$$x = a_{\mathrm{Extension}} \cdot e_x \cdot \cos(gamma) + b_{\mathrm{Extension}} \cdot e_y \cdot \sin(gamma)$$

aExtension = $a_{\text{Extension}}$
bExtension = $b_{\text{Extension}}$
omega = rotation speed in 1/s

RIGIDBODY

rigid body movement (translation and rotation) due to acting forces of the flow

The geometry with MOVE -flag **$MOVE_index$** moves due to the acting forces of the flow as well as additional outer forces and momentum.

In particular, we solve the ODE of movement of rigid rotating bodies:

$$\frac{d}{dt}\left(\mathbf{x}_{COG}\right) = \mathbf{v}_{COG}$$

$$\frac{d}{dt}\left(m \cdot \mathbf{v}_{COG}\right) = F_{fluid} + F_{gravity} + F_{outer} + F_{contact}$$

$$\frac{d}{dt}\left(\mathbf{I} \cdot \omega_{COG}\right) = M_{fluid} + M_{outer} + M_{contact}$$

The variables are

- t : time
- m : mass of the body,
- $\mathbf{x}_{COG}$: position of the center of gravity of the body ; this can be interrogated by the function xCOG() ,
- $\mathbf{v}_{COG}$: velocity of the center of gravity; this can be interrogated by the function vCOG() ,
- F_{fluid} : forces acting from the fluid onto the body (automatically measured and applied!!!), to be requested by the function FCOG() ,
- $F_{gravity}$: the gravity forces deduced from the definition of gravity of the appropriate material ,
- F_{outer} : additional / outer forces other than fluid or gravity / body forces ,
- $\mathbf{I}$: tensor of rotational inertia ,
- ω_{COG} : rotational speed about the center of gravity of the body, to be requested by the function omCOG() ,
- M_{fluid} : moment about the center of gravity (automatically measured and applied!!!!), this can be inquired by the function MCOG() ,
- M_{outer} : outer moments other than the moment applied by the fluid ,
- $F_{contact}, M_{contact}$: if RIGIDBODY_UseCollisionModel = true , then MESHFREE detects the body-body- and body-boundary-intersections and automatically applies contact forces and moments .

Remark : the items above have to be initialized in the MOVE statement (see below)

```
MOVE ( $MOVE_index$ ) = ( %MOVE_rigid% ,
xCenterInit, yCenterInit, zCenterInit,
Mass,
xxInertia, xyInertia, xzInertia, yxInertia, yyInertia, yzInertia, zxInertia, zyInertia, zzInertia,
xVelocityInit, yVelocityInit, zVelocityInit,
xOmegaInit, yOmegaInit, zOmegaInit,
xForce, yForce, zForce,
xMomentum, yMomentum, zMomentum,
OPTIONAL:xxdFduInit, xydFduInit, xzdFduInit, yxdFduInit, yydFduInit, yzdFduInit, zxdFduInit, zydFduInit, zzdFduInit ,
OPTIONAL:xxdGdOmega, xydGdOmega, xzdGdOmega, yxdGdOmega, yydGdOmega, yzdGdOmega, zxdGdOmega,
zydGdOmega, zzdGdOmega )
```

- (**xCenterInit** , **yCenterInit** , **zCenterInit**): initial center of gravity $\mathbf{x}_{COG}$
- **Mass** : mass of rigid body m
- (**xxInertia** , **xyInertia** , **xzInertia** , **yxInertia** , **yyInertia** , **yzInertia** , **zxInertia** , **zyInertia** , **zzInertia**): initial tensor of inertia $\mathbf{I}$
- (**xVelocityInit** , **yVelocityInit** , **zVelocityInit**): initial velocity $\mathbf{v}_{COG}$
- (**xOmegaInit** , **yOmegaInit** , **zOmegaInit**): inital rotational state ω_{COG}

- (**xForce** , **yForce** , **zForce**): outer forces F_{outer}
- (**xMomentum** , **yMomentum** , **zMomentum**): outer momentum M_{outer}
- (xxdFduInit, xydFduInit, xzdFduInit, yxdFduInit, yydFduInit, yzdFduInit, zxdFduInit, zydFduInit, zzdFduInit): initial guess of dF/du (tensor)
- (xxdGdOmega, xydGdOmega, xzdGdOmega, yxdGdOmega, yydGdOmega, yzdGdOmega, zxdGdOmega, zydGdOmega, zzdGdOmega): initial guess of dG/dOmega (tensor)

3.1.21. NumericalControl

numerical control options

See the list of options below.

List of members:

CoeffDtVirt	per MESHFREE point definition of the virtual time step size
ENFORCE_min_max	set lower and upper bound for any MESHFREE variable
ENFORCE_min_max_RejectLinearSolution	rejection of the solution of a sparse linear system if minimum-maximum criteria are not fulfilled

CoeffDtVirt

per MESHFREE point definition of the virtual time step size

Define the parameter A_{virt} in VirtualTimeStepSize per MESHFREE point with CHAMBER -index **iChamber** :

```
CoeffDtVirt (iChamber) = (LocalValue)
```

LocalValue is a RightHandSideExpression .

A previous version implements a constant, chamber-wise definition, see COEFF_dt_virt and VirtualTimeStepSize .
If a CoeffDtVirt definition exists for a MESHFREE point, then the original COEFF_dt_virt is neglected.

Example:

```
CoeffDtVirt (1) = [Y %ind_dt_local% /Y %ind_dt% *0.1]
```


ENFORCE_min_max

set lower and upper bound for any MESHFREE variable

In order to assure some minimum and maximum conditions, the user is able to restrict the solution to any MESHFREE variable by:

```
ENFORCE_min_max ($Material$,%ind_Variable%) = ( minNotToBeUndercut, maxNotToBeExceeded, OPTIONAL:SlopeNotToBeExceeded )
```

MESHFREE simply cuts the solution of the given variable after a time step is completed.

minNotToBeUndercut: MESHFREE cuts the function values in the sense $f_i = \max(f_i, \text{minNotToBeUndercut})$
maxNotToBeExceeded: MESHFREE cuts the function values in the sense $f_i = \min(f_i, \text{maxNotToBeExceeded})$
SlopeNotToBeExceeded: MESHFREE smoothes the function such that $\|\nabla f\|_2 \leq \text{SlopeNotToBeExceeded}$
Equivalent to CODI_min_max .

See also ENFORCE_min_max_RejectLinearSolution .

MESHFREE · InputFiles · USER_common_variables · NumericalControl · ENFORCE_min_max_RejectLinearSolution

ENFORCE_min_max_RejectLinearSolution

rejection of the solution of a sparse linear system if minimum-maximum criteria are not fulfilled

In USER_common_variables , the statement

```
ENFORCE_min_max_RejectLinearSolution ( $MaterialIndex$ , %ind_Entity%) = ( MinimumNotToBeSubceeded, MaximumNotToBeExceeded )
```

leads to a pointwise definition of the accepted minima and maxima of the solution to a sparse linear system.
If the given minima or maxima are exceeded for one or more points, then the whole linear solution is rejected for the current time step.

Warning: Currently, only the pressure entities LIQUID.%ind_p%, %ind_p_dyn% , and %ind_c% (hydrostatic, dynamic, and correction pressures) are supported.

Equivalent to CODI_min_max_RejectLinearSolution .

See also ENFORCE_min_max .

MESHFREE · InputFiles · USER_common_variables · ODE

3.1.22. ODE

solver for ordinary differential equations (ODE)

Let us solve ordinary differential equations (ODE) of the form

$$A\frac{dY}{dt} + BY = Q,$$

where Y is the unknown variable to be integrated and A, B, Q are user given.

The numerical scheme of second order in time, which is used in MESHFREE to solve this type of equations, is Crank-Nicolson-like.

$$\left(A^{n+1} + A^n\right) \cdot \frac{Y^{n+1} - Y^n}{\Delta t} + \left(B^{n+1}Y^{n+1} + B^nY^n\right) = Q^{n+1} + Q^n$$

The resulting equation for the unknown is:

$$Y^{n+1} = \frac{1}{\frac{1}{\Delta t}(A^{n+1} + A^n) + B^{n+1}} \cdot \left(Q^{n+1} + Q^n + \left(\frac{1}{\Delta t}(A^{n+1} + A^n) - B^n\right) \cdot Y^n\right)$$

In USER_common_variables the n-th ODE to be solved is defined by:

```
ODE (n) = ( A, B, Q, Y0 )
```

A , **B** , **Q:** parameters in the model equation which are subject to RightHandSideExpression (Equations , Curves , etc. which also might vary in time)

Y0: initial value of the solution at start time which is also subject to RightHandSideExpression , however, it is only evaluated at the beginning of the simulation.

Note: Currently, the number of ODE is limited to 1000.

The result of the time integration of an ODE can be retrieved by Equations (see ode()) and, therefore, be used in all other functionalities of USER_common_variables .

3.1.23. PhysicalProperties

define physical properties of a material

See the list below.

List of members:	
absolute_pressure	initial pressure [Pa] which is added to get the absolute pressure
cv	specific heat of the material in J/(kg*K)
DarcyBasisVelocity	velocity of porous material [m/s]
DarcyConstant	coupling parameter for porous media [kg/(s*m^3)]
density	material density [kg/m^3]
eta	viscosity definition [Pa*s]
ForchheimerConstant	coupling parameter for porous media [kg/m^4]
gravity	define gravity or body forces of a material [m/s^2]
heatsource	heat source [W/m^3]
lambda	thermal conductivity [W/(m*K)]
mue	shear modulus definition [Pa]
ParticleInteraction	defines the dynamics of particle-particle interaction within the DROPLETPHASE as material property
RedlichKwongGasLaw	more accurate gas law for modeling real gas behavior
sigma	surface tension [N/m]

DarcyBasisVelocity

velocity of porous material [m/s]

Define the reference velocity of the porous material with index **$Material$** :

```
DarcyBasisVelocity( $Material$ ) = RightHandSideExpression
```

The law of Darcy models the influence of a porous medium A on a fluid B that flows through A by the addition of a momentum source term to the standard fluid flow equations of B. See EquationsToSolve for the integration of this source term to the momentum equation and TwoPhaseDarcy for a more specific example of using Darcy within MESHFREE .

The magnitude and direction of this source term is dependent on the relative velocity between A and B. Therefore the DarcyBasisVelocity should be defined as a projection of the velocity of the porous medium to the points of the fluid.

The function **projY()** can be used to project a MESHFREE -entity **%ind_Entity%** from the porous medium to the fluid (and vice versa).

DarcyConstant

*coupling parameter for porous media [kg/(s*m^3)]*

Define the DarcyConstant $\tilde{\beta}$ for the material with index $Material$:

```
DarcyConstant( $Material$ ) = RightHandSideExpression
```

The law of Darcy models the influence of a porous medium A on a fluid B that flows through A by the addition of a momentum source term to the standard fluid flow equations of B. See EquationsToSolve for the integration of this source term to the momentum equation and TwoPhaseDarcy for a more specific example of using Darcy within MESHFREE .

The DarcyConstant regulates the permeability of the porous medium and thus influences the magnitude of this source term.

Isotropic materials

If in the RightHandSideExpression one argument is given, e.g.

```
DarcyConstant( $Material$ ) = ( 1e3 ) # constant Darcy constant of 1e3 kg/(s*m^3)
```

then the porous material is assumed to be isotropic. Thus, β in EquationsToSolve can be viewed as a scalar quantity.

Anisotropic materials

For anisotropic permeability, the DarcyConstant can be set for three perpendicular directions. The RightHandSideExpression then takes twelve arguments, e.g.

```
DarcyConstant( $Material$ ) = ( &bx& , 1, 0, 0, ... # Darcy constant in x-direction, unit vector x
&by& , 0, 1, 0, ... # Darcy constant in y-direction, unit vector y
&bz& , 0, 0, 1 ) # Darcy constant in z-direction, unit vector z
```

In this case β in EquationsToSolve represents a matrix which is constructed from the supplied constants and directions.

Inertial contribution

To extend the Darcy model by an inertial contribution, see ForchheimerConstant .

Notes

- Despite the naming convention, %ind_betaDarcy% will not store $\tilde{\beta}$, but $\beta = \frac{\tilde{\beta}}{\rho}$ in EquationsToSolve

---

ForchheimerConstant

coupling parameter for porous media [kg/m^4]

While the constant defined in DarcyConstant represents the classical Darcy relation for porous media of the form

$$-\nabla p = \tilde{\beta}_D u$$

one may further extend this by an inertial contribution of Forchheimer type by defining the constant $\tilde{\beta}_F$ in

$$-\nabla p = \tilde{\beta}_D u + \tilde{\beta}_F \|u\| u$$

via

```
ForchheimerConstant ( $Material$ ) = RightHandSideExpression
```

In case this constant is defined, β in EquationsToSolve is given by $\beta = \frac{1}{\rho}\left(\tilde{\beta}_D + \tilde{\beta}_F \|\mathbf{v} - \mathbf{v}_\beta\|\right)$.

Isotropic materials

If in the RightHandSideExpression one argument is given, e.g.

```
ForchheimerConstant ( $Material$ ) = ( 1.0 ) # scalar Forchheimer constant of 1.0 1/m
```

then the porous material is assumed to be isotropic. Thus, β in EquationsToSolve can be viewed as a scalar quantity.

Anisotropic materials

If DarcyConstant is specified for three perpendicular directions, three arguments can be supplied to ForchheimerConstant , e.g.

```
ForchheimerConstant ( $Material$ ) = ( &Fx& , &Fy& , &Fz& )
```

Then, the constant &bx&, &by&, &bz& in DarcyConstant are modified in the sense that $bx = bx + Fx\|\mathbf{v} - \mathbf{v}_\beta\|$.

Notes

- The behavior of ForchheimerConstant replicates the behavior of specifying the Forchheimer term via an equation in DarcyConstant which uses %ind_v_0(1:3)% (the main purpose of ForchheimerConstant is thus to simplify inputs)
- In particular, the relative velocity norm within the Forchheimer term is based on $\mathbf{v}^n$ and DarcyBasisVelocity
- Despite the naming convention, %ind_betaDarcy% will not store $\tilde{\beta}_D$, but the above β (in case of a non-zero ForchheimerConstant)
- The case of non-scalar DarcyConstant but scalar ForchheimerConstant will be treated as if 3 identical values (&Fx&=&Fy&=&Fz&) were supplied to ForchheimerConstant .

ParticleInteraction

defines the dynamics of particle-particle interaction within the DROPLETPHASE as material property

Originally, the particles within DROPLETPHASE were not interacting at all. An interaction between particles within a DROPLETPHASE chamber may now be enabled by defining:

```
ParticleInteraction( $Material$ ) = (k_n, e_n, E_a, R_a, mu)
```

The interaction is resolved by a DEM approach which calculates forces on the basis of virtual overlap and relative velocity of the droplets. See DropletCollisions

Parameter	Meaning	Possible Values	Default
k_n	Spring Constant for particle interaction	k_n >= 0.0	0.0 (no collision modeling)
e_n	if 0 <= e_n <= 1 Coefficient of Restitution (0 ideal plastic, 1.0 ideal elastic), if e_n < 0, negative value of the damping coefficient	between 0 and 1 or negative	0.0
E_a	Adhesive potential difference relative to the particle mass	non-negative	0.0 (no adhesion)
R_a	Broadness of zone of attraction relative to d30	non-negative	1.0
mu	Friction Coefficient	non-negative	0.0 (off)

Example:

```
ParticleInteraction( $Material$ ) = (1.0, .1, 1e-3, 1.0, 0.0)
```

specifies that the particles of material $Material$ within a DROPLETPHASE chamber interact with each other. For the collision a spring constant of size 1.0 is specified, a coefficient of restitution of 0.1 means that 90% of the kinetic energy is dissipated by the colliding particles. Additionally, an adhesive potential is given acting within a close range of the particles, attracting each other.

RedlichKwongGasLaw

more accurate gas law for modeling real gas behavior

For the use of the Redlich-Kwong gas law define the PhysicalProperties density, cv, lambda, eta and the initial absolute pressure for the material
with index **$Material$** by using

```
density( $Material$ ) = (%MED_REDLICH_KWONG%, MolarMass, PressureCritical, TemperatureCritical)
cv( $Material$ ) = (%MED_REDLICH_KWONG%, MolarMass, PressureCritical, TemperatureCritical)
lambda( $Material$ ) = (%MED_REDLICH_KWONG%, MolarMass, PressureCritical, TemperatureCritical)
eta( $Material$ ) = (%MED_REDLICH_KWONG%, MolarMass, PressureCritical, TemperatureCritical)
absolute_pressure ( $Material$ ) = InitAbsolutePressure
```

The parameters are:

- MolarMass [g/mol] of the material, e.g. Hydrogen: 2.01588
- PressureCritical [Pa]: pressure from the critical point data of the material, e.g. Hydrogen: 1.3152*10^6
- TemperatureCritical [K]: temperature from the critical point data of the material, e.g. Hydrogen: 33.19

Example:

```
begin_alias{ }
"TCRIT" = "33.19" # [K]
"PCRIT" = "1.3152e6" # [Pa]
"Mw" = "2.01588" # [g/mol]
"p0" = "93.6" # [bar]
end_alias
...
density( $GAS$ ) = (%MED_REDLICH_KWONG%, [ &Mw& ], [ &PCRIT& ], [ &TCRIT& ]) # density in [kg/m³]
cv( $GAS$ ) = (%MED_REDLICH_KWONG%, [ &Mw& ], [ &PCRIT& ], [ &TCRIT& ]) # heat capacity in [Nm/(Kg*K)]
lambda( $GAS$ ) = (%MED_REDLICH_KWONG%, [ &Mw& ], [ &PCRIT& ], [ &TCRIT& ]) # heat conductivity in [W/(mK)]

eta( $GAS$ ) = (%MED_REDLICH_KWONG%, [ &Mw& ], [ &PCRIT& ], [ &TCRIT& ]) # viscosity in [Pa*s]
absolute_pressure ( $GAS$ ) = [ &p0& *100000.0] # initial pressure in [Pa]
...
begin_alias{ }
"wall" = " BC$...$ ACTIVE$...$ IDENT%...% MAT$GAS$ TOUCH%...% MOVE$...$ LAYER0 CHAMBER1 "
end_alias
```

Do not forget the absolute_pressure (see also COEFF_p_divV)!!!

absolute_pressure

initial pressure [Pa] which is added to get the absolute pressure

Define the absolute pressure for the material with index **$Material$** :

```
absolute_pressure ( $Material$ ) = RightHandSideExpression
```

This is needed if Redlich Kwong gas law (see RedlichKwongGasLaw) and/or COEFF_p_divV is used!!!

cv

*specific heat of the material in J/(kg*K)*

Define the specific heat for the material with index **$Material$** :

```
cv( $Material$ ) = RightHandSideExpression
```

Alternatively:

```
specificheat( $Material$ ) = RightHandSideExpression
```


density

material density [kg/m^3]

Define the density for the material with index **$Material$** :

```
density( $Material$ ) = RightHandSideExpression
```

eta

*viscosity definition [Pa*s]*

Define the viscosity for the material with index **$Material$** :

```
eta( $Material$ ) = RightHandSideExpression
```

Alternatively:

```
viscosity( $Material$ ) = RightHandSideExpression
```

List of members:

%MED_JOHNSON_COOK%	parameters for calculating viscosity in the Johnson-Cook model
%MED_LIQUID_FILM%	viscosity definition in liquid films [Pa*s]

%MED_JOHNSON_COOK%

parameters for calculating viscosity in the Johnson-Cook model

Since the yield stress in the Johnson-Cook model can become negative, resulting in a negative viscosity, the user can specify a minimum viscosity to avoid this.

```
eta( $Material$ ) = ( %MED_JOHNSON_COOK% , minimum_allowed_viscosity, OPTIONAL: eps_dot_0 )
```

eps_dot_0: reference strain rate $\dot{\varepsilon}_0$ in JohnsonCook equation. If nothing is set, then the default is 1.0!

%MED_LIQUID_FILM%

*viscosity definition in liquid films [Pa*s]*

Define two viscosities: one for the normal direction, one for the tangential direction (applies only if DROPLETPHASEis active).

```
eta( $Material$ ) = ( %MED_LIQUID_FILM% , etaNormal, etaTangential )
```

- etaNormal :: defines η_{drop}^{normal} in the numerical scheme of DROPLETPHASE
- etaTangential ::defines $\eta_{drop}^{tangential}$ in the numerical scheme of DROPLETPHASE

Hint:

```
eta( $Material$ ) = etaGeneral
```

defines the same eta both in normal and tangential directions.

gravity

define gravity or body forces of a material [m/s^2]

```
gravity( $Material$ ) = ( g_x, g_y, g_z)
```

g_x, g_y, g_z are the components of the vector of gravity / body forces.
They are subject to the RightHandSideExpression .

MESHFREE · InputFiles · USER_common_variables · PhysicalProperties · heatsource

heatsource

heat source [W/m^3]

Define a heat source for the material with index **$Material$** :

```
heatsource( $Material$ ) = RightHandSideExpression
```

MESHFREE · InputFiles · USER_common_variables · PhysicalProperties · lambda

lambda

*thermal conductivity [W/(m*K)]*

Define the thermal conductivity for the material with index **$Material$** :

```
lambda( $Material$ ) = RightHandSideExpression
```

Alternatively:

```
thermalconduction( $Material$ ) = RightHandSideExpression
```

MESHFREE · InputFiles · USER_common_variables · PhysicalProperties · mue

mue

shear modulus definition [Pa]

This value refers to the parameter μ in the StressTensorAlgorithm .
The different options are listed below.

List of members:

PureElastic	elastic modulus
JohnsonCook	Johnson-Cook model
GeneralYieldStress	provide a general formulation/model of the yield stress

MESHFREE · InputFiles · USER_common_variables · PhysicalProperties · mue · GeneralYieldStress

GeneralYieldStress

provide a general formulation/model of the yield stress

General definition of the yield stress for the material with index **$Matflag$** depending on the simulation results.

```
mue( $Matflag$ )= ( %MED_YIELDSTRESS%, mue0, Syield, OPTIONAL:Relax )
```

Syield: yield stress depending on any parameter, see General and LIQUID
mue0: shear modulus in regions of linear elastic stress (before reaching the yield stress)

Relax: parameter in [0,1] for the upper bound of the rate of change of the stresses
(e.g. Relax=0.3 means that the stresses are allowed to change by 30% from one time step to the next)

All of these values are of type RightHandSideExpression .

In order to extract a proper μ to be used to integrate the stress tensor by the StessTensorAlgorithm,
the expression for Syield is numerically differentiated with respect to the plastic strain (see %ind_eps_plastic%).

$$\mu_{\text{effective}} = \frac{d\text{Syield}}{d\epsilon_{\text{plastic}}}$$

Note: A positive correspondence between Syield and %ind_eps_plastic% has to be provided.

List of members:

DruckerPragerModel	use the GeneralYieldStress functionality to describe the behavior of granular materials

DruckerPragerModel

use the GeneralYieldStress functionality to describe the behavior of granular materials

The Drucker-Prager model provides a yield stress depending on the pressure.

$$\boldsymbol{S}_{\text{yield}}(p) = C_{\text{DruckerPrager}} \cdot (p + p_{\text{PreCompression}}) + \boldsymbol{S}_{\text{yield}}^{\text{fictitious}}$$

Numerically, we require the following stability constraints.

1.) Limit the change of the yield stress from one time step to the next by filtering (value of alpha):

$$\bar{\boldsymbol{S}}_{\text{yield}}^{n+1} = \max\left((1-\alpha)\boldsymbol{S}_{\text{yield}}^{n} + \alpha \boldsymbol{S}_{\text{yield}}(p^{n+1}), \boldsymbol{S}_{\text{yield}}^{\min}\right)$$

$$\boldsymbol{S}_{\text{yield}}^{n+1} = \max\left(\min\left(\bar{\boldsymbol{S}}_{\text{yield}}^{n+1}, \boldsymbol{S}_{\text{yield}}^{\max}\right), \boldsymbol{S}_{\text{yield}}^{\min}\right)$$

2.) Feasible viscosity:

- Provide sufficient numerical viscosity (see StressTensorAlgorithm) by imposing an effective μ dependent on the plastic strain, i.e. enhance the yield stress formulation.

$$\boldsymbol{S}_{\text{yield}}^{n+1} = \max\left(\min\left(\bar{\boldsymbol{S}}_{\text{yield}}^{n+1}, \boldsymbol{S}_{\text{yield}}^{\max}\right), \boldsymbol{S}_{\text{yield}}^{\min}\right) \cdot (1 + C_{\mu} \cdot \epsilon_{\text{plastic}})$$

- Alternatively, provide a sufficient viscosity of the following form.

$$\eta = C_{\eta} \cdot \boldsymbol{S}_{\text{yield}}^{n+1} \frac{h}{\|\boldsymbol{v}\|}$$

For examples, see Sand .

MESHFREE · InputFiles · USER_common_variables · PhysicalProperties · mue · JohnsonCook

JohnsonCook

Johnson-Cook model

The material with index **$Matflag$** behaves according to the Johnson-Cook model.

```
mue( $Matflag$ )= ( %MED_JOHNSON_COOK% , mue0, A, B, n, C, m, Tm, T0, OPTIONAL:Relax )
```

A , **B** , **n** , **C** , **m** , **Tm** , **T0** : definition of the yield stress motivated by the Johnson-Cook model which is given by

$$\sigma_{\text{yield}} = [A + B\varepsilon^n]\left[1 + C\ln\left(\frac{\dot{\varepsilon}}{\dot{\varepsilon}_0}\right)\right]\left[1 - \left(\frac{T - T_0}{T_m - T_0}\right)^m\right]$$

mue0: shear modulus in regions of linear elastic stress (before reaching the yield stress)

Relax: parameter in [0,1] for the upper bound of the rate of change of the stresses
(e.g. Relax=0.3 means that the stresses are allowed to change by 30% from one time step to the next)

The reference strain rate $\dot{\varepsilon}_0$ is set to 1.0 by default, but the user can change it optionally (see e t a , %MED_JOHNSON_COOK%).

All of these values are of type RightHandSideExpression .

MESHFREE · InputFiles · USER_common_variables · PhysicalProperties · mue · PureElastic

PureElastic

elastic modulus

pure elastic material behavior

```
mue( $Material$ ) = RightHandSideExpression
```

MESHFREE · InputFiles · USER_common_variables · PhysicalProperties · sigma

sigma

surface tension [N/m]

Define the surface tension for the material with index **$Material$** :

```
sigma ( $Material$ ) = RightHandSideExpression
```

Alternatively:

```
surfacetension( $Material$ ) = RightHandSideExpression
```

MESHFREE · InputFiles · USER_common_variables · PointCloudQualityCheck

3.1.24. PointCloudQualityCheck

check the quality of a read in point cloud

If a point cloud is read by ReadInPointCloud , then a quality check is performed with exactly the point cloud read, and then the program is stopped thereafter. See also qualitycheck .

3.1.25. PointCloudReduction

select/mark MESHFREE points by reducing the point cloud

PointCloudReduction (n) = (f_Integration, f_Target, OPTIONAL:%PointCloudReduction_UseOldTimeStep%)

Select MESHFREE points out of the complete point cloud that represent a certain target quantity.
The algorithm aims to estabish connected subdomains. For each subdomain j we require:

$$f_{\text{Target}} \leq \sum_{i \in \Omega_{\text{sub}}(j)} f_{\text{Integration},i}$$

Only one point out of the cluster j is marked.

The result of the PointCloudReduction can be requested by the reduct() -functionality in Equations :

- marked MESHFREE point in a cluster represents the value of the integral $\sum_{i \in \Omega_{\text{sub}}(j)} f_{\text{Integration},i}$

  reduct(n, %EQN_Reduct_Accumulated%)

- marked MESHFREE point represents the cluster index j of $\Omega_{\text{sub}}(j)$

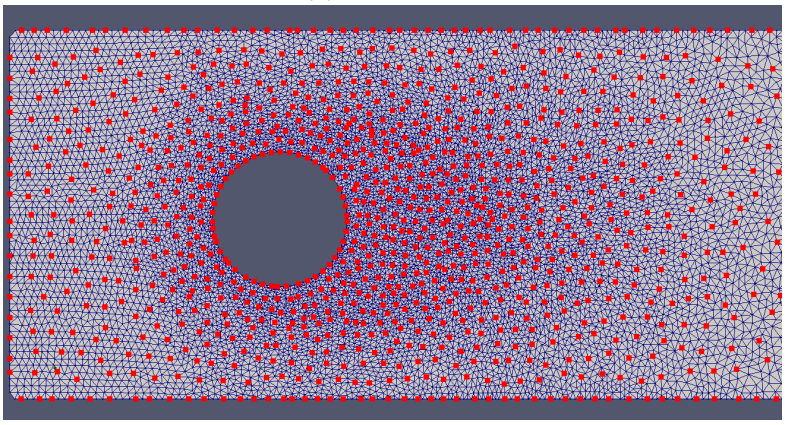
  reduct(n, %EQN_Reduct_iCluster%)

%PointCloudReduction_UseOldTimeStep%: MESHFREE tries to use the reduction results of the previous time step first (i.e. keep the selection status of points from the previous time step if possible). Then, it runs the reduction on top of it.
Under this option, the reduction results are stored on the point cloud (in order to keep this info for the next time cycle), which requires additional memory for each PointCloudReduction which is subject to this option.

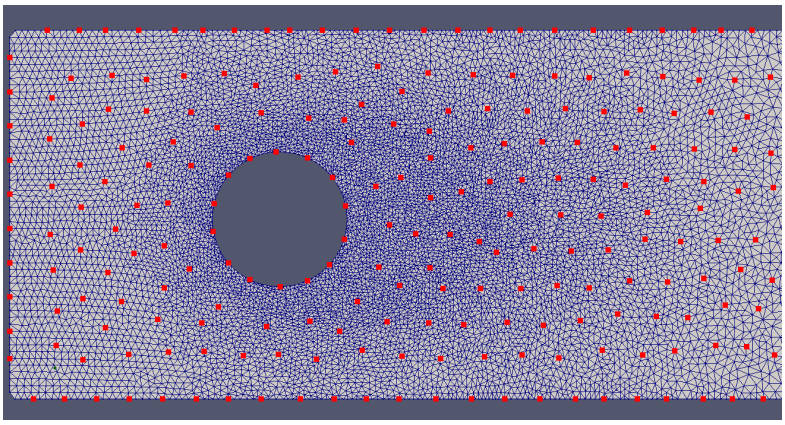
Examples:

```
PointCloudReduction (1) = ( [1], [10] ) # mark every 10-th MESHFREE point
SAVE_ITEM = ( %SAVE_scalar% , [reduct(1,%EQN_Reduct_Accumulated%)], "nbPointsRepresented" ) # how many points are represented by the marked point
SAVE_ITEM = ( %SAVE_scalar% , [reduct(1,%EQN_Reduct_iCluster%)], "numberingClusteringIndex" ) # display the cluster index (index of fish scale) produced by the PointCloudReduction
PointCloudReduction (2) = ( [Y %ind_Vi% ], [ &Hmax& ^3] ) # mark MESHFREE points which represent a volume that is approximately equal to &Hmax& ^3
SAVE_ITEM = ( %SAVE_scalar% , [reduct(2,%EQN_Reduct_Accumulated%)], "volumeRepresented" ) # how many points are represented by the selected point
SAVE_ITEM = ( %SAVE_scalar% , [reduct(2,%EQN_Reduct_iCluster%)], "volumeClusteringIndex" ) # display the cluster index (index of fish scale) produced by the PointCloudReduction
PointCloudReduction (3) = ( [reduct(1,%EQN_Reduct_Accumulated%)>0], [10] ) # mark every 10-th MESHFREE point out of the PointCloudReduction (1), i.e. every 100-th point
SAVE_ITEM = ( %SAVE_scalar% , [reduct(3,%EQN_Reduct_Accumulated%)], "volumeRepresented" ) # how many points are represented by the marked point
SAVE_ITEM = ( %SAVE_scalar% , [reduct(3,%EQN_Reduct_iCluster%)], "volumeClusteringIndex" ) # display the cluster index (index of fish scale) produced by the PointCloudReduction
```

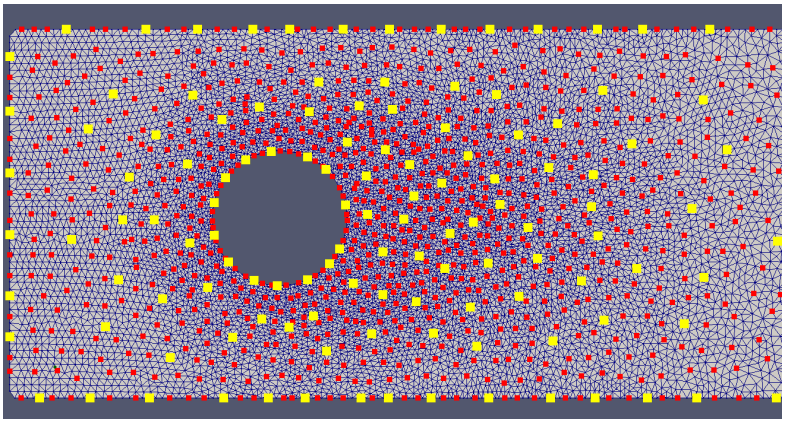
result of PointCloudReduction (1):



result of PointCloudReduction (2):



result of PointCloudReduction (3):



DOWNLOAD COMPREHENSIVE EXAMPLE

3.1.26. RESTART

control the restart functionality

This includes the writing of restart files with respect to a user-defined step size as well as
the launch of a MESHFREE simulation based on a previously saved restart file. For details see below.

Note: One needs to consider a few things when using restarts
and the SAVE format HDF5ERF, see RestartIssues .

List of members:

LaunchRestart	launch MESHFREE on the basis of a restart file
RestartStepSize	define after how many time cycles a restart file has to be generated
DefineRestart	save restart files
RestartPath	Define path and file name of restart files

DefineRestart

save restart files

The write-out of restart files is defined by RestartStepSize .

Example 1:

```
restart = (0)
RestartStepSize = ( 100, %RESTART_sequence% )
```

Write a restart file every 100 time cycles. The restart files are numbered from 1 until N. All files are kept on disc.
This consumes memory, but a restart is possible from any restart file. To invoke the restart, just set

```
restart = (n)
```

n is the number/index of the restart file.

Example 2 (default):

```
restart = (0)
RestartStepSize = ( 100, %RESTART_single% )
```

Write a restart file every 100 time cycles. Every new restart file overwrites the existing old one.
This saves memory, but there is a risk. A restart file can be corrupt or the computer crashes during the write operation of the restart file. In order to invoke the restart, just set

```
restart = (1)
```

Example 3 (switch off):

```
restart = (0)
RestartStepSize = ( 0, %RESTART_single% )
```

For **n** =0, no restart files are written.

Note: Restart files can also be triggered by an EVENT , see %EVENT_WriteRestart% , or by the signal save in the SIGNAL-file, see SequentialReadingOfSignalFile .

See also checkpoint for an alternative automatic restart functionality.

LaunchRestart

launch MESHFREE on the basis of a restart file

```
Restart = m
```

m is the ordinal number of the restart file. This will launch MESHFREE based on the restart file with number m.

Default: Restart = 0 (do not launch by restart file, but start from the beginning)

The behavior for reading and writing restart files is explaines in RestartPath .

List of members:

ExchangeBEOnRestart	exchange parts of the boundary elements during restart

ExchangeBEOnRestart

exchange parts of the boundary elements during restart

This functionality allows to include additional boundary elements files during restart. The alias definitions for these new boundary elements have to be copied from pre-restart aliases. Furthermore, pre-restart aliases and their associated boundary elements can be removed on restart.

```
restart_additionalBE = ( NewBEFile, NewGeometryManipulations , NewGeometryRestrictions )
restart_copying = ( CopyFromAlias, CopyToNewAlias )
restart_toberemoved = ( RemoveAlias1, RemoveAlias2, ... )
```

Example:
Read in only **top_new** from the additional boundary elements file **geometryfile2.FDNEUT** , copy the alias definition for **top_new**
from the pre-restart alias **top** , as well as remove the boundary elements associated to the pre-restart aliases **top** and **bottom**
during restart.

```
begin_boundary_elements{ }
include{ geometryfile1.FDNEUT} # this file contains the aliases top and bottom
end_boundary_elements
begin_alias{ }
"top" = " BC$wall$ ACTIVE$noinit_always$ IDENT%BND_slip% MAT$Mat1$ TOUCH%TOUCH_liquid%
MOVE$MOVE_none$ LAYER0 CHAMBER1 " # alias top
"bottom" = " BC$wall$ ACTIVE$noinit_always$ IDENT%BND_slip% MAT$Mat1$ TOUCH%TOUCH_liquid%
MOVE$MOVE_none$ LAYER0 CHAMBER1 " # alias bottom
end_alias
restart_additionalBE = ( geometryfile2.FDNEUT, only{ top_new} )
restart_copying = ( top, top_new )
restart_toberemoved = ( top, bottom )
```

Note: Filling of the new boundary elements can be controlled by the parameter restartnewBE_filling.

List of members:

restart_additionalBE	include additional boundary elements file during restart
restart_copying	copy alias definition for additional boundary elements during restart
restart_toberemoved	remove pre-restart boundary elements during restart

restart_additionalBE

include additional boundary elements file during restart

Additional boundary elements can be included during restart by:

```
restart_additionalBE = ( NewBEFile, NewGeometryManipulations , NewGeometryRestrictions )
```

NewBEFile: Additional boundary elements file to be included. The same formats are supported as for include{ File}.

The categories NewGeometryManipulations and NewGeometryRestrictions are optional. None, a choice of them, or even all of them
in the same statement/line are accepted. They have to be separated by a comma.

Warning: The alias definitions for additional boundary elements have to be copied from pre-restart aliases by restart_copying .
Pre-restart aliases and their associated boundary elements can be removed on restart by restart_toberemoved .

Alternative syntax: Restart_AdditionalBE

List of members:

NewGeometryManipulations	geometrical modifications of additional boundary elements files during restart
NewGeometryRestrictions	restrictions for additional boundary elements files during restart

NewGeometryManipulations

geometrical modifications of additional boundary elements files during restart

Options:

- scale{ }
- offset{ }
- rotate{ }
- mirror{ }

Functionality and syntax are the same as for include{ File} during classical start of a simulation.

NewGeometryRestrictions

restrictions for additional boundary elements files during restart

Options:

- only{ }
- ignore{ }
- append{ }
- sloppy{ }

Functionality and syntax are the same as for include{ File} during classical start of a simulation.

restart_copying

copy alias definition for additional boundary elements during restart

```
restart_copying = ( CopyFromAlias, CopyToNewAlias )
```

Copy the alias definition of the pre-restart alias **CopyFromAlias** for the additional alias **CopyToNewAlias** during restart.

Note: Copying requires additional boundary elements included by restart_additionalBE .
Pre-restart aliases and their associated boundary elements can be removed on restart by restart_toberemoved .

Alternative syntax: Restart_Copying

restart_toberemoved

remove pre-restart boundary elements during restart

```
restart_toberemoved = ( RemoveAlias1, RemoveAlias2, ... )
```

Remove the boundary elements associated to the pre-restart aliases **RemoveAlias1** , **RemoveAlias2** , ... during restart.

Alternative syntax: Restart_ToBeRemoved

MESHFREE · InputFiles · USER_common_variables · RESTART · RestartPath

RestartPath

Define path and file name of restart files

The file name and location of restart files may be specified in the UCV via the options restart_path and restart_file .

Example 1 (two arguments):

```
restart_path = ('RestartWriteFolder', 'RestartReadFolder')
restart_file = 'RestartFile'
```

The first argument of restart_path determines the path that restart files are written into.
The second argument of restart_path determines the path that restart files are read from.
Analogous to SAVE_path , both paths are influenced by FPM_RESULTDIR_PREFIX and two hidden files
1) .SYMLINK__ _FPM_ID_>>ID< <__ .symlink__="" _fpm_id_="" _to_restartpath_write="">> >ID< <__ _to_restartpath_read="" and="" are="" created="" directory="" link="" read="" the="" to="" which="" write="">> >ID<< is replaced by the ID of the corresponding MESHFREE run).
restart_file determines the file name of restart files. Restart files will follow the naming convention >>restart_file< <.restart_0001 where="">> >restart_file<< is replaced by the string supplied to the restart_file command.

In the above example, restart files RestartFile.restart_0001 would be saved to RestartWriteFolder/ in the working directory and read from RestartReadFolder/ in the working directory.

Example 2 (single argument):

```
restart_path = 'RestartFolder'
restart_file = 'RestartFile'
```

If only a single argument is supplied to restart_path , the read and write folder for restart files are identical.

NOTE
In particular, it is recommended to specify these options when begin_save enviroments are used. While it is not mandatory, it ensures that no unexpected storage locations or unexpected file names occur.

COMPATIBILITY
The behavior of older versions of MESHFREE and UCVs without the above commands is maintained through the following defaults:
If restart_path is not defined, it is set to SAVE_path .
If restart_file is not defined, it is set to '>>SAVE_file< <_0000 with="">> >SAVE_file<< being replaced by the string supplied to the SAVE_file command.

FALLBACK
If no appropriate restart file is found in restart_path/restart_file.restart_0001, as a fallback, a search for the file .restart_0001 is done within the working directory.
If also this file does not exist, the simulation will stop.

MESHFREE · InputFiles · USER_common_variables · RESTART · RestartStepSize

RestartStepSize

define after how many time cycles a restart file has to be generated

There are two different types:

- %RESTART_sequence% - produces consecutively numbered restart files
- %RESTART_single% - overwrites the restart file each time

See also DefineRestart .

List of members:

%RESTART_sequence%	define a sequence of restart files
%RESTART_single%	define the production of a single restart file

%RESTART_sequence%

define a sequence of restart files

```
RestartStepSize = ( n, %RESTART_sequence% , OPTIONAL:NumberFilesToKeep )
```

Every **n** time cycles, a new restart file is created. The restart files are numbered consecutively.
The names of the restart files are 'SAVE_file.restart_0001', 'SAVE_file.restart_0002', 'SAVE_file.restart_0003', ...

In addition to the restart file, a restart info file is created ('SAVE_file.restart_0001___countTS___time', ...).
It contains information on the current time step index and time.

Keep the last NumberFilesToKeep >0 restart files and let MESHFREE delete the older ones.

Note:

- Restart files can also be triggered by an EVENT , see %EVENT_WriteRestart% , or by the signal save in the SIGNAL-file,
  see SequentialReadingOfSignalFile . The additional restart file obtains the next ordinal number in the sequence of restart files.
- Restart info files are also written in case of EVENT - or SIGNAL-triggered writing of a restart file, e.g.
  'SAVE_file.restart_0001___EVENT___countTS___time' and
  'SAVE_file.restart_0001___SIGNAL___countTS___time'.

```
RestartStepSize = ( n, %RESTART_sequence% , OPTIONAL:NumberFilesToKeep ,
OPTIONAL:NumberFilesToKeepEVENT , OPTIONAL:NumberFilesToKeepSIGNAL )
```

Keep the last NumberFilesToKeep >0, NumberFilesToKeepEVENT >0, and NumberFilesToKeepSIGNAL >0
restart files with standard, EVENT , and SIGNAL trigger and let MESHFREE delete the older ones.

Note:

- NumberFilesToKeep refers only to restart files with standard trigger, i.e. %RESTART_sequence% .
  If a restart file is also triggered by an EVENT or a SIGNAL in the same time step, this restart file is excluded from the deletion process.
- NumberFilesToKeepEVENT refers only to restart files with EVENT trigger. If a restart file is also triggered by a SIGNAL in the same time step, this restart file is excluded from the deletion process. This also holds vice versa.

See also DefineRestart .

%RESTART_single%

define the production of a single restart file

```
RestartStepSize = ( n, %RESTART_single% )
```

Every **n** time cycles, a new restart file is created. The new file replaces the old one.
The name of the restart file is 'SAVE_file.restart_0001'.

In addition to the restart file, a restart info file is created ('SAVE_file.restart_0001___countTS___time').
It contains information on the current time step index and time.

Note:
Restart files can also be triggered by an EVENT , see %EVENT_WriteRestart% , or by the signal save in the SIGNAL-file, see SequentialReadingOfSignalFile . The new restart file replaces the old one irrespective of the trigger standard (%RESTART_single%), EVENT , or SIGNAL.

See also DefineRestart .

3.1.27. ReadInPointCloud

read in an already existing point cloud from file

Currently, pointcloud data read by the ReadInPointCloud functionality can only be used for STANDBY pointclouds.
In the near future, MESHFREE will be extended such that they can serve as initial pointcloud for classical chamber tasks such as LIQUID , DROPLETPHASE etc.
In order to get values from the STANDBY -pointcloud, employ approxY() only. No other function can be used so far.
The STANDBY -pointcloud is subject to all MPI-reorganization steps.

```
begin_pointcloud{ }
include{ Filename}
end_pointcloud
```

The list of supported file formats can be found below.

List of members:

ASCII	read in already existing point cloud from ascii format
EnSight	read in already existing point cloud from EnSight format

ASCII

read in already existing point cloud from ascii format

The ascii file format is the following:

```
# ASCII
# %ind_x(1)% %ind_x(2)% %ind_x(3)% ... %ind_kob%
realValue realValue realValue ... realValue
...
realValue realValue realValue ... realValue
```

The first line defines this file as ascii file. The second line tells what kind of values are contained in the given columns. There is no constraint on the order of the %ind_...%-items.

Warning: Currently, this option is only used for the PointCloudQualityCheck . This means, if a point cloud is read by this option,
then a quality check is performed with exactly the point cloud read, and then the program is stopped thereafter. See also qualitycheck .

Note: If the information of %ind_kob% is not given, all MESHFREE points read are assumed to be interior.

EnSight

read in already existing point cloud from EnSight format

In order to read in an ensight file, the following items have to be provided.

```
begin_pointcloud{ }
include{ /m/scratch/hive/FPM/JK/results/KarreOriginal.case} format{ensight} ...
timeFrame{20} ...
variables{ %ind_p%="pressure" , %ind_v(1)%="velocity" , %ind_h%="H" } ...
toChamber{11} ...
toMaterial{ $MatStandby$ }
end_pointcloud
KOP(11) = STANDBY
INITDATA ( $MatStandby$ , %ind_h% ) = 0.1
```

- timeFrame{n}: index of the time frame to be read. If the *.case file does not contain a TIME-section, this item can be omitted.
- variables{ %ind_FPM_1%="variableNameInEnsight" , %ind_FPM_2%="variableNameInEnsight" , ... } : be sure to use exactly the variable names as they appear in the *.case file
- toChamber{n}: chamber index given to the new MESHFREE points
- toMaterial{$Mat...$}: material index given to the new MESHFREE points -> especially useful if employing the INITDATA functionality to setup function values

Remark :

- If the smoothing length is present in the case-file and also read in by variables{ ..., %ind_h%="whateverTheName", ... }, then MESHFREE will be able to correctly establish a search tree for the MESHFREE points of the STANDBY -pointcloud.
- If smoothing length IS NOT present in the case file, it can still be defined by the INITDATA functionality.
- If smoothing length is NEITHER read in from the case file NOR defined by the INITDATA functionality, then MESHFREE tries to estimate the smoothing length by itself during the first 5 time cycles of the simulation and write the results into the variable %ind_h% . After the 5th time cycle, %ind_h% is not touched anymore.
- The smoothing length is particularly important for the neightbor-search for function approximation approxY() . If the smoothing length does not represent the point distribution, there might be serious inefficiencies or inaccuracies: if %ind_h% too big, MESHFREE has to handle too many neighbor points in the approxY() -function; if too small, MESHFREE might not find enough neighbors for a proper function approximation.
- The STANDBY pointcloud is not yet saved into the RESTART file.

List of members:

GeometryManipulations	geometry manipulations of the pointcloud upon read in of the case file
GeometryMovement	movement of the STANDBY-pointcloud during simulation
WriteOutManipulations	option to disable writing out the STANDBY point cloud

GeometryManipulations

GeometryManipulations

geometry manipulations of the pointcloud upon read in of the case file

The user has the opportunity to manipulate the geometry, coming from the case-file. In the same way as already done for boundary elements (see GeometryManipulations),
we can add USEFUL operations to the include statements:

```
begin_pointcloud{ }
include{ /m/scratch/hive/FPM/JK/results/KarreOriginal.case} format{ensight} ...
timeFrame{20} ...
variables{ %ind_p%="pressure" , %ind_v(1)%="velocity" , %ind_h%="H" } ...
toChamber{11} ...
toMaterial{ $MatStandby$ } ...
scale{ 2.0,1.0,1.0}, offset{ 1.0,0.0,0.0}
end_pointcloud
```

All items allowed in GeometryManipulations are also allowed here, however some of them do not make sense, such as reorientation{ etc.

MESHFREE · InputFiles · USER_common_variables · ReadInPointCloud · EnSight · GeometryMovement

GeometryMovement

movement of the STANDBY-pointcloud during simulation

The toMove{} functionality allows the user to let the pointcloud move during time integration. toMove{} assigns a proper MOVE -flag to the pointcloud:

```
begin_pointcloud{ }
include{ /m/scratch/hive/FPM/JK/results/KarreOriginal.case} format{ensight} ...
timeFrame{20} ...
variables{ %ind_p%="pressure" , %ind_v(1)%="velocity" , %ind_h%="H" } ...
toChamber{11} ...
toMaterial{ $MatStandby$ } ...
toMove{ $SomeDefinedMoveIndex$ }
end_pointcloud
MOVE ( $SomeDefinedMoveIndex$ ) = (%MOVE_...%, ... )
```

Any movement function as described in MOVE is allowed.

AN ALTERNATIVE to the toMove{}-functionality would be:

```
begin_pointcloud{ }
include{ /m/scratch/hive/FPM/JK/results/KarreOriginal.case} format{ensight} ...
timeFrame{20} ...
variables{ %ind_p%="pressure" , %ind_v(1)%="velocity" , %ind_h%="H" } ...
toChamber{11} ...
toMaterial{ $MatStandby$ }
end_pointcloud
INITDATA ( $MatStandbby$ , %ind_MOVE% ) = $SomeDefinedMoveIndex$
MOVE ( $SomeDefinedMoveIndex$ ) = (%MOVE_...%, ... )
```

i.e. the toMove{} is simply assigning the $SomeDefinedMoveIndex$ with the appropriate variable %ind_MOVE%

MESHFREE · InputFiles · USER_common_variables · ReadInPointCloud · EnSight · WriteOutManipulations

WriteOutManipulations

option to disable writing out the STANDBY point cloud

The writeOut{} functionality allows the user to stop writing out the STANDBY point cloud starting from a certain SAVE step or
to never write it out at all.

```
begin_pointcloud{ }
include{ /m/scratch/hive/FPM/JK/results/KarreOriginal.case} format{ensight} ...
timeFrame{20} ...
variables{ %ind_p%="pressure" , %ind_v(1)%="velocity" , %ind_h%="H" } ...
toChamber{11} ...
toMaterial{ $MatStandby$ } ...
writeOut{SomeInteger}
end_pointcloud
```

The given integer value is used as follows: If

```
SomeInteger < 0
```

the STANDBY point cloud is never written out. If

```
SomeInteger = 0
```

the STANDBY point cloud is written out at every SAVE step. In every other case
the cloud is only written out for the first SomeInteger SAVE steps.

3.1.28. RepeatCurrentTimeStep

repeat the current time step with different parameters or reduced pointcloud

Repeat the current time step with

1.) the same pointcloud, but with changes in the simulation parameters, see %RepeatCurrentTimeStep_BasedOnSamePointCloud% and RepeatCurrentTimeStep_ChangeCVconfiguration
2.) a reduced pointcloud, see %RepeatCurrentTimeStep_BasedOnReducedPointCloud% . Optionally, also here the simulation parameters can be changed.

MESHFREE creates a copy of the current pointcloud, does a time step, and deletes the pointcloud again. The only way to save data from the repeated time step is by RepeatCurrentTimeStep_SaveVariables .
Finally, there is the chance to also initialize parameters of the temporary pointcloud by the original one, see RepeatCurrentTimeStep_InitializeVariables .

```
# definitions of the repeating operations
RepeatCurrentTimeStep (n) = ( %RepeatCurrentTimeStep_BasedOnReducedPointCloud% ,
IndexOfPointCloudReduction, increaseFactorOf_H )
RepeatCurrentTimeStep (n) = ( %RepeatCurrentTimeStep_BasedOnSamePointCloud% )

RepeatCurrentTimeStep_ChangeCVconfiguration (n) = ( "ord_laplace=2" ,0, "ord_gradient=2" ) # set the approximation order (temporarily to 2 for the repeated time step)
RepeatCurrentTimeStep_SaveVariables (n) = ( %indU_2_p_corr%, %ind_p_dyn% , # save the dynamic pressure from the repeated time step in a user generated variable, see UserDefinedIndices
%indU_2_v(1)%, %ind_v(1)% , # save the x-component of the velocity from the repeated time step in a user generated variable, see UserDefinedIndices
%indU_2_v(2)%, %ind_v(2)% , # save the y-component of the velocity from the repeated time step in a user generated variable, see UserDefinedIndices
%indU_2_v(3)%, %ind_v(3)% ) # save the z-component of the velocity from the repeated time step in a user generated variable, see UserDefinedIndices
```

List of members:

%RepeatCurrentTimeStep_BasedOnReducedPointCloud%	repeat the current time step based on a reduced point cloud
%RepeatCurrentTimeStep_BasedOnSamePointCloud%	repeat current time step keeping the pointcloud exactly as original
RepeatCurrentTimeStep_ChangeCVconfiguration	change the configuration of the common_variables.dat for the repeating of time steps
RepeatCurrentTimeStep_InitializeVariables	initialize the (temporary) pointcloud of a repeating operation for particular entities
RepeatCurrentTimeStep_SaveVariables	save results from a repeated time step on the original pointcloud
RepeatCurrentTimeStep_AdditionalComputationsAfterDataTransfer	additinal computations on original pointcloud after data transfer is finished

DOWNLOAD COMPREHENSIVE EXAMPLE

%RepeatCurrentTimeStep_BasedOnReducedPointCloud%

repeat the current time step based on a reduced point cloud

```
PointCloudReduction (IndexOfPointCloudReduction) = ( [1], [8] ) # define a pointcloud reduction, here: select every 8th point
RepeatCurrentTimeStep (n) = ( %RepeatCurrentTimeStep_BasedOnReducedPointCloud% ,
IndexOfPointCloudReduction, increaseFactorOf_H ) # define a repeated execution of the time step by the given pointcloud reduction
```

- IndexOfPointCloudReduction: In order to invoke this option, a PointCloudReduction has to be necessarily active of the form

  ```
  PointCloudReduction (IndexOfPointCloudReduction) = ( [1], [8] ) # mark every 8-th MESHFREE point
  ```

  The timestep then is re-computed based on this reduced point cloud.

- increaseFactorOf_H: even though the reduced point cloud somehow suggests also an increase in H, the factor

desired has to be explicitely given here.
For example, marking every 8-th point would more or less mean increaseFactorOf_H=2.

REMARK: The reduced-pointcloud-algorithm passes through 3 main steps

- preparation:
  - establish a new, additional pointcloud structure based on the pointcloud reduction given
  - establish point search tree
  - establish neihborlists (employment of the neighborlists of the original pointcloud did not succeed in any case)
  - establish MPI communicatin structure for the reduced point cloud AND store the original one
  - reduce the neighbor lists due to the given NEIGHBOR_FilterMethod
  - reduce the neighbor lists finally due to the given max_N_stencil and max_N_stencil_INTERIOR
- computation
  - initialization due to RepeatCurrentTimeStep_InitializeVariables
  - perform classical time step
  - postprocessing due to RepeatCurrentTimeStep_SaveVariables
- cleanup
  - delete point cloud structure
  - set in place the original MPI communication structure

%RepeatCurrentTimeStep_BasedOnSamePointCloud%

repeat current time step keeping the pointcloud exactly as original

```
RepeatCurrentTimeStep (n) = ( %RepeatCurrentTimeStep_BasedOnSamePointCloud% )
```

This makes sense only if numerical parameters are changed by RepeatCurrentTimeStep_ChangeCVconfiguration .

REMARK: The same-pointcloud-algorithm passes through 3 main steps.

- preparation:
  - establish a clone of the original pointcloud, i.e. no recomputation of neighborlists
  - bring in place all configuration changes requated by RepeatCurrentTimeStep_ChangeCVconfiguration , save the original configuration
- computation
  - initialization due to RepeatCurrentTimeStep_InitializeVariables
  - recompute the differential operators
  - continue to perform the classical time step
  - postprocessing due to RepeatCurrentTimeStep_SaveVariables
- cleanup
  - delete the clone of the original point cloud
  - reset the original configuration modified previously by RepeatCurrentTimeStep_ChangeCVconfiguration

---

RepeatCurrentTimeStep_AdditionalComputationsAfterDataTransfer

additinal computations on original pointcloud after data transfer is finished

As the repeating of the current time step happens after the original time step is already finished,
we need a means of computing additional values on the original pointcloud. WE MUST NOT use CODI_eq (see CODI), as this function is processed
DURING the executions of the original time step.

```
RepeatCurrentTimeStep (n) = ( %RepeatCurrentTimeStep_BasedOnSamePointCloud% ) # repeatine based on the same pointcloud
RepeatCurrentTimeStep_ChangeCVconfiguration (n) = ( "ord_laplace=2" ,0, "ord_gradient=2" ) # set the approximation order (temporarily to 2 for the repeated time step)
RepeatCurrentTimeStep_SaveVariables (n) = ( %indU_v(1)%, %ind_v(1)% , # define data trensfer from temporary to original point cloud
%indU_v(2)%, %ind_v(2)% ,
%indU_v(3)%, %ind_v(3)% ,
%indU_p_corr%, %ind_p_dyn% ,
%indU_c%, %ind_c% )
# perform additional computations on original pointcloud, based on the data transfered
RepeatCurrentTimeStep_AdditionalComputationsAfterDataTransfer (n) = ( %indU_Dv%, [ sqrt( (Y%indU_v(1)%-Y %ind_v(1)% )^2 + (Y%indU_v(2)%-Y %ind_v(2)% )^2 + (Y%indU_v(3)%-Y %ind_v(3)% )^2 )] ,
%indU_DpCorr%, [ abs(Y%indU_p_corr%-Y %ind_p_dyn% ) ] ,
%indU_Dc%, [ abs(Y%indU_c%-Y %ind_c% ) ] )
```

In the example above,we compute the difference between the velocity, dynamic pressure, and correction pressure solutions between the
original time step and the additionally performed time step. The results of the computations are written to the index variables %indU_Dv%, %indU_DpCorr%, and %indU_Dc%, respectively.

ATTENTION!!!!!!!
The additional computations are executed regardless of the order as they appear in the brackets, i.e. dependent solution cannot be produced. The follwing example

```
RepeatCurrentTimeStep_AdditionalComputationsAfterDataTransfer (n) = ( %indU_A%, [ ... ] ,
%indU_B%, [ ... ] ,
%indU_C%, [ Y%indU_A% + Y%indU_B% ] )
```

is constructed wrongly, as we presume a dependence of %indU_C% on %indU_A% and %indU_B% which cannot be provided by RepeatCurrentTimeStep_AdditionalComputationsAfterDataTransfer .

RepeatCurrentTimeStep_ChangeCVconfiguration

change the configuration of the common_variables.dat for the repeating of time steps

```
RepeatCurrentTimeStep (n) = ( %RepeatCurrentTimeStep_BasedOnSamePointCloud% ) # run repeated time step with a clone of the original pointcloud
RepeatCurrentTimeStep_ChangeCVconfiguration (n) = ( "ord_laplace=2" ,0, # set order of neuman boundary conditions to linear ansatz functions
"ord_gradient=2" ,0, # gradient computation based on linear ansatz functions
"ChangeWhateverParameterYouLike = ValueRequired" ) # change any other value that can be set in common_variables
```

- This is especially useful if working with the same pointcloud, that means using %RepeatCurrentTimeStep_BasedOnSamePointCloud% .
- A list of common_variables lines can be given. The numerical configuration is changed only temporarily for the n-th repeating, and then reset to the original values
- The common_variables - items have to be separated by 0 (or any other number) currently, as there still seems a bug in reading RightHandSideExpression if containing more than one string-objects

---

RepeatCurrentTimeStep_InitializeVariables

initialize the (temporary) pointcloud of a repeating operation for particular entities

```
RepeatCurrentTimeStep (n) = ( %RepeatCurrentTimeStep_BasedOnSamePointCloud% )
RepeatCurrentTimeStep_InitializeVariables (n) = ( %ind_TemporaryPC%, %ind_OriginalPC%,
%ind_2_TemporaryPC%, %ind_2_OriginalPC%,
etc. )
```

- the temporary pointcloud is initialized with the current values of the original pointcloud
- with this feature, we can preset dedicated entities with other values
- always give pairs of indices
- all indices apply, als user defined indices %indU_...%

example:

```
RepeatCurrentTimeStep_SaveVariables (n) = ( %indU_v(1)%, %ind_v(1)% ,
%indU_v(2)%, %ind_v(2)% ,
%indU_v(3)%, %ind_v(3)% )
RepeatCurrentTimeStep_InitializeVariables (n) = ( %ind_v(1)% , %indU_v(1)%,
%ind_v(2)% , %indU_v(2)%,
%ind_v(3)% , %indU_v(3)% )
```

This example shows how to save the velocity result of the temporary pointcloud in the variables %indU_v(i)%, and then write them
back to the temporary pointcloud in the next time cycle.

RepeatCurrentTimeStep_SaveVariables

save results from a repeated time step on the original pointcloud

```
RepeatCurrentTimeStep (n) = ( %RepeatCurrentTimeStep_BasedOnSamePointCloud% )
RepeatCurrentTimeStep_SaveVariables (n) = ( %ind_OriginalPC%, %ind_TemporaryPC%,
%ind_2_OriginalPC%, %ind_2_TemporaryPC% ,
etc. )
```

- after execution of the repeated time step, the temporary pointcloud is deleted
- the only way to keep function values is to copy them (by the present feature) from the temporary to the original pointcloud
- always give pairs of indices
- all indices apply, als user defined indices %indU_...%

example:

```
RepeatCurrentTimeStep_SaveVariables (n) = ( %indU_v(1)%, %ind_v(1)% ,
%indU_v(2)%, %ind_v(2)% ,
%indU_v(3)%, %ind_v(3)% )
```

This example shows how to save the velocity result of the temporary pointcloud ind the variables %indU_v(i)%

3.1.29. SAVE

save computational results in different formats

MESHFREE allows to save results to different file formats, see SAVE_format . The user can save multiple formats at once. If multiple values for SAVE_path are specified, everything is stored in all given locations.

The output frequency is defined via SAVE_choose_meth , SAVE_first , and SAVE_interval . In the example below, the output frequency for all three output formats is changed after 999 timesteps.

All file formats will always save the point coordinates. For some formats additional variables like normals are saved through specifications in the SAVE_format . Other simulation variables need to be specified through SAVE_ITEM statements. For more specialized options, see the links at the bottom.

The location for output is specified through SAVE_file and SAVE_path .

Example:

```
SAVE_format (1) = 'ENSIGHT6 BINARY N---'
SAVE_format (2) = 'ASCII BINARY N---'

SAVE_choose_meth = 'CONT'
SAVE_first (1) = 1
SAVE_interval (1) = 5
SAVE_first (2) = 1000
SAVE_interval (2) = 1

SAVE_file = 'simulation'
SAVE_path = 'results'

SAVE_ITEM = ( %SAVE_scalar%, ScalarExpression, "ScalarDescriptionText" )
SAVE_ITEM = ( %SAVE_vector%, xVectorExpression, yVectorExpression, zVectorExpression, "VectorDescriptionText"
)
```

For saving different file formats or multiple saves with different SAVE parameters, one can alternatively use the experimental begin_save{ environment, which allows for a more intuitive handling of these cases.

List of members:	
BE_MAP	Define mapping from points to BE
begin_save{	Experimental handling of multiple save formats
SAVE_BE_ITEM	item of BE surfaces to be saved for visualization
SAVE_BE_MONITOR_ITEM	monitor item to be saved per BE element for visualization
SAVE_BE_NODE_ITEM	item of BE nodes to be saved for visualization
SAVE_choose_meth	save computational results in different formats
SAVE_CoordinateSystem	saving relative to specified coordinate system (movement)
SAVE_file	file name for the results
SAVE_filter	(Experimental) Filtering of saved Pointcloud via expression
SAVE_first	control first save
SAVE_format	format to save simulation data
SAVE_format_skip	skipping cycle for SAVE_format
SAVE_interval	control saving frequency
SAVE_ITEM	item to be saved for visualization
SAVE_MONITOR_ITEM	monitor item to be saved for visualization
SAVE_path	absolute or relative path for the simulation results
SAVE_PID_ITEM	PID item to be saved for visualization

BE_MAP

Define mapping from points to BE

To map values from nearby points to the centroids of boundary elements one may specify

```
BE_MAP ( $BEmap1$ ) = ( ExpressionToMap, OPTIONAL: iChamber , OPTIONAL: FilterExpression , OPTIONAL: iMethod , OPTIONAL: alphaKernel )
```

The results of this mapping may then be saved via

```
SAVE_BE_ITEM = ( %SAVE_scalar%, [ BEmap( $BEmap1$ ) ], "BE_BEmap1" )
```

Note: This functionality should currently only be used in conjunction with SAVE_BE_ITEM and BEmap() .

Arguments :

- **ExpressionToMap** : This expression is evaluated for each point which is included in the mapping and its result is mapped to the BE, e.g. inline equation
- **iChamber** : index of chamber for which the mapping is done. If the BE is in a different chamber, no mapping is done and zero is returned. *default* : 0 (filtering off, consider all chambers)
- **FilterExpression** : Points are only included in the mapping if the result of this expression is bigger than zero, *default* : 1.0 (filtering off, i.e. consider all points)

- **iMethod** : Mapping method (see below), *default* : %EQN_BEmap_ClosestPoint%
- **alphaKernel** : Parameter to control the shape of the weighting function for %EQN_BEmap_Shephard%, *default* : 1

Mapping methods :
- %EQN_BEmap_ClosestPoint%: Take the value of the point which is closest to the boundary element centroid
- %EQN_BEmap_Shephard%: Take the Shephard interpolation (cf. projY()) over all points located near the boundary element

Examples :
- Use default mapping method for BEs in any chamber and without any point filtering (to map the total pressure to the boundary)

```
BE_MAP ( $BEmap1$ ) = ( [ Y %ind_p% + Y %ind_p_dyn% ] )
```

- Use default mapping method for BEs in chamber 1 and without any point filtering (to map the total pressure to the boundary)

```
BE_MAP ( $BEmap1$ ) = ( [ Y %ind_p% + Y %ind_p_dyn% ], 1 )
```

- Choose mapping method for BEs any chamber and without any point filtering (to map the total pressure to the boundary)

```
BE_MAP ( $BEmap1$ ) = ( [ Y %ind_p% + Y %ind_p_dyn% ], 0, 1.0, %EQN_BEmap_ClosestPoint% )
```

- Choose mapping method for BEs in chamber 1 and filter out interior and free surface points from candidates for mapping (to map the total pressure to the boundary)

```
BE_MAP ( $BEmap1$ ) = ( [ Y %ind_p% + Y %ind_p_dyn% ], 1, equn{ $EQN_BEmap_BEfilter$ },
%EQN_BEmap_Shephard%, 0.1 )
begin_equation{ $EQN_BEmap_BEfilter$ }
if ( (Y %ind_kob% = %BND_none% ) + (Y %ind_kob% = %BND_free% ) ) :: -1.0
else :: 1.0
endif
end_equation
```

- Same example but with an inline equation for the filtering

```
BE_MAP ( $BEmap1$ ) = ( [ Y %ind_p% + Y %ind_p_dyn% ], 1, [ 1.0 - 2.0*( (Y %ind_kob% = %BND_none% ) +
(Y %ind_kob% = %BND_free% ) ) ], %EQN_BEmap_Shephard%, 0.1 )
```

Basic algorithm :
- For each BE, the centroid location is determined
- As candidates for the mapping, all points (from chamber iChamber) in the h-ball around the centroid of the BE are determined
- All inactive (Y%ind_vol%<0.1) points are removed from the list of candidates
- The filter expression is evaluated for each point and points with a value <=0 are removed from the list of candidates
- The mapping is done on the basis of the reduced list

Additional remarks :
- The default/fallback value can be changed via BEmap_DefaultValue

SAVE_BE_ITEM

item of BE surfaces to be saved for visualization

Save scalars or 3D vector items per boundary surface element, e.g. per triangle or quad.

```
SAVE_BE_ITEM = ( %SAVE_scalar%, ScalarExpression, "ScalarDescriptionText" )
SAVE_BE_ITEM = ( %SAVE_vector%, xVectorExpression, yVectorExpression, zVectorExpression,
"VectorDescriptionText" )
```

The arguments **xVectorExpression** , **yVectorExpression** , **zVectorExpression** , and **ScalarExpression** can be established
as regular RightHandSideExpression .

SAVE_BE_MONITOR_ITEM

monitor item to be saved per BE element for visualization

Saves scalar items in the EnSight case-file of the boundary for those boundary elements which correspond to monitor points.
Suitable monitor points are created through MONITORPOINTS_CREATION .

For each boundary element the scalar values of the defined item of all monitor points residing on this boundary element, e.g. a triangle, are summed. If there are no monitor points on a boundary element, the resulting value is -999999.

```
SAVE_BE_MONITOR_ITEM = ( WhatShallMESHFREEdo, ScalarExpression, "DescriptionText" )
```

WhatShallMESHFREEdo:

- %CUMU_NONE% (no cumulation between time steps, only values of the current time step)
- %CUMU_INTERVAL% (cumulation between time steps until time interval given by the definition of SAVE_interval is completed)
- %CUMU_SIMULATION% (cumulation between time steps throughout the whole simulation)
- %CUMU_SMOOTH% (smooth monitor items along boundary elements, using weight factor 1 for each cell)
- %CUMU_SMOOTH_AreaBased% (smooth monitor items along boundary elements (BE), using the area of the BE as weight factor)
- %CUMU_ASSIGN% (assign monitor items along boundary elements)

The argument **ScalarExpression** can be established as regular RightHandSideExpression .

The **DescriptionText** gets the prefix "UDPmon_" in the results file. For example, "velocity_magnitude" gets extended to "UDPmon_velocity_magnitude". The full name can then be used in ParaView's calculator for further operations.

List of members:

%CUMU_SMOOTH_AreaBased%	smooth monitor items along the boundary in every time step
%CUMU_SMOOTH%	smooth monitor items along the boundary in every time step
%CUMU_ASSIGN%	assign a value to a monitor item along the boundary
%CUMU_NONE%	do not cumulate the monitor values on the boundary elements (BE)
%CUMU_INTERVAL%	cumulate the monitor values of newly created monitor points on the BE a save interval is finished
%CUMU_SIMULATION%	cumulate the monitor values of newly created monitor points on the BE throughout the simulation

%CUMU_ASSIGN%

%CUMU_ASSIGN%

assign a value to a monitor item along the boundary

This feature should be used to assign quantities to boundary elements that are based on monitor point evaluation on boundary elements.

SAVE_BE_MONITOR_ITEM ($itemName$) = (%CUMU_SMOOTH% , **uValue** , "DescriptionText")

uValue has to be a function/value on boundary elements, direct point cloud attributes can not be used. A mapping to the boundary elements by the creation of monitor points and a SAVE_BE_MONITOR_ITEM o r BE_MONITOR_ITEM is necessary (cf. %CUMU_SMOOTH%).

The result of such an assignment can be used as input for a subsequent smoothing operation by %CUMU_SMOOTH% using BEmon() .

MESHFREE · InputFiles · USER_common_variables · SAVE · SAVE_BE_MONITOR_ITEM · %CUMU_INTERVAL%

%CUMU_INTERVAL%

cumulate the monitor values of newly created monitor points on the BE a save interval is finished

Cumulation between time steps until time interval given by the definition of SAVE_interval is completed.
That means, currently, all newly created monitorpoints will contribute to this item in a cumulative way. A reset of this item is performed after a SAVE_interval is finished.

MESHFREE · InputFiles · USER_common_variables · SAVE · SAVE_BE_MONITOR_ITEM · %CUMU_NONE%

%CUMU_NONE%

do not cumulate the monitor values on the boundary elements (BE)

No cumulation between time steps, only values of monitor points of the current time step are used.
That means, currently, all newly created monitorpoints will contribute to this item in the current time step.

MESHFREE · InputFiles · USER_common_variables · SAVE · SAVE_BE_MONITOR_ITEM · %CUMU_SIMULATION%

%CUMU_SIMULATION%

cumulate the monitor values of newly created monitor points on the BE throughout the simulation

Cumulation between time steps throughout the whole simulation.
That means, currently, all newly created monitorpoints will contribute to this item in a cumulative way. No reset of this monitor item is performed.

MESHFREE · InputFiles · USER_common_variables · SAVE · SAVE_BE_MONITOR_ITEM · %CUMU_SMOOTH%

%CUMU_SMOOTH%

smooth monitor items along the boundary in every time step

This feature should be used when smoothing of total physical quantities on boundary elements is desired in every time

step. In this case, the total sum-up of the quantity will not change by the smoothing.

SAVE_BE_MONITOR_ITEM ($itemName$) = (%CUMU_SMOOTH% , **Radius** , **WeightKernel** , **uValue** , OPTIONAL: %CUMU_SMOOTH_StopAtEdges% , "DescriptionText")

- **Radius** -> allowed interaction radius r between cells/boundary elements (see further down)
- **WeightKernel** -> the coefficient for the weight kernel α (see further down)
- **uValue** -> the cell function value u_i (see further down)
- OPTIONAL: %CUMU_SMOOTH_StopAtEdges% -> smoothing should not go over secondary edges (given by the angle criterion COMP_CosEdgeAngle)

Let us suppose given function values u_i for all boundary elements i .
We define a distribution of u_i from the boundary element (cell) i to the cell j by

$$\tilde{u}_{ij} = \frac{u_i}{\sum_k W_{ik}} W_{ij}$$

where

- $W_{ij} = \exp(-\alpha \cdot r_{ij})$ -> see **WeightKernel**
- $r_{ij} = \frac{\|\mathbf{x}_i^{COG} - \mathbf{x}_j^{COG}\|}{r^2}$ -> see **Radius**
- $\mathbf{x}_i^{COG}$ is the centrer of gravity of the i-th cell

The smoothed function $\tilde{u}_i$ is the sum of all distributions, i.e.

$$\tilde{u}_i = \sum_j \tilde{u}_{ji}$$

We have total conservation of the form

$$\sum_i u_i = \sum_i \tilde{u}_i.$$

Note:

- The **Radius** is independent of the SmoothingLength in the simulation. It has to be chosen according to the characteristic length of the boundary elements, e.g. a multiple >1 of the edge length of triangles.
- The smoothed distribution $\tilde{u}_{ij}$ can only be non-zero, if the cells i and j have a topological connection.
- **uValue** has to be a function/value on boundary elements, direct point cloud attributes can not be used. A mapping to the boundary elements
  by the creation of monitor points and a SAVE_BE_MONITOR_ITEM or BE_MONITOR_ITEM is necessary.
- Only function values $u_i \neq 0$ on boundary element i are smoothed. Thus, boundary elements with no corresponding monitor points, i.e. cumulation value of -999999, have to be treated properly.

Example : Produce a SAVE_BE_MONITOR_ITEM and smooth the result.

```
SAVE_BE_MONITOR_ITEM ( $item_1$ ) = ( %CUMU_SIMULATION% , [1],
"number_of_monitor_points_created_on_BE") # simply count the monitor points created in this boundar element
SAVE_BE_MONITOR_ITEM ( $item_2$ ) = ( %CUMU_SMOOTH% , 0.3, 3, equn{ $EQ_smooth_1$ },
"smoothed_number_of_monitor_points_created_on_BE") # smooth out the total number of created monitor points
begin_equation{ $EQ_smooth_1$ }
if ( BEmon( $item_1$ ) ! -999999 ) :: BEmon( $item_1$ )
else :: 0.0
endif
end_equation
```


%CUMU_SMOOTH_AreaBased%

smooth monitor items along the boundary in every time step

This feature should be used when smoothing of area based physical quantities on boundary elements is desired in every

time step. In this case, the integral over the boundary before and after smoothing will be the same.

SAVE_BE_MONITOR_ITEM ($itemName$) = (%CUMU_SMOOTH_AreaBased% , **Radius** , **WeightKernel** , **uValue** , OPTIONAL: %CUMU_SMOOTH_StopAtEdges% , "DescriptionText")

- **Radius** -> allowed interaction radius r between cells/boundary elements (see further down)
- **WeightKernel** -> the coefficient for the weight kernel α (see further down)
- **uValue** -> the cell function value u_i on the boundary element (see further down)
- OPTIONAL: %CUMU_SMOOTH_StopAtEdges% -> smoothing should not go over secondary edges (given by the angle criterion COMP_CosEdgeAngle)

Let us suppose given function values u_i for all boundary elements i .
We define a distribution of u_i from the boundary element (cell) i to the cell j by

$$\tilde{u}_{ij} = \frac{A_i u_i}{\sum_k A_k W_{ik}} W_{ij}$$

where

- A_i is the area of the i-th cell
- $W_{ij} = \exp(-\alpha \cdot r_{ij})$ -> see **WeightKernel**
- $r_{ij} = \frac{\|\mathbf{x}_i^{COG} - \mathbf{x}_j^{COG}\|}{r^2}$ -> see **Radius**
- $\mathbf{x}_i^{COG}$ is the centrer of gravity of the i-th cell

The smoothed function $\tilde{u}_i$ is the sum of all distributions, i.e.

$$\tilde{u}_i = \sum_j \tilde{u}_{ji}$$

We have integral conservation of the form

$$\sum_i u_i A_i = \sum_i \tilde{u}_i A_i.$$

Note:

- The **Radius** is independent of the SmoothingLength in the simulation. It has to be chosen according to the characteristic length of the boundary elements, e.g. a multiple >1 of the edge length of triangles.
- The smoothed distribution $\tilde{u}_{ij}$ can only be non-zero, if the cells i and j have a topological connection.
- **uValue** has to be a function/value on boundary elements, direct point cloud attributes can not be used. A mapping to the boundary elements
  by the creation of monitor points and a SAVE_BE_MONITOR_ITEM or BE_MONITOR_ITEM is necessary.
- Only function values $u_i \neq 0$ on boundary element i are smoothed. Thus, boundary elements with no corresponding monitor points, i.e. cumulation value of -999999, have to be treated properly.

Example : Produce a SAVE_BE_MONITOR_ITEM and smooth the result with respect to the area of the boundary elements.

```
SAVE_BE_MONITOR_ITEM ( $item_1$ ) = ( %CUMU_SIMULATION% , [1/BEarea(1)],
"number_of_monitor_points_created_per_area") # simply count the monitor points per area
SAVE_BE_MONITOR_ITEM ( $item_2$ ) = ( %CUMU_SMOOTH_AreaBased% , 0.3, 3, equn{ $EQ_smooth_1$ },
"smoothed_number_of_monitor_points_area_based") # smooth out the area based number of monitor points
begin_equation{ $EQ_smooth_1$ }
if ( BEmon( $item_1$ ) ! -999999 ) :: BEmon( $item_1$ )
else :: 0.0
endif
end_equation
```


SAVE_BE_NODE_ITEM

item of BE nodes to be saved for visualization

Save scalars or 3D vector item per boundary node.

```
SAVE_BE_NODE_ITEM = ( %SAVE_scalar%, ScalarExpression, "ScalarDescriptionText" )
SAVE_BE_NODE_ITEM = ( %SAVE_vector%, xVectorExpression, yVectorExpression, zVectorExpression,
"VectorDescriptionText" )
```

The arguments **xVectorExpression** , **yVectorExpression** , **zVectorExpression** , and **ScalarExpression** can be established
as regular RightHandSideExpression .

MESHFREE · InputFiles · USER_common_variables · SAVE · SAVE_CoordinateSystem

SAVE_CoordinateSystem

saving relative to specified coordinate system (movement)

By default the results are saved relative to the standard coordinate system (no movement). For each SAVE_format a specific coordinate system
for saving can be defined by

```
SAVE_CoordinateSystem (n) = $MOVEFlag$
```

n: assigns this attribute to SAVE_format (n)

$MOVEFlag$: reference to given MOVE -statement, defines the coordinate system relative to which the results are saved

Example:

```
SAVE_format (1) = 'ENSIGHT6 BINARY N---'
SAVE_CoordinateSystem (1) = $MOVE_vconst$
...
MOVE ( $MOVE_vconst$ ) = ( %MOVE_velocity% , 0.0, 0.0, 1.0)
```

MESHFREE · InputFiles · USER_common_variables · SAVE · SAVE_ITEM

SAVE_ITEM

item to be saved for visualization

Either scalar or 3D vector items can be saved.

```
SAVE_ITEM = ( %SAVE_scalar%, ScalarExpression, "ScalarDescriptionText" )
SAVE_ITEM = ( %SAVE_vector%, xVectorExpression, yVectorExpression, zVectorExpression, "VectorDescriptionText"
)
```

The arguments **xVectorExpression** , **yVectorExpression** , **zVectorExpression** , and **ScalarExpression**
can be established as regular RightHandSideExpression .
Example:

```
SAVE_ITEM = ( %SAVE_vector%, [Y %ind_v(1)% ], [Y %ind_v(2)% ], [Y %ind_v(3)% ], "velocity" ) # velocity vector
SAVE_ITEM = ( %SAVE_scalar%, [Y %ind_p% ], "hydrostatic_pressure" ) # hydrostatic pressure (part of the pressure due to gravity and other body forces, see HydrostaticPressure )
SAVE_ITEM = ( %SAVE_scalar%, [Y %ind_p_dyn% ], "dynamic_pressure" ) # dynamic pressure (part of the pressure due to dynamic or compression forces, see DynamicPressure )
SAVE_ITEM = ( %SAVE_scalar%, [Y %ind_T% ], "temperature" ) # temperature
SAVE_ITEM = ( %SAVE_scalar%, [Y %ind_h% ], "smoothing_length" ) # smoothing length
SAVE_ITEM = ( %SAVE_scalar%, [Y %ind_act% ], "activation_status" ) # activation status of point (to filter only active points)
SAVE_ITEM = ( %SAVE_scalar%, [Y %ind_cham% ], "chamber_index" ) # chamber index (to filter points of different phases in a multiphase setup)
SAVE_ITEM = ( %SAVE_scalar%, [Y %ind_dtb% /Y %ind_h% ], "normed_distance_to_boundary" ) # normed distance to boundary wrt smoothing length
SAVE_ITEM = ( %SAVE_scalar%, [Y %ind_kob% ], "kind_of_boundary" ) # geometrical type of point (interior, free surface, inflow, outflow, wall etc.)
```


SAVE_MONITOR_ITEM

monitor item to be saved for visualization

Saves items for points of the monitor point cloud. Suitable monitor points are created through MONITORPOINTS_CREATION .
The syntax of SAVE_MONITOR_ITEM is identical to the one of SAVE_ITEM .

```
SAVE_MONITOR_ITEM = ( %SAVE_scalar%, ScalarExpression, "ScalarDescriptionText" )
SAVE_MONITOR_ITEM = ( %SAVE_vector%, xVectorExpression, yVectorExpression, zVectorExpression, "VectorDescriptionText" )
```

The arguments **xVectorExpression** , **yVectorExpression** , **zVectorExpression** , and **ScalarExpression** can be established
as regular RightHandSideExpression .

See also MONITORPOINTS .

SAVE_PID_ITEM

PID item to be saved for visualization

```
SAVE_format (1) = 'ENSIGHT6 BINARY N---'

SAVE_choose_meth = 'CONT'
SAVE_first (1) = 1
SAVE_interval (1) = 5

SAVE_file = 'AnyFileName'
SAVE_path = 'AnyFilePath'

SAVE_ITEM = ( %SAVE_vector%, xVectorExpression, yVectorExpression, zVectorExpression, "VectorDescriptionText"
)
SAVE_ITEM = ( %SAVE_scalar%, ScalarExpression, "ScalarDescriptionText" )
SAVE_ITEM = ...

SAVE_PID_ITEM = ( SwitchExpression_1, "PID description" )
SAVE_PID_ITEM = ( SwitchExpression_2, "description of second PID item" )
SAVE_PID_ITEM = ...
```

The PID defines a selection. **SwitchExpression_1** , **SwitchExpression_2** , SwitchExpression_... are mathematical expressions.
If the expression is positive, then the MESHFREE point belongs to the PID-selection, otherwise it does not.

Note:

- Currently, up to 64 PID definitions are possible (number of bits of a double real).
- The description text appears in the result file.
- Currently, it works only for ENSIGHT6 BINARY.

Example 1: PID based on materials or chamber

```
SAVE_PID_ITEM = ([Y%ind_cham%=1], "WATER")
SAVE_PID_ITEM = ([Y%ind_cham%=2], "AIR")
```

Example 2: PID based on subregions

```
SAVE_PID_ITEM = ( [InDom("SubRegion1")], "PID_SUB_1")
SAVE_PID_ITEM = ( [InDom("SubRegion2")], "PID_SUB_2")
```

"SubRegion1" and **"SubRegion2"** have to be valid aliases which define closed geometrical domains.

SAVE_choose_meth

save computational results in different formats

Options:

- Saving mode based on the number of time cycles

```
SAVE_choose_meth = 'CONT'
```

- Saving mode based on simulation time.

```
SAVE_choose_meth = 'TIME'
```

See also SAVE_interval .
Note:

It is not possible to define different SAVE_choose_meth for different SAVE_format or SAVE_path via indexing. To define different methods in such cases, use begin_save{ environments instead.

SAVE_file

file name for the results

File name for the results, usually without extension.

See SAVE_path for complete description.

SAVE_filter

(Experimental) Filtering of saved Pointcloud via expression

The experimental SAVE_filter allows for filtering of the pointcloud via expression. Currently, this feature is restricted to ENSIGHT6 BINARY only. **Example:**

```
SAVE_format (1) = 'ENSIGHT6 BINARY N--T'
SAVE_filter (1) = [Y%ind_kob%=%BND_free%] # only save points of free surface.
```

SAVE_first

control first save

Start saving after a number of time cycles or a given simulation time.

See SAVE_interval for a more detailed description.

SAVE_format

format to save simulation data

SAVE_format specifies the format for result files for point cloud and geometry. The general syntax is:

```
SAVE_format = ' MainFormat FourFormatLetters AdditionalOptions(optional) '
```

Example:

```
SAVE_format (1) = 'ENSIGHT6 BINARY N---'
SAVE_format (2) = 'ASCII N---'
SAVE_format (3) = 'ERFHDF5 N---'
```

Main Formats

- "ENSIGHT6 BINARY "
- "ASCII " (only supports "N---", "N-T-" and "ONLY:PARTICLES")
- "ERFHDF5 " (only supports "N---" and "ONLY:PARTICLES")

Four format letters

Usage of the four format letters:

Four format letters	Meaning
'N---'	display only active nodes (Y %ind_act% > 0)
'A---'	display ALL nodes, even the inactive MESHFREE points; in this case the user should also save the quantity %ind_act% to distinguish these in postprocessing.
'N--T'	nodes and tetrahedra coming from the Delaunay decomposition of the MESHFREE point cloud
'N-T-'	nodes and surface triangles produced by Delaunay decomposition of the free surface and the regular boundaries
'NN--'	nodes and boundary normals, only for 'ENSIGHT6 BINARY'
'NC--'	nodes and connectivities between multiple chambers, useful for visualizing contact between phases, see PHASE_distinction .
'....P'	additional option for 'ENSIGHT6 BINARY'; invokes the visualization of the metaplanes, very useful for debugging

Additional options

- 'ONLY:PARTICLES' (Save only the MESHFREE points and do not save the geometry, as it might contain a huge amount of data).
  Example:
  ```
  usage: "SAVE_format(1) = 'ENSIGHT6 BINARY N--- ONLY:PARTICLES'"
  ```
- NO:PARTS (Do not split the MESHFREE point chambers into parts for 'ENSIGHT6 BINARY'. ParaView as well as VisIt have
  problems and usually produce errors, if one of the chambers disappears. This might be the case, if
  the SHALLOWWATER solver is used together with LIQUID , but SHALLOWWATER is switched off after a certain time.)
- TIMEACC:n set the number of decimal places in the case file for the time set. The standart format is e12.5, i.e. TIMEACC:5, bit this will lead to problems if saving every timecycle for a big time and small time steps size (say t=1 and dt=1.0e-6 cannot be resolved anymore in the case file). This option has only effect for ENSIGHT6 BINARY.

Note: When using begin_save{ environments, the command SAVE_format (i) with i>1 is no longer supported.

List of members:

ASCII	computation results column-wise in an ASCII formatted file
ENSIGHT6	computation results in Ensight6 format
ERFHDF5	computation results in ESI format ERF-HDF5

ASCII

computation results column-wise in an ASCII formatted file

Per save time, MESHFREE creates one big result file (ASCII_0001.dat, etc.). It columnwise contains the values for each active MESHFREE point.

- 1st column: time
- 2nd column: x-component of position
- 3rd column: y-component of position
- 4th column: z-component of position
- 5th column and up: results as defined by the SAVE_ITEM statements in the order as given in USER_common_variables .

Example:

```
SAVE_format (1) = 'ASCII N---'
SAVE_ITEM = ( %SAVE_scalar%, [ Y %ind_cluster% ], "iCluster" )
SAVE_ITEM = ( %SAVE_scalar%, [ Y %ind_Vi% ], "VolumePerPoint" )
```

ENSIGHT6

computation results in Ensight6 format

EnSight is a very common file format to save time series. It is supported by several visualization tools, e.g. ParaView.

EnSight results always start out with a case file. MESHFREE will write out two case files, one for the point cloud and one for the boundary elements. Their name is controlled by SAVE_file .

```
...
SAVE_file = 'simulation'
SAVE_path = 'results'
...
```

will produce files 'simulation.case' for the point cloud and 'BE_simulation.case' for the boundary elements in the subfolder 'results'. Because of the structure of the EnSight file format there are two additional hidden subfolders called '.EnsightData___simulation-output' and '.EnsightData_BE___BE_simulation-output', also depending on SAVE_file . These contain the actual data.

ENSIGHT6 has the following syntax:

```
SAVE_format (1) = ENSIGHT6 BINARY NNTTP ONLY:PARTICLES NO:PARTS
SAVE_format (2) = ENSIGHT6 BINARY ----
```

where the first is the maximum and the second is the minimum required syntax. At minimum at least four letters are required.

Their meaning is dependent on their position:

- 1. Position: -/N/A (zero-dimensional; points)
  - '-' Do not write extra node information. MESHFREE points might not be available in some visualization software for visualization as points. Positions are still written out for the triangulation.
  - 'N' Write out nodes as points.
  - 'A' Write out all points including inactive ones.
- 2. Position: -/N/C/S (one-dimensional; lines)
  - '-' Do not write out any lines.
  - 'N' Save point normals explicitely as line objects.
  - 'C' Save interface connectivities between chambers for each interface point.
  - 'S' Save segments/pathlines.
- 3. Position: -/T (two-dimensional; faces)
  - '-' Do not write out any triangles.
  - 'T' Write out triangulation of the surfaces of the point cloud.

- 4. Position: -/T (three-dimensional; solids)
  - '-' Do not write out any tetrahedra.
  - 'T' Write out tetrahedralization of the point cloud volume.

These four items may be followed by a 'P' to write out metaplanes in the BE case file.

ONLY:PARTICLES will only write out the point cloud but not the boundary elements. And NO:PARTS will save both the point cloud and the boundary elements as a single EnSight part each. This is to prevent potential problems with specific visualization software.

Note: In the EnSight6 standard (page 9-121ff, 851ff), the maximum length for part and variable names is 79. Longer names are cut at this length.

ERFHDF5

computation results in ESI format ERF-HDF5

Save data as ERFs:

```
SAVE_format (1) = 'ERFHDF5 N---'
```

ERF is short for "ESI RESULT FILE". It is the standardized data format of the ESI group, based on the HDF5 data format of the HDF group (Hierarchical Data Format). It can be used to store the data from a MESHFREE simulation, i.e. positions and velocity of the points and the boundary elements and self-defined SAVE_Items for these points.

List of members:

Introduction	General informations on ESI format ERF-HDF5
FurtherInformation	Further informations on ESI format ERF-HDF5
RestartIssues	Notes about using ERF-HDF5 with restarts

DOWNLOAD COMPREHENSIVE EXAMPLE

FurtherInformation

Further informations on ESI format ERF-HDF5

In this section we take a closer look at a specific ERF file, see HDFView_example.jpg , and try to understand what blocks the MESHRFREE-generated ERF files generally contain and which data is stored where.

Constant and varying data: ERF distinguishes between blocks with constant data and blocks with data that varies depending on independent variables. Mostly, "time" is the only independent variable. All constant data is stored in the file "constant" and all varying data is stored in the file "singlestate". In "singlestate" there are several subfiles called "stateXXXXXX" for the different timesteps under consideration. The number of the independent variables is stored in the file "indices", while the definition of them is done in the singlestates under "entityresults" in "indexident" (timestep number) and "indexeval" (concrete value of the time).

MESHFREE points and boundary elements: Two different kinds of simulation results are written out, the informations on the MESHFREE points and their respective SAVE_Item values and the informations on the boundary elements that represent the geoemtry. The MESHFREE points are treated as 0D Finite Elements, i.e. single points without triangulation, of the type "FPM", while the boundary elements are 2D Finite Elements of the type "SHELL".

Entity IDs: Every MESHFREE point n and every boundary element n is referenced via a unique ID entid(n). It should be stressed, that every boundary element, i.e. every triangle, is referenced by only one ID, not by three.

The connectivities files: There are two files that contain information on neighborhood relations. The "connectivities" file in the "constant" folder contains information on the fixed neighborhood relations between the boundary elements while the files with the same name in the "singlestate" folder contain information on the changing neighborhood relations between the MESHFREE points. But that is not all, the latter folder also associates point coordinates via its attributes with the simulation data of the SAVE_Items. It is a very important file that describes connectivities between sets of data in general.

The files variable and variablegroup: The constant file "variable" contains metadata on the SAVE_Items like their name, if they are scalars or vectors or their units. These variables have to be paired with a "variablegroup", because this is the standard procedure and not because it would be needed in the case of MESHFREE . So for every variable a variable group with the same name is created, which contains just this variable.

The entityresults files: The constant file "entityresults" contains the positions of the boundary elements at timestep 0. Its non-constant counterpart contains the simulation data for the MESHFREE points. The data for the SAVE_ITEMS is stored in "FPM" in "res" and referenced via the entity IDs. The file "FPMNODE" contains the absolute coordinates of the MESHFREE points and their vector-valued velocities. "SHELL" and "NODE" are the equivalents of "FPM" and "FPMNODE" for the boundary elements. "NODE" does not contain the absolute coordiantes of the boundary elements but their relative coordinates compared to the ones at timestep 0.

Distinguish between boundary parts: In "PART" the aliases for different parts of the boundary are saved in the dataset "title" and IDs for these parts are stored in "pid". This would be needed, if one wants to make some parts of the boundary invisible for visualization purposes.

Introduction

General informations on ESI format ERF-HDF5

The following files need to be linked respectively compiled to build ERF blocks with MESHFREE . This is currently done by default when building MESHFREE .

- The HDF5 library: libhdf5.a
- C routines from ESI: erfhdf5.cpp
- Fortran bindings: erf_api.h

The general structure of HDF5 is simple: It consists of so-called HDF5 groups (which are files), their properties and attributes (also together referred to as metadata) and the raw data (e.g. simulation results). The metadata and the raw data are together referred to as datasets.

An ERF file consists of so-called ERF blocks, which are formally HDF5 groups. How these blocks have to be structured and which kind of datasets they have to contain depends on the kind of results one wants to store. Many blocks are optional, only a few are mandatory. This is the reason why some programs, which support the ERF format, might not be able to process ERF files produced by MESHFREE , simply because these programs expect optional blocks that are not needed for MESHFREE . It should also be noted that MESHFREE always writes at least one boundary element out, even if "ONLY:PARTICLES" is selected, just to make its ERF files processable for more visualization programs.

Here are some useful links to delve further into the matter:

- The ERF documentation of the ESI group:
  https://myesi.esi-group.com/ERF-HDF5/
  Besides the documentation a handy program called HDF-View can also be downloaded here. It allows to read and write
  HDF5 files and visualizes the hierarchical structure of such a file.
- HDF5 tutorials from the HDF group:
  https://support.hdfgroup.org/HDF5/Tutor/

Below is an example, containing a common_variables , a USER_common_variables and a HDF5 file. It can be used to take a look at an actual ERF file or to change some of the SAVE_Items and see how this affects the produced ERF file.

EXAMPLE

MESHFREE · InputFiles · USER_common_variables · SAVE · SAVE_format · ERFHDF5 · RestartIssues

RestartIssues

Notes about using ERF-HDF5 with restarts

The currently used strategie for managing freed memory space when using ERFHDF5 with restarts is not yet optimal and may lead to erfh5 files occupying much more memory space than they actually need (after one or more restarts). These holes in the memory can be closed by using the following terminal commands:

```
h5repack FileName.erfh5 placeholder.erfh5
rm FileName.erfh5
mv placeholder.erfh5 FileName.erfh5
```

MESHFREE · InputFiles · USER_common_variables · SAVE · SAVE_format_skip

SAVE_format_skip

skipping cycle for SAVE_format

Additional control option for the frequency of saving result files associated to a specified SAVE_format .

```
SAVE_format_skip (n) = m
```

n: assigns this attribute to SAVE_format (n)

m is a positive integer which is used in the following procedure:

- Decision if results are saved in the current time step only according to SAVE_interval . If yes, update the save index.
- For each SAVE_format check if save index is divisible without remainder by the associated SAVE_format_skip -value **m** . If yes, save the results. Otherwise, skip saving.

Default: SAVE_format_skip (n) = 1 (i.e. frequency of saving only controlled by SAVE_interval)

Note: SAVE_first and SAVE_interval apply to each SAVE_format .

Example: Save each second time step for the first SAVE_format , only save every 6 time steps for the second SAVE_format , and
only save every 10 time steps for the third SAVE_format .

```
SAVE_format (1) = 'ENSIGHT6 BINARY N---'
SAVE_format (2) = 'ASCII N---'
SAVE_format (3) = 'ERFHDF5 N---'

SAVE_first (1) = 1
SAVE_interval (1) = 2

SAVE_format_skip (1) = 1 # this line is not necessary since it is the default
SAVE_format_skip (2) = 3
SAVE_format_skip (3) = 5
```

See also begin_save{ .

MESHFREE · InputFiles · USER_common_variables · SAVE · SAVE_interval

SAVE_interval

control saving frequency

SAVE_interval allows to control the frequency of saving result files. Its interpretation is either the number of time steps or the simulation time depending on SAVE_choose_meth .

The output frequency can be refined in intervals of interest by providing multiple SAVE_first and SAVE_interval statements. These apply to each SAVE_format .

An additional control option for the ouput frequency is given by SAVE_format_skip .

Example 1: time step dependent writeouts

```
SAVE_choose_meth = 'CONT'
SAVE_first (1) = 1
SAVE_interval (1) = 5
SAVE_first (2) = 1000
SAVE_interval (2) = 1
SAVE_first (3) = 1100
SAVE_interval (3) = 5
```

Here, starting from the first time step, after every 5 time cycles a result file is generated.
After 1000 time cycles, a result is generated after every time step.
Finally, after 1100 time cycles, the output again is generated after every 5 time steps.

Example 2: simulation time dependent writeouts

```
SAVE_choose_meth = 'TIME'
SAVE_first (1) = 0.5
SAVE_interval (1) = 0.1
```

This writes a result file each time a simulation time of $0.5 + 0.1n$ seconds $(n \geq 0)$ has been reached (or surpassed).

MESHFREE · InputFiles · USER_common_variables · SAVE · SAVE_path

SAVE_path

absolute or relative path for the simulation results

Use SAVE_file and SAVE_path to set the location for the results.

```
SAVE_file = 'FileBaseName'
SAVE_path = 'FilePath'
```

Multiple save paths:
If the results shall be saved into multiple different directories, one may define different values of SAVE_path within different begin_save{ environments.

Attention: The old indexing into save path, i.e. SAVE_path (n), is no longer supported and has been replaced by the begin_save{ functionality.

Prefix via environment variable:
With the command line option -r or the environment variable FPM_RESULTDIR_PREFIX, a prefix to the SAVE_path can be defined, for example to save all results inside the same directory on a large hard drive. This prefic will also apply to all definitions of SAVE_path within begin_save{ environments.

Symbolic links:
Every simulation generates one or more hidden files called

.SYMLINK__ _FPM_ID_ID_of_run__ _to_SAVEPATH_{number_of_save_path}

These are symbolic links to the location of a result file and can be used to access all SAVE paths conveniently from one place. In particular, if no begin_save{ environments are used, a single symbolic link is created. On the other hand, if begin_save{ environments exist, a symbolic link for each SAVE_path within these environments is created.

---

begin_save{

Experimental handling of multiple save formats

The experimental begin_save{ environment makes it possible to differentiate between several saving environments with different parameter settings and possibly different saving formats in a straightforward way. A user can define up to 10 begin_save{ environments, each with its own set of SAVE parameters. With this, simulation results might be saved in different files in different ways.

This environment uses the command SAVE_type , which allows the user to control the type of data that will be saved.

A begin_save{ environment may contain the following features:

- SAVE_choose_meth
- SAVE_format
- SAVE_first
- SAVE_interval
- SAVE_type
- SAVE_file
- SAVE_path
- SAVE_CoordinateSystem
- SAVE_ITEM

If any of the aforementioned SAVE statements are declared outside of begin_save{ , they are used as initializations for all begin_save{ environments. Statements inside the environments take precedence over outside statements and can overwrite them.

A SAVE_MONITOR_ITEM or a SAVE_BE_MONITOR_ITEM statement has to be declared outside of the environment. SAVE_format_skip is redundant, but might still be used. When using begin_save{ environments it is not possible to declare

SAVE_format (i) with i>1 outside of them (which would be very confusing anyway).

```
SAVE_first = 2 ! initialization of begin_save{
SAVE_interval = 4 ! initialization of begin_save{
begin_save{ }
SAVE_choose_meth = 'TIME'
SAVE_format = 'ENSIGHT6 BINARY N--T'
SAVE_first = 0.005 # overwrites initialization values
SAVE_interval = 0.001 # overwrites initialization values
SAVE_type = 'Monitor'
SAVE_type = 'Boundary'
SAVE_file = 'testEnsight'
SAVE_path = 'testEnsight'
SAVE_CoordinateSystem = $MOVE_vconst$
SAVE_ITEM = ( %SAVE_scalar%, [Y %ind_ETA% ], "eta" )
SAVE_ITEM = ( %SAVE_scalar%, [Y %ind_r% ], "density" )
end_save

begin_save{ }
SAVE_choose_meth = 'CONT'
SAVE_format = 'ERFHDF5 N---'
SAVE_first (2) = 15
SAVE_interval (2) = 10
SAVE_file = 'testERF'
SAVE_path = 'testERF'
end_save
```

Not all SAVE formats are fully supported:

- ENSIGHT6 : Fully supported.
- ERFHDF5 : At present, only a single begin_save{ environment with the ERFHDF5 format may be used. It is, however, still
  possible to combine a ERFHDF5 begin_save{ environment with save_environments that use different formats.
  Also not all options of SAVE_type are supported; one may only use 'PointCloud', 'TimeStep', 'None' or the default value. If 'Boundary'
  or 'Monitor' are chosen, SAVE_type is set to default.
- ASCII: Several ASCII environments are possible, but the SAVE_type feature is not supported within this format, except for
  'TimeStep'.

List of members:

SAVE_type	Choose which type of data shall be saved

SAVE_type

Choose which type of data shall be saved

The begin_save{ environment supports the command SAVE_type , which allows the user to control the type of data that will be saved. By default all data is saved, but setting SAVE_type to 'PointCloud', 'Boundary' or 'Monitor' allows for saving only data on the point cloud, boundary elements or monitor points respectively. Setting SAVE_type to 'TimeStep' allows for saving only the .timestep and .timestep.header files.

Combinations of several types are also possible.

By setting SAVE_type to 'None', no boundary, point cloud or monitor data is saved and timestep files are not saved either.

'Boundary' and 'Monitor' are not supported by ERFHDF5; if chosen, the flag for SAVE_type is set to default.

```
begin_save{ }
...
SAVE_type = 'Monitor'
SAVE_type = 'Boundary'
SAVE_type = 'TimeStep'
...
end_save

begin_save{ }
...
SAVE_type = 'PointCloud'
...
end_save
```


3.1.30. Selection

Switch/Case-type selection statement

Allows to use selections depending on aliases. Besides exact matches a default case is supported.

In the simple case selection statements work on scalar aliases.

```
begin_alias{ }
"SelectionAlias" = "ON"
end_alias
...
begin_selection{ "SelectionAlias"}
case{ "OFF"}
...
case{ "ON"}
...
case_else{ }
...
end_selection
```

Remark: The alias used by the selection needs to be defined before the selection statement!

The 'case_else{}' is optional. Within the case blocks all USER_common_variables syntax is allowed. All statements of a valid case block, i.e. the case which matches the current value of the SelectionAlias, are globally visible.

It is also possible to use selections on alias vectors:

```
begin_alias{ }
"SelectionAliasVector" = "ON,OFF,OFF,ON"
end_alias
...
begin_selection{ "SelectionAliasVector"}
case{ "2,OFF"}
...
case{...}
...
case_else{ }
...
end_selection
```

For alias vectors the case statement contains the index (starting from 1) and the value to be checked.

In general, selection statements can be nested up to a certain limit.

An extension of the Selection to mathematical expressions is possible:

```
begin_alias{ }
"SelectionAliasVector" = "-3.1415926"
end_alias
...
begin_selection{ }
case{ [&SelectionAliasVector&>0]}
...
case{ [&SelectionAliasVector&<0]}
...
case_else{ }
...
end_selection
```

This is the so called mathematical-selection, and represents a way to mimic if-elseif-else constructions in the input file. The

begin_selection{ }-clause must not contain any argument.

Soon, the direct implementation of if-elseif-else will follow.

3.1.31. SmoothingLength

define the smoothing length by a set of commands

In MESHFREE , the **smoothing length** is the parameter for the spatial discretization in MESHFREE . For each point within the pointcloud it defines the radius of point interation.
All points within a radius of the local smoothing length around a point are called **neighbors** of the point. The stencils for setting up the discretization are based on these neighbor relations.

Based on the definition of the smoothing length MESHFREE will automatically fill the simulation domain with a pointcloud corresponding to the choice of the smoothing length.

Choosing smaller smoothing length yields finer discretizations. The smoothing length should locally be at maximum a little smaller than the size of the effect that should be resolved - let it be a thin geometry part or a boundary layer.

Strategies for defining the smoothing length

MESHFREE offers different strategies for specifying the discretization - steered by the compulsory parameter USER_h_funct .
Constant smoothing length

CONS : Constant smoothing length provides a constant discretization in the simulation domain. It is specified by

```
USER_h_funct = 'CONS'
```

A constant coarse smoothing length is the preferred mode for the first setup.
Discrete smoothing length

DSCR : variable smoothing length allows user defined refinements on location and physical quantities.

```
USER_h_funct = 'DSCR'
```

Good to know:

- For example, this is useful if you want to refine locally around thin geometry parts. (see SMOOTH_LENGTH)
- If a small smoothing length is attached to a large geometry part, many reference points for the determination of the smoothing length are created on the geometry. If there are too many, then the computation becomes inefficient and will abort if this upper bound is met.

Adaptive smoothing length

ADTV : There are also automatic approaches to adapt the smoothing length to the transient simulation. The idea is to see the smoothing length as function on the pointcloud. The user can assign values to Y %ind_h_adaptive% and the pointcloud is organized with respect to this proposal of the smoothing length, see ADTV for a more detailed description

```
USER_h_funct = 'ADTV'
```

Adaptive plus discrete smoothing length

ADDS allows for combining the two previous approaches:

```
USER_h_funct = 'ADDS'
```

Miscellaneous

Checking the smoothing length

The local smoothing length on the pointcloud can be visualized by saving the index Y %ind_h% :

```
SAVE_ITEM = (%SAVE_scalar%, [Y %ind_h% ], "SmoothingLength")
```

Quality of the smoothing length function

A transition from a fine to a coarse smoothing length should always be smooth and not abrupt - otherwise small effects due to approximation or discretization can build up and lead to instabilities.

List of members:

USER_h_funct	choose either constant, locally variable, or adaptive smoothing length
USER_h_min	minimum allowed smoothing length
USER_h_max	maximum allowed smoothing length
SMOOTH_LENGTH	provide a function of smoothing length

SMOOTH_LENGTH

provide a function of smoothing length

Options for discrete (locally variable) smoothing length definitions:

```
SMOOTH_LENGTH ( $SLflag$ ) = ( %H_constant% , H )
SMOOTH_LENGTH ( $SLflag$ ) = ( %H_spherical% , H_min, L_min, dH/dr, H_max )
SMOOTH_LENGTH ( $SLflag$ ) = ( %H_radial% , H_min, L_min, axis_x, axis_y, axis_z, dH/dr, H_max )
SMOOTH_LENGTH ( $SLflag$ ) = ( %H_linear% , H_min, L_min, normal_x, normal_y, normal_z, H_max )
SMOOTH_LENGTH ( $SLflag$ ) = ( %H_ring% , H_min, L_min, dH/dr, n_x, n_y, n_z, H_max )
```

See also DSCR .

For linking the smoothing length description to the boundary, you need to set the smoothing length tag **$SLflag$** . An example can be found
under SMOOTH_LENGTH .

List of members:

%H_constant%	constant smoothing length or smoothing length given as equation
%H_spherical%	spherical smoothing length distribution around points or geometry elements
%H_linear%	linear smoothing length distribution with respect to a plane
%H_radial%	radial smoothing length distribution with respect to an infinite tube
%H_ring%	annular smooth length distribution with respect to a torus

%H_constant%

constant smoothing length or smoothing length given as equation

Constant smoothing length or smoothing length given by an explicit equation.

```
SMOOTH_LENGTH ( $SLflag$ ) = ( %H_constant% , H, OPTIONAL:weight , OPTIONAL:d(weight)/d(length) )
```

H: smoothing length to be used

weight: The resulting smoothing length will be computed as H_resulting = H*weight. That makes sense if a normalized function exists
which can be used in order to locally refine, for example refinement due to accuracy constraints.

d(weight)/d(length): local change rate of the weight. This has an impact only if working with the original version of UseBoxSystemVersion (=0 or =1).

%H_linear%

linear smoothing length distribution with respect to a plane

Form a plane. On one side, the smoothing length is constant. On the other side, the smoothing length linearly grows.

```
SMOOTH_LENGTH ( $SLflag$ ) = ( %H_linear% , H_min, L_min, normal_x, normal_y, normal_z, H_max )
```

H_min: minimum smoothing length on the given plane

L_min: stripe on top of the plane, where H is kept at the value of H_min

(**normal_x** , **normal_y** , **normal_z**): vector perpendicular to the plane. The norm of the vector gives dH/dr, i.e. the growth rate of H when tending apart from the plane.

H_max: maximum smoothing length

%H_radial%

radial smoothing length distribution with respect to an infinite tube

Form an infinitely long tube of radius **L_min** and construct the smoothing length around the tube.

```
SMOOTH_LENGTH ( $SLflag$ ) = ( %H_radial% , H_min, L_min, axis_x, axis_y, axis_z, dH/dr, H_max )
```

H_min: minimum smoothing length

L_min: radius of the tube

(**axis_x** , **axis_y** , **axis_z**): direction of the tube

dH/dr: growth rate of H outside the tube

H_max: maximum smoothing length

MESHFREE · InputFiles · USER_common_variables · SmoothingLength · SMOOTH_LENGTH · %H_ring%

%H_ring%

annular smooth length distribution with respect to a torus

Form a torus around which the smoothing length is constructed.

```
SMOOTH_LENGTH ( $SLflag$ ) = ( %H_ring% , H_min, L_min, dH/dr, n_x, n_y, n_z, H_max )
```

H_min: minimum H along the ring/torus

L_min: small radius of the torus

dH/dr: increase of smoothing length per distance from torus

(**n_x** , **n_y** , **n_z**): vector perpendicular to the plane in which the torus is placed. The length of this vector forms the big radius of the torus.

H_max: maximal accepted smoothing length

MESHFREE · InputFiles · USER_common_variables · SmoothingLength · SMOOTH_LENGTH · %H_spherical%

%H_spherical%

spherical smoothing length distribution around points or geometry elements

Form a ball of radius **L_min** and construct the smoothing length around it.

```
SMOOTH_LENGTH ( $SLflag$ ) = ( %H_spherical% , H_min, L_min, dH/dr, H_max )
```

H_min: minimum smoothing length

L_min: radius of "ball" within which the smoothing length is kept on the level of H_min

dH/dr: increase rate of H outside of the L_min-ball with respect to the (Euclidean) distance (based on unit lengths)

H_max: maximum smoothing length

USER_h_funct

choose either constant, locally variable, or adaptive smoothing length

Currently implemented:

- USER_h_funct = 'CONS' (constant, see CONS)
- USER_h_funct = 'DSCR' (discrete, see DSCR)
- USER_h_funct = 'ADTV' (adaptive, see ADTV)
- USER_h_funct = 'ADDS' (adaptive + discrete, see ADDS)

List of members:

CONS	constant smoothing length defintion
DSCR	discrete (locally variable) smoothing length definition
ADTV	adaptive smoothing length definition
ADDS	adaptive + discrete smoothing length definition

ADDS

adaptive + discrete smoothing length definition

Experimental coupling of ADTV and DSCR : In each time step the minimum of the proposed smoothing length %ind_h_adaptive% (ADTV) and the proposed
discrete smoothing length (DSCR) is used as the smoothing length.

```
USER_h_funct = 'ADDS'
USER_h_min = RealNumber
USER_h_max = anotherRealNumber
CODI_eq ( $Material$ ,%ind_h_adaptive%) = [ ... some equation ... ]
SMOOTH_LENGTH ( $SLflag$ ) = (%H_BuiltInFunction%, ... )
...
INITDATA ( $Material$ ,%ind_h_adaptive%) = [ ... some equation ... ]
```

Analogously to ADTV , the proposed smoothing length value for the ADTV -part is written into the index %ind_h_adaptive% for each point.
The standard discrete smoothing length definitions can be used (see SMOOTH_LENGTH).

See also Equations and CODI .

This feature is helpful to construct a problem-specific initial smoothing length distribution.

ADTV

adaptive smoothing length definition

Current experimental development is the adaptive smoothing length:

```
USER_h_funct = 'ADTV'
USER_h_min = RealNumber
USER_h_max = anotherRealNumber
```

The idea here is to write a proposed smoothing length value for each point into the index %ind_h_adaptive% :

```
CODI_eq ( $Material$ ,%ind_h_adaptive%) = [ ... some equation ... ]
```

The following rules apply:

1.) This equation is evaluated at the end of each time step.
2.) At the beginning of the next time step, these values are copied to %ind_h% , and thus taken as the smoothing length distribution for the new time step.
Warning: The new %ind_h% -values are not undertaken any further checking of consistency, currently, that explicitely means:
3.) The user has to carefully verify the smoothing length distribution for the next time step. One way to go is given in the example below.
4.) The method currently has one drawback: as the adaptive h-values are determined at the END of the time step, there is no way
of defining the INITIAL h-distribution.
Current assumption: h_Initial = USER_h_max
A problem-specific initial smoothing length definition is possible by using ADDS (adaptive + discrete).

Example:

```
begin_alias{ }
"H_min" = "0.1"
"H_max" = "0.5"
"HchangePerTimeStep" = "0.1"
"SpeedOfBox" = "4.0"
"dH_over_dr" = "0.15"
end_alias
USER_h_funct = 'ADTV'
USER_h_min = &H_min&
USER_h_max = &H_max&
CODI_eq ( $Mat1$ ,%indU_absgradV%) = [ sqrt(dYdx( %ind_v(1)% )^2+dYdy( %ind_v(1)% )^2+dYdx( %ind_v(2)% )^2+dYdy( %ind_v(2)% )^2)* &H_max& / &SpeedOfBox& ] # some measure of gradient of velocity
CODI_eq ( $Mat1$ ,%indU_h_1stguess%) = [max( &H_max& *(1-Y%indU_absgradV%) , &H_min& )] # set a definition of adaptive smoothing length
CODI_eq ( $Mat1$ ,%indU_h_smooth%) = [max( min( Y%indU_h_1stguess% , (1+ &HchangePerTimeStep& )*Y %ind_h% ) , (1- &HchangePerTimeStep& )*Y %ind_h% ) ] # make sure H varies not more than a given threshold from time step to time setp
CODI_min_max ( $Mat1$ ,%indU_h_smooth%) = (-10000,10000, &dH_over_dr& ) # restrict local slope of the adtipte smoothinge length function
CODI_eq ( $Mat1$ ,%ind_h_adaptive%) = [ Y%indU_h_smooth% ] # copy the constructed function to %ind_h_adaptive%
```

See also Equations and CODI .

CONS

constant smoothing length defintion

For constant smoothing length choose:

```
USER_h_funct = 'CONS'
USER_h_min = RealNumber
USER_h_max = sameRealNumber
```

MESHFREE · InputFiles · USER_common_variables · SmoothingLength · USER_h_funct · DSCR

DSCR

discrete (locally variable) smoothing length definition

For locally variable smoothing length choose:

```
USER_h_funct = 'DSCR'
SMOOTH_LENGTH ( $SLflag$ ) = (%H_BuiltInFunction%, ... )
USER_h_min = RealNumber
USER_h_max = anotherRealNumber
```

For the different options for **%H_BuiltInFunction%** see SMOOTH_LENGTH .

MESHFREE · InputFiles · USER_common_variables · SmoothingLength · USER_h_max

USER_h_max

maximum allowed smoothing length

```
USER_h_max = RealNumber
```

MESHFREE · InputFiles · USER_common_variables · SmoothingLength · USER_h_min

USER_h_min

minimum allowed smoothing length

```
USER_h_min = RealNumber
```

MESHFREE · InputFiles · USER_common_variables · TimeControl

3.1.32. TimeControl

time control options

The possible commands for initial time, final time, and time step control are described below.

As Meshfree performs a transient simulation, the simulation time interval **must** be specified in any setting.

```
Tstart = 0 #Simulation running from t=0 seconds
Tend = 21 # ... to t=21 seconds
```

Optionally, the simulation can be performed for a maximum of TimeIntegration_N_final timesteps. The simulation will stop if either TimeIntegration_N_final timesteps have been performed or the simulation time has reached Tend .

Non-adaptive Timestep size

The Ucv-parameter

```
DELT_dt_variable = 0 (default)
```

indicates that the timestep size does not automatically adapt to the flow characteristics. (e.g. CFL-conditions)
Meshfree steadily increases the timestep size from DELT_dt_start until DELT_dt is reached. The cv-parameter time_step_gain limits the change rate of the timestep size. If DELT_dt is smaller than DELT_dt_start , then the timestep size is constant DELT_dt_start .

Adaptive timestep size (recommended)

The Ucv-parameter

```
DELT_dt_variable = 1
```

indicates that the timestep size automatically adapts to the flow characteristics in the simulation such that CFL conditions are met, see parameter COEFF_dt .
Also here, the cv-parameters time_step_loss and time_step_gain limit the change rate of the timestep size.
Additional time timestep size criterions can be defined per material with the parameter DELT_dt_AddCond .

Good to know:

- Apart from DELT_dt_AddCond , all parameters are read exactly once at the beginning of the simulation and can thus only contain scalar values (not equations!)
- The local proposed timestep size is calculated per point and is available in the index %ind_dt_local% .

List of members:

Tstart	(compulsory) initial time of a simulation
Tend	(compulsory) maximum final time of a simulation
TimeIntegration_N_final	(optional) final time step of a simulation
DELT_dt	(compulsory) maximum allowed time step size
DELT_dt_start	(compulsory) time step size at the start of a simulation
DELT_dt_variable	(optional) let MESHFREE control the time step size
DELT_dt_AddCond	(optional) defines a custom time step criterion

MESHFREE · InputFiles · USER_common_variables · TimeControl · DELT_dt

DELT_dt

(compulsory) maximum allowed time step size

This value is compulsory. If not given, MESHFREE will stop.

```
DELT_dt = 1.0e-2
```

See DELT_dt_variable for further details.

MESHFREE · InputFiles · USER_common_variables · TimeControl · DELT_dt_AddCond

DELT_dt_AddCond

(optional) defines a custom time step criterion

```
DELT_dt_AddCond ( $MATERIAL$ ) = RHS
```

If defined, MESHFREE will evaluate the given RightHandSideExpression at the start of each timestep and respect this value as an additional criterion for the maximum timestep size of the material with the specified tag.

Good to know:

- It is only possible to define DELT_dt_AddCond once per material, hence for incorporationg multiple conditions, these must be included into the RHS of the equation.

DELT_dt_start

(compulsory) time step size at the start of a simulation

This value is compulsory. If not given, MESHFREE will stop.

```
DELT_dt_start = 1.0e-2
```

To avoid instabilities, its value has to be adapted to the chosen point cloud resolution and relevant velocity.

This value is also used in the first time cycle after restart.

Note: If DELT_dt_start is set to a negative number, then at restart the simulation is continued with the same time step size as at the time the restart file was written.

DELT_dt_variable

(optional) let MESHFREE control the time step size

```
DELT_dt_variable = 1
```

default: DELT_dt_variable = 0

If DELT_dt_variable == 1, MESHFREE controls the time step size by itself but does not exceed DELT_dt (adaptive time stepping).

If DELT_dt_variable == 0, MESHFREE steadily increases the time step size from DELT_dt_start until DELT_dt is reached.

Tend

(compulsory) maximum final time of a simulation

This value is compulsory. If not given, MESHFREE will stop.

```
Tend = 1
```

A simulation will stop if either TimeIntegration_N_final or Tend is reached.

TimeIntegration_N_final

(optional) final time step of a simulation

This value is optional. If set, the simulation stops after the specified number of time steps.

```
TimeIntegration_N_final = 1000
```

A simulation will stop if either TimeIntegration_N_final or Tend is reached.

MESHFREE · InputFiles · USER_common_variables · TimeControl · Tstart

Tstart

(compulsory) initial time of a simulation

This value is compulsory. If not given, MESHFREE will stop.

```
Tstart = 0
```

MESHFREE · InputFiles · USER_common_variables · __DEFAULT_configuration_file__

3.1.33. __DEFAULT_configuration_file__

allows to provide Ucv_DEFAULT.dat as a generalistic/default definition

The default file allows to define default setting for groups/portions of geometry-items, fulfilling a naming convention.
With this, MESHFREE is ready to only be provided a geometry file, and start a simulation without any further input definition.

The default definition file has a unique name: "Ucv_DEFAULT.dat". The general rules to bind it in are:

- if, in the current project folder, there is a file with the name "Ucv_DEFAULT.dat", then this file is read-in first, before USER_common_variables.dat is read in
- if the environment variable MESHFREE_USE_DEFAULT_FILE=true, the program will use the Ucv_DEFAULT.dat . In this case,
  - EITHER the environment variable MESHFREE_Ucv_DEFAULT is set, then it points to the Ucv_DEFAULT-file to be used (i.e. the users have the chance to use their general default configuration,
  - OR the program will automatically generate a Ucv_DEFAULT.dat in the hope, it will cover the needs of the current appliocation.

In Ucv_DEFAULT.dat, one is free to pre-define anything. Most useful it is to define the "_DEFAULT" alias names.
The definition of an alias with the suffix "_DEFAULT" is a recognized as a default definition for a certain group of geometry.
For example:

```
begin_alias{ }
"wall_DEFAULT" = " BC$BC_wall_DEFAULT$ ACTIVE$InitAlways_DEAFULT$ IDENT%BND_slip%
MAT&mat1_DEFAULT& TOUCH%TOUCH_always% MOVE$MOVE_DEFAULT$ LAYER0 CHAMBER1 "
"bot*_DEFAULT" = " &wall_DEFAULT& "
"in*_DEFAULT" = " BC$BC_in_DEFAULT$ ACTIVE$InitAlways_DEAFULT$ IDENT%BND_outflow%
MAT&mat1_DEFAULT& TOUCH%TOUCH_always% MOVE$MOVE_DEFAULT$ LAYER0 CHAMBER1
POSTPROCESS$PP_in_DEFAULT$ "
"out*_DEFAULT" = " BC$BC_out_DEFAULT$ ACTIVE$InitAlways_DEAFULT$ IDENT%BND_outflow%
MAT&mat1_DEFAULT& TOUCH%TOUCH_always% MOVE$MOVE_DEFAULT$ LAYER0 CHAMBER1
POSTPROCESS$PP_out_DEFAULT$ "
"top*_DEFAULT" = " &wall_DEFAULT& "
"front*_DEFAULT" = " &wall_DEFAULT& "
"back*_DEFAULT" = " &wall_DEFAULT& "
end_alias
```

For example, the alias-definition "in*_DEFAULT" matches for all geometry items, starting with "in", such as "inflow"

Please also refer to AliasForGeometryItems .

See the comprehensive example and have a special look into Ucv_DEFAULT.dat .
See the classical USER_common_variables.dat, where the user only has to provide the geometry file. If the Ucv_DEFAULT is general enought, no additional information is given and the simulation can be started immediately.

DOWNLOAD COMPREHENSIVE EXAMPLE

3.1.34. __GeneralRemarks__

general remarks upon the syntax within UCV files

The USER_common_variables file utilizes its own scripting syntax and this page serves as overview over the syntax in USER_common_variables.dat (UCV).

Warning: First of all, the scripting language is **case sensitive** .

There are three major concepts involved: variables, assignments (in order to assign boundary conditions), and environments (for defining things that naturally do not fit into one line).

Variables

There are four types of variables that can be referenced within the UCV files:

- **&AliasVariableName&** references an **alias variable** defined by the user as string in the alias section, see ALIAS , or in the construct section, see ConstructClause .
- **$AcronymVariableName$** refers to an **acronym** or **soft variable** ; MESHFREE automatically assigns consecutive integer values to the $...$-variables in the order they are appearing within the UCV.
- **%MESHFREEVariableName%** refers to **MESHFREE internal variables** such as the index variables (see Indices) and constants (see __Constants__). Generally, the user cannot define these variables (the only exception is UserDefinedIndices).
- **@SystemVariable@** represents **system** or **software information** .

More information in Variables .

Assignments

Assignments in the UCV can take the following forms. The number of arguments depends on the LHS statement.
LHS = RHS: left hand side with no argument

The assignment **LHS = RHS** (left hand side with no argument) can have the two following meanings:

- A value is assigned to a parameter, e.g. the end time for the simulation shall be 10 seconds:

```
Tend = 10.0 # set parameter Tend to 10.0 seconds
```

- A new item of LHS is added and an implicit enumeration takes place, e.g. SAVE_ITEM . For example, the code snippet

```
SAVE_ITEM = RHS1 # add a save item for RHS1
SAVE_ITEM = RHS2 # add a save item for RHS2
```

  adds two SAVE_ITEMs, one for RHS1 and one for RHS2.

LHS(arg) = RHS: left hand side with one argument

In the assignment **LHS(arg) = RHS** , the right hand side is assigned to the argument regarding the LHS, e.g.

- PhysicalProperties : the density of the material referenced by acronym $WATER$ is 1000.00:

```
density( $WATER$ ) = 1000.00
```

With that, also the acronym variable $WATER$ is automatically initialized and can be referenced in the alias section with the MAT tag.

- BoundaryConditions : a boundary condition for the temperature defined for the acronym BC_wall :

```
BC_T ( $BC_wall$ ) = RHS
```

With that, also the acronym variable BC_wall is automatically initialized and can be referenced in the alias section with the BC tag.

LHS(arg1,arg2) = RHS: left hand side with two arguments

In the assignment **LHS(arg1,arg2) = RHS** , the right hand side is assigned to the two arguments regarding the LHS, e.g.

- Initial conditions for a quantity, e.g. the initial temperature (referenced by internal variable %ind_T%) in the simulation of the material (referenced with acronym $WATER$) is 310.8 Kelvin:

```
INITDATA ( $WATER$ ,%ind_T%) = 310.8
```

- see the documentation of CODI for more examples of two arguments on the LHS.

So far, we have not tackled the **RHS** , for this please refer to RightHandSideExpression .

Environments

In the UCV syntax, there are also environments to provide certain functionalities. An environment starts with begin_environment{"nameOfEnvironment"}, ends with end_environment and can be referred to by the name "nameOfEnvironment". Here are some examples for environments:

- ALIAS :

```
begin_alias{ "optionalName"}
"alias1" = " String to replace &alias1& "
end_alias
```

- BoundaryElements :

```
begin_boundary_elements{ "optionalName"}
include{ ...
end_boundary_elements
```

- Equations: The equation requires a name in order to be referenced.

```
begin_equation{ "nameOfEquation"}
some equation ...
end_equation
```

Execution control for statements

- Selection : Execution of statements based on a condition, decision by the value of an alias variable which statements to execute in a UCV file. (Similar to If-else)
- Loops : Repetition of statements, N-times repetition of statements with an iterator variable.

Options for structuring UCVs

Sometimes UCV files can get very complex and the individual lines get very long. Here are some tools for structuring.

- include_Ucv{ : includes the specified file into the UCV File.
- ContinuationLines : for line breaking of long statements.

List of members:

Variables	variables used in the USER_common_variables input file
ContinuationLines	break long lines into shorter ones in order to have more readable input files
RightHandSideExpression	syntax for right hand side expressions in USER_common_variables

ContinuationLines

break long lines into shorter ones in order to have more readable input files

Long lines can be split into shorter ones, if that improves readability of the input file. The token "..." at the end of the line (BUT BEFORE THE COMMENTS!!!!) tells
the file reader that the next line in file still belongs to the present line.

Example: The DropletSource can be written in one-line form

```
DropletSource (1) = ( 0.05, [(1.7* &Hmin& )^3], [4.5+rand(1)*(1.7+0.3)], [-0.2+rand(1)*(0.4+0.3)], [0+rand(1)*(1+0.3)], 1, $Mat1$ )
```

The same in multiple-line form, one can easily add remarks to each of the items in the brackets

```
DropletSource (1) = ( 0.05, ... # how much droplet volume per time is to be created
[(1.7* &Hmin& )^3], ... # droplet size to be created
[4.5+rand(1)*(1.7+0.3)], ... # x-position (center) of the new droplet
[-0.2+rand(1)*(0.4+0.3)], ... # y-position (center) of the new droplet
[0+rand(1)*(1+0.3)], ... # z-position (center) of the new droplet
1, ... # put the new droplet in this chamber
$Mat1$ ... # new droplet to obtain this material flag
)
```


RightHandSideExpression

syntax for right hand side expressions in USER_common_variables

Right hand side expressions are all expressions on the right of the "="-sign.

For example, an expression in USER_common_variables could look like this:

```
BC_v ($...$) = (Expression0, Expression1, Expression2, ... )
```

Each of the expressions, separated by comma, can be of three different types.

1.) Arithmetic expression in-between []-brackets: [... Y%ind_...% ...]

Example:

```
BC_v ($...$) = ( ... , [ ... Y%ind_...% ... ], ... )
```

2.) Link to an existing equation: equn{$EqnName$}

Example:

```
BC_v ( $...$ ) = ( ... , equn{ $EqnName$ }, ... )
```

In this case, the equation needs to be defined somewhere in the input file:

```
begin_equation{ $EqnName$ }
BodyOfEquation
end_equation
```

3.) Link to an existing curve: curve{$CrvName$}depvar{%ind_Var%}

Example:

```
BC_v ( $...$ ) = ( ... , curve{ $CrvName$ }, ... )
```

In this case, the curve must be defined somewhere in the input file:

```
begin_curve{ $CrvName$ }, depvar_default{ %ind_Var%}
BodyOfCurve
end_curve
```

%ind_Var% defines the quantity/entity the left column of the curve is representing (independent variable).
See also 1D_Curves .

Variables

variables used in the USER_common_variables input file

There are currently four types of variables that the user may use in the USER_common_variables file:

- **&AliasVariableName&** references an alias variable, to be defined in the alias section (pure string replacement definitions), see ALIAS and ConstructClause
- **$AcronymVariableName$** refers to an acronym; MESHFREE assigns consecutive integer values to the $...$-varibales given by the user
- **%MFvariableName%** refers to a variable predefined by MESHFREE , also representing integer values; among others, the index variables (see Indices) and the constant (see __Constants__) are of this type. The user cannot define these variables, with the exception of UserDefinedIndices .
- **@SYSTEMvariable@** contain system or software information

Variable Types

Alias Variable: &AliasVariableName&

Alias variables are defined by the user in the alias section of the USER_common_variables file. The values of these variables are strings. At any position where the variable is referenced by **&AliasVariableName&** , the string is placed.

Example 1: Define the scaler alias variable v_inflow to be "10.0".

```
begin_alias{ "ModelParameter"} #giving an intuitive name - no further meaning
"v_inflow" = "10.0" #defines the alias variable
begin_alias{ "ModelParameter"}
```

This definition can be used, for example in a boundary condition:

```
BC_v ( $inflow$ ) = ( %BND_inflow% , &v_inflow& )
```

&v_inflow& is then string-replaced with the definition "10.0" and becomes:

```
BC_v ( $inflow$ ) = ( %BND_inflow% , 10.0 )
```

Example 2: Define the vectorial alias variable Class and use it to define different geometry parts (see

AliasForGeometryItems).

```
begin_alias{ }
"Class" = "inflow, wall, outflow" # definition of geometry class
...
"&Class(1)&" = " BC$BC_in$ ..." # definition of inflow alias
"&Class(2)&" = " BC$BC_wall$ ..." # definition of wall alias
"&Class(3)&" = " BC$BC_out$ ..." # definition of outflow alias
end_alias
```

Good to know:

- The alias definition plays a central role in connecting the definition of model parameters to boundary elements: see AliasForGeometryItems .
- The alias definition can contain nested statements, in particular, an alias definition can contain a reference to another alias variable. It is important that these definitions can be uniquely resolved.
- Execution control for statements in the USER_common_variables can be done based on the value of an alias variable, see Selection .
- The usage of wildcards in the name of the alias variable is also possible in AliasForGeometryItems .

Acronym Variable: $AcronymVariableName$

Acronym variables (or soft variables) are defined by the user by using them in a left hand side expression. They can then be referred to by **$AcronymVariableName$** . Internally, in MESHFREE they are handled as integers, but for the user their actual value is not of importance as these variables are used as labels.

Example 3: Defines an integration to determine the total mass. The soft variable $MassTotal$ is also automatically initialized then.

```
INTEGRATION ( $MassTotal$ ) = ( %INTEGRATION_INT% , [Y %ind_r% ], $MatUSER$ , %INTEGRATION_Header%,
"Total Mass")
```

If one now wants to use the integration in another place, e.g. an equation, then it can be referred to using the soft variable $MassTotal$:

```
... [ ... integ( $MassTotal$ ) ... ] ...
```

MESHFREE Internal Variable: %MFvariableName%

MESHFREE internal variables are predefined in MESHFREE , also internally stored as integer values. These are

- the index variables, see Indices
- the constants, see __Constants__
- and the UserDefinedIndices (the user can steer what will be stored in these Indices)

Example 4: In an equation accessing the attribute density of a point in an equation, by using the index %ind_r% :

```
... [ ... Y %ind_r% ... ] ...
```

System Variable: @SYSTEMvariable@

A system variable contains system or software information. Currently, the following features are implemented:

- @VERSION@ - returns a string with the version number of MESHFREE
- @DATE@ - returns a string with the date at MESHFREE startup in the form YYYY.MM.DD
- @TIME@ - returns a string with the time at MESHFREE startup in the form HH:MM:SS
- @CLPARAM@ - returns the string passed via the CommandLine option --clparam or -clp
- @ENV(NameOfEnvironmentVariable)@ - returns the value of the environment variable with the given name
- @CV(cv_variable)@ - returns the status of a variable from common_variables
- @[equation_strng]@ - evaluates the given equation, see Equations

Example 5: if USER is the environment variable for the user, then one could incorporate system information in the following way in the save path SAVE_path in the following way:

```
SAVE_path =
'results___version=@VERSION@___user=@ENV(USER)@___MPI=@[real(%MPI_NbProcesses%)]@_OMP=
@[real(%OMP_NbProcesses%)]@___'
# save path containing the user name,
# the MESHFREE -version
# the number of MPI and OMP processes
```

Logging

At the startup of MESHFREE the hidden log folder *.FPM_log_FPM ID=ID_of_run* is created and information on the values assigned to the variables is stored in the following files therein:

- *List_of_Aliases.log* : contains the alias section. As nested definitions of alias section are also possible, this files contains the completely resolved definitions.
- *List_of_Acronyms.log* : the integer values for acronyms, ordered by usage: BCON , MOVE , MAT , SMOO (SmoothingLength), POSTBND (PostProcessing), ACTIVE , TOUCH , EQUN and CURV.
- *List_of_indices.log* : Contains all indices that are referencing to entries of the Y-array. Some of these indices might be sharing an integer value if they belong to different solvers. This is due to memory reasons.
- *List_of_FPMvariables.log* : Contains all identifiers of the form %...% (indices, constants and others). Useful if one wants, for example, decode the integer value Y %ind_kob% to a boundary flag like %BND_none% (inner Point), %BND_wall% (wall), %BND_free% (free surface), ...

3.1.35. __Parameters__

CV-parameters that can also be set in UCV

This page is under development. The list of parameters will be completed gradually.

Note:

- Some CV-parameters (see common_variables) can also be set in USER_common_variables (UCV). The UCV-definition is dominant and overwrites the
  CV-definition (see warnings file in the simulation folder).
- Some of these parameters can be set chamberwise, which can be necessary for multi-phase simulations. If such a parameter is
  not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

List of members:

BEmap_DefaultValue	Default value of BE_MAP (UCV)
BUBBLE_DoTheManagement	(chamberwise) switch regarding bubble analysis (UCV)
BUBBLE_EnforceAveragePressure	fix average pressure for all bubbles (UCV)
BUBBLE_pOffset	define offset pressure for bubble pressure-on-volume analysis (UCV)
COEFF_dt	(chamberwise) factor for computation of time step size (UCV)
COEFF_dt_coll	time step criterion from interaction model (DROPLETPHASE only) (UCV)
COEFF_dt_d30	time step criterion depending on %ind_d30% (DROPLETPHASE only) (UCV)
COEFF_dt_Darcy	define the virtual time step size for applications with Darcy (Brinkman) term (UCV)
COEFF_dt_free	(experimental) factor for exaggerated movement of the free surface (UCV)

COEFF_dt_SurfaceTension_A	time step criterion for surface tension, parameter A (UCV)
COEFF_dt_SurfaceTension_B	time step criterion for surface tension, parameter B (UCV)
COEFF_dt_SurfaceTension_C	(experimental) time step criterion for surface tension, parameter C (UCV)
COEFF_dt_virt	(chamberwise) scaling factor for the virtual time step size (UCV)
COEFF_mue	scaling factor for numerical viscosity (UCV)
COMP_CosEdgeAngle	(chamberwise) parameter to identify edges in geometry (UCV)
COMP_DoOrganizeOnlyAfterHowManyCycles	do the point cloud organization only after how many time cycles (UCV)
COMP_DropletphaseSubcycles	switch for DROPLETPHASE subcycling (UCV)
COMP_DropletphaseWithDisturbance	disturbance for DROPLETPHASE (UCV)
COMP_dt_indep	parameter to switch on independent time stepping for two-phase LIQUID simulations with v-- and vp- (UCV)
COMP_facSmooth_Eta	parameter for weight kernel definition for smoothing of viscosity (UCV)
COMP_nbSmooth_Eta	number of smoothing cycles for effective and total viscosity (UCV)
COMP_RemeshBoundary	parameter to control remeshing of IGES-files (UCV)
COMP_TypeSmooth_Eta	type for smoothing of viscosity (UCV)
COMP_TypeSmooth_Rho	type for smoothing of density (UCV)
compute_FS	(chamberwise) switch to compute free surfaces (UCV)
compute_phase_boundary	(obsolete) invoke detection of interface connections (UCV)
CONTROL_StopAfterReadingGeometry	stops the MESHFREE program after geometry is read (UCV)
damping_p_corr	(chamberwise) parameter to reduce the dynamic pressure as initial guess for the next time level (UCV)
DIFFOP_ConsistentGradient	consistent gradient in the sense d/dn = n*grad (UCV)
DIFFOP_kernel_Gradient	(chamberwise) factor for the weight kernel for the least squares approximation stencils for gradients (UCV)
DIFFOP_kernel_Laplace	(chamberwise) factor for the weight kernel for the least squares approximation stencils for the Laplacian (UCV)
DIFFOP_kernel_Neumann	(chamberwise) factor for the weight kernel for the least squares approximation stencils for Neumann operators (UCV)
DIFFOP_kernel_Transport	(chamberwise) factor for the weight kernel for the least squares approximation stencils for the transport operators (UCV)

DIFFOP_laplace	type of least squares approximation stencils for the Laplacian (UCV)
DIFFOP_Neumann_ExcludeBND	(chamberwise) parameter to exclude boundary points from the neighborhood for the computation of the Neumann operators (UCV)
DIFFOP_WeightReductionInCaseOfDeactivation	(chamberwise) parameter to reduce the weight of a neighbor point in case of deactivation (UCV)
DP_UseOnlyRepulsiveContactForce	switch regarding attractive forces in spring-damper model (UCV)
eps_p	precision in the breaking criterion for the linear systems of pressure (UCV)
eps_phyd	precision in the breaking criterion for the linear systems of hydrostatic pressure (UCV)
eps_T	precision in the breaking criterion for the linear systems of temperature (UCV)
eps_v	precision in the breaking criterion for the linear systems of velocity (UCV)
FLIQUID_ConsistentPressure_Version	version how to compute the consistent pressure (UCV)
FOFTLIQUID_AdditionalCorrectionLoops	additional velocity correction loops (UCV)
IGES_Accuracy	relative accuracy for consistency checks of IGES-faces (UCV)
IGES_HealCorruptFaces	allow a certain depth of healing triangulation of IGES faces by refinement (UCV)
LINEQN_scaling	choose the way how to scale/normalize the linear systems (UCV)
LINEQN_solver	linear solver to be used for the coupled vp- or v-- system (UCV)
LINEQN_solver_ScalarSystems	linear solver to be used for the scalar systems like pressure, temperature, etc. (UCV)
max_N_stencil	maximum number of neighbor points accepted for stencil computation and numerics (UCV)
MEMORIZE_ResetReadFlag	reset frequency for MEMORIZE_Read flag (UCV)
ord_eval	define approximation order for refill points (UCV)
ord_gradient	(chamberwise) approximation order of the gradient operators (UCV)
ord_laplace	define approximation order of the Laplace operators (UCV)
PHASE_distinction	invoke detection of interface connections (UCV)
PointDsplMethod	(experimental) Choice among different ways to move points in Lagrangian framework (UCV)
radius_hole	relative allowed hole size (UCV)
rel_dist_bound	relative distance of neighboring points at boundaries for initial filling (UCV)
RepairGeometry	enforce clustering of geometry nodes upon read-in (UCV)
RepresentativeMass_iData	(chamberwise) parameter for the RepresentativeMass algorithm (UCV)
restartnewBE_filling	(chamberwise) parameter to control filling of new boundary elements upon restart (UCV)

SAMG_Setupreuse	accelerates SAMG solver for quasi-stationary point clouds (UCV)
SAVE_atEndOfTimestep	choose to save data for visualization at the end of time steps instead of at the start (UCV)
SAVE_PrecisionTimestepFile	choose the precision (number of digits) for values in the timestep file (UCV)
SCAN_ClustersOfConnectivity	(chamberwise) switch on cluster checking of MESHFREE point cloud by neighborhood connectivity (UCV)
STRESSTENSOR_Variante	version of stress tensor time integration (UCV)
STRESSTENSOR_Variante_Factor	factor in stress tensor time integration wrt the shear modulus (UCV)
V00_SmoothDivV	Chorin projection: smooth the local values of div(v) before going into the correction pressure computation (UCV)
VOLUME_correction	(chamberwise) parameter to correct volume by GLOBALLY adjusting the divergence of velocity term (UCV)
VOLUME_correction_FreeSurface	(chamberwise) parameter to correct volume by tiny global lifting of the free surface (UCV)
VOLUME_correction_local	(chamberwise) parameter to correct volume by LOCALLY adjusting the divergence of velocity term due to representative mass balance (UCV)
VP0_VelocityCorrection	(chamberwise) switch to compute free surfaces (UCV)

BEmap_DefaultValue

Default value of BE_MAP (UCV)

```
BEmap_DefaultValue = 0.0
```

Default: BEmap_DefaultValue = -888888.0

Defines the value which is returned whenever BE_MAP () does not find any points close to the BE centroid or all points have been filtered out.

BUBBLE_DoTheManagement

(chamberwise) switch regarding bubble analysis (UCV)

```
BUBBLE_DoTheManagement = 1
```

Default: BUBBLE_DoTheManagement = 0
Allowed values: BUBBLE_DoTheManagement = 0, 1, 2 (see BubbleAlgorithm)

OPTIONAL SECOND DIGIT: switch off bubble consistency checks

```
BUBBLE_DoTheManagement = 1 1
```

Default value = 0

If put to 1, then consistency checks for bubbles, concerning their re-configuration, are switched off. For example, one of these checks is:
If a new bubble forms out of two old bubbles, then the new bubble is invalid, if one of the old bubbles is invalid (see BubbleVolume).

OPTIONAL THIRD DIGIT: switch off implicit pressure computation

```
BUBBLE_DoTheManagement = 11 1
```

Default value = 0
If put to 1, implicit computation of bubble pressure is switched off, see BubbleAlgorithm (BubbleImplicitPressure and BubbleSemiimplicitPressure).

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER), that is

```
BUBBLE_DoTheManagement (i) = 1 # i is the chamber index
```

If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · BUBBLE_EnforceAveragePressure

BUBBLE_EnforceAveragePressure

fix average pressure for all bubbles (UCV)

```
BUBBLE_EnforceAveragePressure = 1.0e5 # atmospheric pressure
```

In a closed computational domain with fixed amount of gas and air (for example tank half full with liquid),
it makes sense to fix the average pressure of the bubbles as a whole. I.e., for all times, we require

$$\sum_{i}^{N_{\text{bubbles}}} p^{i}_{\text{bubble},tot}(t) V^{i}(t) = p_{\text{average}} V_{\text{total}}(t)$$

If a positive number is given, all bubbles' pressure values are corrected by a constant value such that the average pressure constraint is satisfied.

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · BUBBLE_pOffset

BUBBLE_pOffset

define offset pressure for bubble pressure-on-volume analysis (UCV)

```
BUBBLE_pOffset = 1.0e5 # atmospheric pressure
```

The bubble´s pressure-volume-law is

$$p^{\kappa}_{\text{bubble},tot}(t) \cdot V_{\text{bubble}}(t) = \text{const.}$$

based on the bubbles total interior pressure.

With the pressure offset, we are able to work with any reference pressure, using the pressure offset to map the pressure analysis to the correct total pressure.

$$\left(p_{\text{bubble}}(t) + p_{\text{offset}}\right)^{\kappa} \cdot V_{\text{bubble}}(t) = \text{const.}$$

COEFF_dt_Darcy

define the virtual time step size for applications with Darcy (Brinkman) term (UCV)

```
COEFF_dt_Darcy = 0.1
```

Default: COEFF_dt_Darcy = 1.0

The virtual time step size for the correction pressure computation in case of a Darcy term is present, is computed as

$$\Delta t_\beta = \mathrm{COEFFdtDarcy} \cdot \Delta t$$

See v-- and vp- for details, especially look for Δt_β .

Note: Actually, it makes sense to choose this value < 1 only in the case of vp- . In the other cases, it will most probably lead to fluctuating numerical solutions for the dynamic pressure.

COEFF_dt_SurfaceTension_A

time step criterion for surface tension, parameter A (UCV)

```
COEFF_dt_SurfaceTension_A = 1.0
```

Default: COEFF_dt_SurfaceTension_A = 0.5

The whole time step criterion is derived in DOCUMATH_TimeStepCriterionSurfaceTension.pdf , the present parameter represents the parameter A within this document.

COEFF_dt_SurfaceTension_B

time step criterion for surface tension, parameter B (UCV)

```
COEFF_dt_SurfaceTension_B = 1.0
```

Default: COEFF_dt_SurfaceTension_B = 0.5

The whole time step criterion is derived in DOCUMATH_TimeStepCriterionSurfaceTension.pdf , the present parameter represents the parameter B within this document.

COEFF_dt_SurfaceTension_C

(experimental) time step criterion for surface tension, parameter C (UCV)

```
COEFF_dt_SurfaceTension_C = 20.0
```

Default: COEFF_dt_SurfaceTension_C = 10.0

The whole time step criterion is derived in DOCUMATH_TimeStepCriterionSurfaceTension.pdf ,
the present parameter represents the parameter B within this document.

Warning: This parameter was introduced during the development of the free surface functionality of MESHFREE . It seems to be obsolete, as it should be given
automatically by the construction of the differential operators. Use this parameter only for testing.

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · COEFF_dt

COEFF_dt

(chamberwise) factor for computation of time step size (UCV)

```
COEFF_dt = 0.1
```

Default: COEFF_dt = 0.2

In MESHFREE , each point computes his own local, temporal time step size by

$$dt_i^{\text{local}} = \text{COEFFdt} \cdot \min\left(\frac{h_i}{\|\boldsymbol{v}_i\|}, \sqrt{\frac{h_i}{\|\boldsymbol{g}_i\|}}, \sqrt{\frac{1}{\sigma_i \cdot \nu} \cdot \rho h_i^3} \right)$$

The first term is the typical CFL condition (MESHFREE point shall not move more than $\text{COEFFdt} \cdot h_i$ per time step.
The second term comes from gravity waves.
The third term is motivated by surface waves due to surface tension. The complete derivation of this term is to be found in DOCUMATH_TimeStepCriterionSurfaceTension.pdf .

The global time step size is finally computed by

$$dt^{\text{global}} = \max_{i=1\ldots N} \left(dt_i^{\text{local}} \right)$$

The time step restrictions come due to the fact, that the point movement in MESHFREE is explicit.

For steering of the time step size in USER_common_variables , see TimeControl .

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · COEFF_dt_coll

COEFF_dt_coll

time step criterion from interaction model (DROPLETPHASE only) (UCV)

```
COEFF_dt_coll = 0.1
```

Default: COEFF_dt_coll = 0.0 (off)
For DROPLETPHASE particles that are potentially in a collision with other particles or a wall, the timestep is reduced by this criterion in order to guarantee a good timestep resolution of the collision.
If this time step criterion leads to a very strong time step restriction, performance can be improved by using COMP_DropletphaseSubcycles .

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · COEFF_dt_d30

COEFF_dt_d30

time step criterion depending on %ind_d30% (DROPLETPHASE only) (UCV)

```
COEFF_dt_d30 = 0.5
```

Default: COEFF_dt_d30 = 0.0 (off)

If a value bigger than zero is specified for this parameter, the timestep criterion

$$dt\|\boldsymbol{v}_i\| = C_{d30}\frac{D_i}{2} \quad \forall i$$

is introduced. This time step criterion is particularly relevant in case DROPLETPHASE interactions are computed.

If this time step criterion leads to a very strong time step restriction, performance can be improved by using COMP_DropletphaseSubcycles .

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · COEFF_dt_free

COEFF_dt_free

(experimental) factor for exaggerated movement of the free surface (UCV)

```
COEFF_dt_free = 3.0
```

Default: COEFF_dt_free = 1.0

In the example above, the free surface travels three times as fast as given by the velocity.

Note: This parameter was introduced for faster finding of the steady state of a flow in conjunction with EULER .
For LAGRANGE , it does not make sense to use it.

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · COEFF_dt_virt

COEFF_dt_virt

(chamberwise) scaling factor for the virtual time step size (UCV)

```
COEFF_dt_virt = 0.01
```

Default: COEFF_dt_virt = 1.0

See VirtualTimeStepSize for the mathematical/numerical algorithm.

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · COEFF_mue

COEFF_mue

scaling factor for numerical viscosity (UCV)

COEFF_mue corresponds to the paramter C in the definition of the numerical viscosity, see %ind_ETA_sm% . For the use in the numerical scheme, see v-- and vp- .

```
COEFF_mue = 0.5
```

Default: COEFF_mue = 1.0

Note: Positive values of COEFF_mue<1.0 should lead to results that are closer to the actual solution. However, this can

lead to
numerical instabilities. In this case, COEFF_mue should be enlarged. If required, also values >1.0 can be chosen, e.g. 2 or 4.

COMP_CosEdgeAngle

(chamberwise) parameter to identify edges in geometry (UCV)

```
COMP_CosEdgeAngle = 0.5
```

Default: COMP_CosEdgeAngle = 0.8

Edges between boundary elements are detected if

$$cos(n_1 \cdot n_2) < \text{COMP\_CosEdgeAngle}$$

with n_1, n_2 the normals of the associated boundary points.

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

COMP_DoOrganizeOnlyAfterHowManyCycles

do the point cloud organization only after how many time cycles (UCV)

```
COMP_DoOrganizeOnlyAfterHowManyCycles = 3
```

Default: COMP_DoOrganizeOnlyAfterHowManyCycles = 1

This feature tries to prevent adding or removing operations of MESHFREE points.
The whole neighborhood relationship is kept.
The points, however, are moved as usual with their transport velocity.

This feature is especially useful if the pointcloud moved only little compared to the smoothing length.
Reasons for this might be (among others):

- small value of COEFF_dt
- big values of surface tension, also here the time step size might drop considerably.
- KOP using EULER instead of LAGRANGE with non-moving geometries

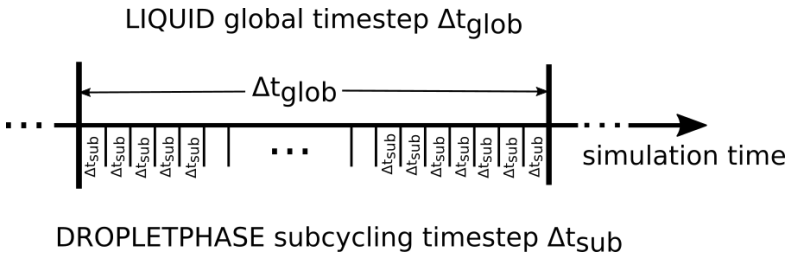
Note: This feature is especially helpful if LINEQN_solver and/or LINEQN_solver_ScalarSystems is set to 'SAMG'.
As the neighborhood graphs are kept for several time steps, the matrix setup operations do not have to be executed for these time cycles, and so a lot of computation time can be saved.

COMP_DropletphaseSubcycles

switch for DROPLETPHASE subcycling (UCV)

For modeling the dynamics of particle-particle and particle-wall interaction very small timesteps might be necessary. These timesteps can be orders of magnitude smaller than the maximum timestep for a participating fluid. If the global timestep is reduced to these small timesteps, then the performance is significantly decreased.

In order to keep a good performance, there is the possibility to resolve the dynamics of the DROPLETPHASE in subcycles.



The functionality is switched on with parameter COMP_DropletphaseSubcycles.

```
COMP_DropletphaseSubcycles = 1 # turn on subcycling
```

This means that at the beginning of the timestep in DROPLETPHASE it is determined how many substeps are likely needed to fulfill criterions for COEFF_dt_d30 and COEFF_dt_coll in every substep. This number of substeps will be performed. If during subcycling it is realized that the substep size was too big, then this will yield a reduction of the global timestep in the next timestep. The next global timestep (from DROPLETPHASE perspective) is only determined by COEFF_dt criterion, as it guarantees sufficient quality of neighborhood information for the particles.

There is also the option to introduce a limit for the maximum number of allowed subcycles: if

```
COMP_DropletphaseSubcycles = -10 # use at maximum 10 subcycles
```

is specified, the algorithm will strictly obey a maximum of 10 subcycles, irrespective of possible violations of time step criteria. Other than in the case above, the global timestep will then also be influenced by the specified number and the criterions given by COEFF_dt_d30 and COEFF_dt_coll_UCV.

Default: COMP_DropletphaseSubcycles = 0 (subcycling switched off)

The subcycling only gets activated whenever the global time step Δt_{glob} is larger than any of the DROPLETPHASE time steps dictated by

- DELT_dt_AddCond
- COEFF_dt_d30
- COEFF_dt_coll

In this case, the solver will execute multiple subcycles with a reduced time step that satisfies both of these conditions.

Structure of subcycling

At the beginning of a global time step the following is done first:

- Reading of PhysicalProperties
- Computation of layer thickness and curvature (see LiquidLayer)

Then, in each subcycle the following steps are executed:

- Treatment of boundary conditions (in particular wall collisions for %BND_COLLISION%)
- Update body forces defined via gravity , FreeFlight
- Resolve Particle-Particle collisions as defined via ParticleInteraction , see DropletCollisions
- Calculation of the new particle velocities
- Movement the particles (second order displacement)
- For particles near boundary update the distance to boundary virtually by considering the calculated displacement normal to the boundary element.

Currently not included in the subcycling:

- LiquidLayer : modeling of liquid layers as a 2D shallow water phase

Important remarks

Due to the structure of the subcycling procedure the following points should be kept in mind

- Specifying a value not equal 0 here yields that the particle displacement must be done within the DROPLETPHASE -Routine instead of the central displacement-Routine.
- The value supplied via DarcyBasisVelocity will be read before the subcycling and stored in %ind_v0Darcy%. When considering a drag force acting on the droplets (cf. FreeFlight) projecting the LIQUID velocity in every subcycle is often unnecessary. In these cases it is better to store the projected velocity in %ind_v0Darcy% and use this index in the drag equation supplied via gravity .

COMP_DropletphaseWithDisturbance

disturbance for DROPLETPHASE (UCV)

```
COMP_DropletphaseWithDisturbance = 1
```

Default: COMP_DropletphaseWithDisturbance = 0

By default the update of the positions of DROPLETPHASE points is:

$$\mathbf{x}_{\text{default}} = \mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \Delta t_i \cdot \mathbf{v}_i^{n+1}$$

$\mathbf{x}_i^{n+1}$ is the current and $\mathbf{x}_i^n$ is the previous position.
Δt_i is the current time step size and $\mathbf{v}_i^{n+1}$ is the current velocity.

If the disturbance is switched on by COMP_DropletphaseWithDisturbance = 1, the default update is disturbed by the following procedure.

1.) Rotate the default update position $\mathbf{x}_{\text{default}}$ by a small, smoothing length dependent angle (based on a random number) with respect to a random, normalized axis through the previous position:
   - random number r determines the sign (-1, 0, 1) of angle α (rotation only for non-zero sign)
   - random vector $\mathbf{d}$ determines the rotation axis as $\mathbf{a} = \frac{\mathbf{x}_i^n - \mathbf{d}}{\|\mathbf{x}_i^n - \mathbf{d}\|_2}$
   - rotation of default update position by $\mathbf{x}_i^{n+1} = (\mathbf{a} \cdot \mathbf{x}_{\text{default}})\mathbf{a} + \cos(\alpha)(\mathbf{a} \times \mathbf{x}_{\text{default}}) \times \mathbf{a} + \sin(\alpha)(\mathbf{a} \cdot \mathbf{x}_{\text{default}})$

2.) Adapt the current velocity.

Note: This procedure guarantees that the distance between previous and current position is not changed compared to the default behavior.

COMP_RemeshBoundary

parameter to control remeshing of IGES-files (UCV)

```
COMP_RemeshBoundary = 1
```

Default: COMP_RemeshBoundary = -1

The boundary is remeshed if COMP_RemeshBoundary>0.
That makes sense only if an IGES-file is used. In this case, the triangle size is taken by COMP_RemeshBoundary *SmoothingLength.

The result of the meshing operation is written in the file .FPMproject_CompleteGeometry.FDNEUT.
In order to visualize, a .case-file is written in SAVE_path .

COMP_TypeSmooth_Eta

type for smoothing of viscosity (UCV)

```
COMP_TypeSmooth_Eta = 0
```

Default: COMP_TypeSmooth_Eta = 1 (logarithm -- smoothing -- exponent)

Direct smoothing is achieved by COMP_TypeSmooth_Eta = 0.

COMP_TypeSmooth_Rho

type for smoothing of density (UCV)

```
COMP_TypeSmooth_Rho = 1
```

Default: COMP_TypeSmooth_Rho = 0 (logarithm -- smoothing -- exponent)

Direct smoothing is achieved by COMP_TypeSmooth_Rho = 0.

COMP_dt_indep

parameter to switch on independent time stepping for two-phase LIQUID simulations with v-- and vp- (UCV)

Set

```
COMP_dt_indep = 1
```

or any other integer value >0 to switch on the independent time stepping for two-phase LIQUID simulations with v-- and vp-. Furthermore, the write-out of the .dtindep file into the same folder as the default timestep file (see TimestepFile) is enabled.

Default: COMP_dt_indep = 0

COMP_facSmooth_Eta

parameter for weight kernel definition for smoothing of viscosity (UCV)

```
COMP_facSmooth_Eta = 6.0
```

Default: COMP_facSmooth_Eta = 3.0

$$W_{ij} = \exp\left(-c \cdot \frac{(\boldsymbol{x}_j - \boldsymbol{x}_i)^2}{\frac{1}{2}(h_j^2 + h_i^2)}\right)$$

The value of COMP_facSmooth_Eta defines c in the equation above.

The bigger the value of COMP_facSmooth_Eta , the more narrow the kernel and the less points in neighborhood are considered for smoothing.

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · COMP_nbSmooth_Eta

COMP_nbSmooth_Eta

number of smoothing cycles for effective and total viscosity (UCV)

COMP_nbSmooth_Eta = 5

Default: COMP_nbSmooth_Eta = 2

We smooth the values of %ind_ETA_sm% and %ind_ETA_eff% .

If $\hat{\eta}_i^k$ is the smoothed version, the total viscosity after the k-th smoothing cycle at the MESHFREE point with index i, then the new value at cycle (k+1) is given by

$$\hat{\eta}_i^{k+1} = \frac{\sum_{j=1}^{N} W_{ij} \cdot \hat{\eta}_j^k}{\sum_{j=1}^{N} W_{ij}}$$

i.e. a Shepard-based smoothing.
The weight kernel W_{ij} is defined by COMP_facSmooth_Eta .

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · CONTROL_StopAfterReadingGeometry

CONTROL_StopAfterReadingGeometry

stops the MESHFREE program after geometry is read (UCV)

CONTROL_StopAfterReadingGeometry = 1

Default: CONTROL_StopAfterReadingGeometry = 0 (no geometry checking)

option	effect
1	MESHFREE reads the geometry, writes a result file and then the computation stops. Some simple checks concerning the geometry can be done without waiting for the whole point cloud generation.
2	MESHFREE reads the geometry, and then goes into the time integration without creating the MESHFREE pointcloud. I.e. the geometry is moving due to the MOVE statements given in USER_common_variables.dat. Results are written due to the SAVE_first and SAVE_interval statements, enabling the user to veryfy the MOVE commands.
3	same as 2. Additionally, in each time cycle we compute to search tree for the geometry (boundary elements), thus, we can check the performance of the organization steps or check rigid body movement with collisions.

Note: The parameter RepairGeometry is ignored, if CONTROL_StopAfterReadingGeometry > 0.

DIFFOP_ConsistentGradient

*consistent gradient in the sense d/dn = n*grad (UCV)*

```
DIFFOP_ConsistentGradient = 1
```

Default: DIFFOP_ConsistentGradient = 0

Adapt the normal direction of the gradient operator such that n*grad = d/dn, where d/dn is the Neumann (i.e. very stable) operator.

DIFFOP_Neumann_ExcludeBND

(chamberwise) parameter to exclude boundary points from the neighborhood for the computation of the Neumann operators (UCV)

```
DIFFOP_Neumann_ExcludeBND = 90.0
```

Default: DIFFOP_Neumann_ExcludeBND = -1.0 (do not exclude any boundary point from the neighborhood)

In order to exclude all neighbor boundary points from the stencil, set

```
DIFFOP_Neumann_ExcludeBND = 360
```

A boundary point j is excluded from the Neumann stencil computation of point i, if the angle between the two boundary normals fulfills

$$(\boldsymbol{n}_i \sphericalangle \boldsymbol{n}_j) < \alpha$$

where alpha is the value of DIFFOP_Neumann_ExcludeBND , to be given in degrees.

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

DIFFOP_WeightReductionInCaseOfDeactivation

(chamberwise) parameter to reduce the weight of a neighbor point in case of deactivation (UCV)

```
DIFFOP_WeightReductionInCaseOfDeactivation = 0.0
```

Default: DIFFOP_WeightReductionInCaseOfDeactivation = 0.0001 (keep a small value in order to not run into numerical singularity of the leaset-squares-systems if all neighbors are deactivated hazardously)

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

DIFFOP_kernel_Gradient

(chamberwise) factor for the weight kernel for the least squares approximation stencils for gradients (UCV)

The differential operators are introduced in DOCUMATH_DifferentialOperators.pdf .
Especially, see section 1 of this document, where the weight kernels are introduced. In principle, the weight kernel has the form

$$W(\mathbf{x}_j, \mathbf{x}) = \exp\left(-\alpha \frac{\|\mathbf{x}_j - \mathbf{x}\|^2}{h(\mathbf{x}_j)^2}\right)$$

With DIFFOP_kernel_Gradient , we define the parameter α for the weight kernel used for the gradient approximation stencils.
Big values make the kernel narrow, small values make it broad.

DIFFOP_kernel_Gradient = 6

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · DIFFOP_kernel_Laplace

DIFFOP_kernel_Laplace

(chamberwise) factor for the weight kernel for the least squares approximation stencils for the Laplacian (UCV)

DIFFOP_kernel_Laplace = 6

Default: DIFFOP_kernel_Laplace = 2

Big values make the kernel narrow, small values make it broad, c.f. DIFFOP_kernel_Gradient .

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · DIFFOP_kernel_Neumann

DIFFOP_kernel_Neumann

(chamberwise) factor for the weight kernel for the least squares approximation stencils for Neumann operators (UCV)

DIFFOP_kernel_Neumann = 5.0

Default: DIFFOP_kernel_Neumann = 2.0

The weight for the computation of the differential Neumann operators is given by

$$W_{ij} = exp\left(-\alpha \cdot \frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{\frac{1}{2}(h_i^2 + h_j^2)}\right)$$

where alpha is equal to the value of DIFFOP_kernel_Neumann .

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

MESHFREE · InputFiles · USER_common_variables · __Parameters__ ·

DIFFOP_kernel_Transport

DIFFOP_kernel_Transport

(chamberwise) factor for the weight kernel for the least squares approximation stencils for the transport operators (UCV)

```
DIFFOP_kernel_Transport = 6
```

Default: DIFFOP_kernel_Transport = 2

Big values make the kernel narrow, small values make it broad, c.f. DIFFOP_kernel_Gradient .

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER). If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · DIFFOP_laplace

DIFFOP_laplace

type of least squares approximation stencils for the Laplacian (UCV)

Default: DIFFOP_laplace = DIFFOP_laplace_optimized

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · DP_UseOnlyRepulsiveContactForce

DP_UseOnlyRepulsiveContactForce

switch regarding attractive forces in spring-damper model (UCV)

```
DP_UseOnlyRepulsiveContactForce = 0
```

Default: DP_UseOnlyRepulsiveContactForce = 1

For certain collision models such as the spring-damper model in DROPLETPHASE , the model may formally lead to attractive forces during the separation phase. By default these attractive forces will be prevented and the contact force set to zero. Setting the above flag to zero will instead allow attractive forces.

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · FLIQUID_ConsistentPressure_Version

FLIQUID_ConsistentPressure_Version

version how to compute the consistent pressure (UCV)

```
FLIQUID_ConsistentPressure_Version = 2111 # deprecated, see AlternativeDPA
FLIQUID_ConsistentPressure_Version = 1127 # use this instead
```

Default: FLIQUID_ConsistentPressure_Version = 1111

option	description
first digit	Version of how to compute the consistent dynamic pressure, cf. DynamicPressureAlgorithm .
	Version 1: div((1/rho)*grad_p) = ... see ClassicalDPA
	Version 2: sum(W_ij*(p_j-p_i)) = ... see AlternativeDPA
	Version 3: experimental, do not use.
	Version 4: dynamic pressure is not computed (i.e. it remains what is there from the step $p_{dyn}^{n+1} = p_{dyn}^{n} + c$ with c denoting the correction pressure in the Chorin (v--) or penalty (vp-) formulation
second digit	Version how to compute the acceleration.
	Version 1: $\frac{d\mathbf{v}}{dt} = (\mathbf{v}^T \cdot \nabla) \cdot \mathbf{v}$ -> quasistationary approach
	Version 2: $\frac{d\mathbf{v}}{dt} = \frac{(\mathbf{v}(t) - \mathbf{v}(t-\Delta t))}{\Delta t}$ -> dynamic approach
	Version 3: $\frac{d\mathbf{v}}{dt} = \nabla^T \cdot \left((\mathbf{v}-\mathbf{v}_0)(\mathbf{v}-\mathbf{v}_0)^T\right) - (\mathbf{v}-\mathbf{v}_0)\left(\nabla^T \mathbf{v}\right)$ -> local quasistationary approach with chain rule in order to isolate the div(v)-part. The reference system is travelling with the speed $\mathbf{v}_0$ of the local MESHFREE point.
third digit	Version how to compute PSI, see ComputationOfPSI .
	Version 1: div(div(eta*grad(v)))
	Version 2: divBAR(div(eta*grad(v)))
fourth digit	Version how to compute PHI.
	There are 8 variations, see ComputationOfPHI . This option makes sense only in case of the ClassicalDPA . In case of AlternativeDPA , keep this value at 1.

Note: The **second digit** has impact only if

- regularization of the pressure system is requested by RegularizeDPA . Here, it impacts the way the target pressure gradient is computed.
- version 4 or 8 is used for ComputationOfPHI (fourth digit).
- %BND_none% is used as a boundary condition, as this condition is based on the AlternativeDPA -algorithm, and so this digit impacts the computation of the target pressure gradient.

We suggest:

```
FLIQUID_ConsistentPressure_Version = 1227
FLIQUID_ConsistentPressure_CoeffMM = 0.01
```

useful options	characteristics of the numerical results
FLIQUID_ConsistentPressure_Version = **1111** (classical approach)	%BND_none% only valid in quasistationary boundaries
FLIQUID_ConsistentPressure_Version = **1127** (same as 2111)	very smooth results, also here %BND_none% only valid in quasistationary boundaries
FLIQUID_ConsistentPressure_Version = **1227**	%BND_none% valid in any case, as accelerations are computed exactly. However, the results might be noisy.
FLIQUID_ConsistentPressure_Version = **1327**	%BND_none% valid in any case. However, accelerations are computed on a local quasistationary approach (each point forms an observer coordinate system). These values might be less precise than 1227, the results however are more smooth.
FLIQUID_ConsistentPressure_Version = **1228**	Numerically most natural, as the acceleration is given by the finite temporal difference of the previous and current velocities, and PHI is the divergence of this term. However, it produces more noises in the pressure solution.

Note: The understanding of **"quasistationary"** is:

- at a fixed location of an observer, the physical quantities only slowly change in time.
- watercrossing with fixed pool and moving car IS NOT quasistationary, because an observer standing in the pool will notice dramatic changes as the car drives by.
- watercrossing with fixed car and moving pool IS INDEED quasistationary, because the observer in the car will see slow changes of the water motion as the car constanty drives through the pool.

In case of **non-quasistationary** flow, set FLIQUID_ConsistentPressure_Version = 1227 or FLIQUID_ConsistentPressure_Version = 1327.

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · FOFTLIQUID_AdditionalCorrectionLoops

FOFTLIQUID_AdditionalCorrectionLoops

additional velocity correction loops (UCV)

```
FOFTLIQUID_AdditionalCorrectionLoops = 2
```

Default: FOFTLIQUID_AdditionalCorrectionLoops = 0

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · IGES_Accuracy

IGES_Accuracy

relative accuracy for consistency checks of IGES-faces (UCV)

```
IGES_Accuracy = 1.0e-6
```

Default: IGES_Accuracy = 1.0e-4

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · IGES_HealCorruptFaces

IGES_HealCorruptFaces

allow a certain depth of healing triangulation of IGES faces by refinement (UCV)

In order to make work the triangularion of IGES faces, consecutively refine the triangulation by this given number of levels.

IGES_HealCorruptFaces = 5

Default: IGES_HealCorruptFaces = 1

option	description
0	no local refinement, but reject if triangulation occurs to be corrupt
-1	keep even corrupt triangulation

LINEQN_scaling

choose the way how to scale/normalize the linear systems (UCV)

Currently, this parameter is implemented only if LINEQN_solver and/or LINEQN_solver_ScalarSystems is set to 'SAMG'.

LINEQN_scaling = 'NONE'

Default: LINEQN_scaling = 'NORM'

option	description
'NORM'	Normalize, i.e. multiply the rows of the matrix such that the diagonal element becomes 1.
'PODI'	Multiply the row of the matrix with -1 if the original diagonal entry is negative.
'NONE'	Do not normalize at all, i.e. keep the matrix in its original state.
'NATV'	Try to construct the vp- system in the sense of the saddle point method:
	Try to establish (A B \\ B' C), where B' is approximately the transpose of B. It would exactly be the transpose, if B was antisymmetric. B contains the d/dx, d/dy, d/dz operators. In MESHFREE , they are not strictily antisymmetric.

LINEQN_solver_ScalarSystems

linear solver to be used for the scalar systems like pressure, temperature, etc. (UCV)

LINEQN_solver_ScalarSystems = 'BCG2'

Default: LINEQN_solver_ScalarSystems = 'BCN2'

option	description
'BiCG' and 'BCG1'	BiCGstab, using matrix-times-vector emulation for the big system (i.e. do not construct the linear system explicitly, but provide a subroutine that computes the result of the matrix-vector-operation)
'BCG2'	BiCGstab(2), using matrix-times-vector-emulation
'SAMG'	SAMG-solver, Fraunhofer SCAI
'BCN1'	BiCGstab, no SPAI-preconditioning
'BCN2'	BiCGstab(2), no SPAI-preconditioning, **default**

Expert option: auto-chooser
'AUTO:xxxx:yyyy:n' -> Automatically choose between 2 solvers xxxx and yyyy from the list above every n time steps.

More information: BiCGstab , BiCGstab(2) , SAMG

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · LINEQN_solver

LINEQN_solver

linear solver to be used for the coupled vp- or v-- system (UCV)

```
LINEQN_solver = 'BCG2'
```

Default: LINEQN_solver = 'BCX2'

option	description
'BiCG'	BiCGstab, using matrix-times-vector emulation for the big system (i.e. do not construct the linear system explicitly, but provide a subroutine that computes the result of the matrix-vector-operation)
'BCG2'	BiCGstab(2), using matrix-times-vector-emulation
'BCX1'	BiCGstab, explicitly construct the matrix (takes more memory)
'BCX2'	BiCGstab(2), explicitly construct the matrix (faster, but takes more memory), **default**
'SAMG'	Algebraic Multigrid method from the SAMG-solver library, Fraunhofer SCAI
'BCGL'	BiCGstab(l), using matrix-times-vector emulation, experimental , see also BCGSL_ell

Expert option: auto-chooser
'AUTO:xxxx:yyyy:n' -> automatically choose between 2 solvers xxxx and yyyy from the list above every n time steps.

More information: BiCGstab , BiCGstab(2) and BiCGstab(l) , SAMG .

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · MEMORIZE_ResetReadFlag

MEMORIZE_ResetReadFlag

reset frequency for MEMORIZE_Read flag (UCV)

```
MEMORIZE_ResetReadFlag = 3
```

Default: MEMORIZE_ResetReadFlag = 10

If points are read in by MEMORIZE_Read statements, the corresponding flag is reset after the given number of time steps. Interior points with flag larger than zero are excluded from the free surface check.

PHASE_distinction

invoke detection of interface connections (UCV)

```
PHASE_distinction = 'YES'
```

Default: PHASE_distinction = 'NON'
Setting this parameter to 'YES', invokes detection of interphase connections. Each boundary point (also free surface point) searches for another boundary point
of a different chamber, which is close enough and with which it can exchange interphase boundary conditions, see BCON_CNTCT .

If a contact point is found, the index of this point is stored in %ind_iopp% .

PointDsplMethod

(experimental) Choice among different ways to move points in Lagrangian framework (UCV)

```
PointDsplMethod = 4
```

Default: PointDsplMethod = 0

option	description
0	Default -> same as 2
1	First order, velocity assumed constant between time levels
2	Second order, velocity derivative assumed constant between time levels
3	Moves points along the streamlines at that time level
4	Moves points by considering the change of streamlines from the previous time level to this one
5	Substepping method (** WILL NOT WORK WITH MPI for more than one process **)

RepairGeometry

enforce clustering of geometry nodes upon read-in (UCV)

RepairGeometry = 0.001

Default: RepairGeometry = -1.0

If the triangulation and the corresponding node points of two surfaces sharing a common edge do not conform, unphysical effects may occur at the edge in case of points slipping from one surface to the other or tearing off at the edge. RepairGeometry > 0 enforces clustering of the geometry node points relative to the defined smoothing length upon read-in.

Note:

- The use of this parameter alters the geometry, use with caution and consider remeshing the geometry wrt conformity of the node points.
- RepairGeometry is ignored, if CONTROL_StopAfterReadingGeometry > 0.

RepresentativeMass_iData

(chamberwise) parameter for the RepresentativeMass algorithm (UCV)

```
RepresentativeMass_iData = ( iTrigger, newPoints, inactiveOrDeletedPoints, nbSmootingLoops,
correctionFactorPerSmoothingLoop, ...
iMethodSmooth, whichVi, iMethodRepDens, startAtTimeCycle, ...
Wfactor, VWexponent, Kfactor, KWexponent, Mexponent , ...
deletion_weightInflowOutflow, deletion_weightOtherBND , ...
$eqnForFitering$ )
```

Default: off

```
RepresentativeMass_iData = ( 0, 1, 1, 1, 10, 1, 1, 1, 2, 2, 0, 2, 0, 1, 1000, 100, 0 )
```

RepresentativeMass_iData = 1 is equivalent to

```
RepresentativeMass_iData = ( 1 , 1, 1, 1, 10, 1, 1, 1, 2, 2, 0, 2, 0, 1, 1000, 100, 0 )
```

and switches the algorithm on without changing the default values of the other parameters.

RepresentativeMass_iData switches on the distribution of the representative masses within the points in the fluid domain. The strength of the correction itself is controlled by the two parameters VOLUME_correction_FreeSurface or VOLUME_correction_local. One or both of these parameters must additionally be set in order to activate the Volume Correction algorithm.

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

entry	description
iTrigger	global switch for representative mass algorithm
	off: 0 (default), on: 1, see RepresentativeMassAlgorithm
newPoints	number of loops to provide representative mass packages from existing points to new points
	Default: 1
inactiveOrDeletedPoints	**for development/debugging, KEEP AT 1**

nbSmootingLoops	number of iteration loops per time cycle of the Smoothing algorithm
correctionFactorPerSmoothingLoop	multiply the mass change in Smoothing by a reducing factor **(in percent!!!)**
	$$\Delta\widehat{m}_i = \alpha \left(\sum_j \Delta\widehat{m}_{ij} - \Delta\widehat{m}_{ji} \right)$$
iMethodSmooth	method for Smoothing algorithm
	choose 1, 2, or 3.
	Recent true applications show, that most efficient smoothing is achieved with method 3. The other methods might provoke strange behavior.
whichVi	**for development/debugging, KEEP AT 1**
iMethodRepDens	method how to compute the representative density, see DefinitionRepresentativeDensity
startAtTimeCycle	start the representative mass analysis at this time cycle
Wfactor	value of α_W , see DefinitionRepresentativeDensity
VWexponent	value of β_W , see DefinitionRepresentativeDensity
Kfactor	value of α_K , see Smoothing
KWexponent	value of β_K , see Smoothing
Mexponent	**for development/debugging, KEEP AT 1**
deletion_weightInflowOutflow	redistribution of repMass of deleted/deactivated points: additional weight factor for inflow and outflow points **(in percent!!!)**
deletion_weightOtherBND	redistribution of repMass of deleted/deactivated points: additional weight factor for other boundary points except inflow and outflow **(in percent!!!)**
$eqnForFitering$	equation number for the filter that defines, what points are allowed to carry a representative mass.
	Default: 0, other values have to be implemented in USER_common_variables

Example: implementation of a filter in USER_common_variables

```
begin_equation{ $myFilter$ } #if the functional is positive, the point is allowed to carry representative mass
if ( Y%ind_kob%=%BND_slip% ) :: -1 # points on %BND_slip% will not carry RepMass
else :: 1 # all other points regularly carry RepMass
endif
end_equation
RepresentativeMass_iData = ( ..., $myFilter$ ) # put the filter equation at the 17th position
```

Note

- The algorithm is described in RepresentativeMassAlgorithm .
- Using this volume correction will overwrite any setting for the global volume correction by VOLUME_correction .

SAMG_Setupreuse

accelerates SAMG solver for quasi-stationary point clouds (UCV)

```
SAMG_Setupreuse = 1
```

Default: SAMG_Setupreuse = 0 (no reuse)

This feature accelerates the SAMG solver by skipping its setup phase and reusing the last known setup of SAMG, i.e. the neighbor correlations of the point cloud at the time of the last computed setup are used to solve the current linear systems. Therefore, the use of COMP_DoOrganizeOnlyAfterHowManyCycles is highly advised when this option is exploited.

options	description
0	no reuse
1	reuse setup for pressure systems
2	reuse setup for velocity systems
3	reuse setup for pressure and velocity systems

SAVE_PrecisionTimestepFile

choose the precision (number of digits) for values in the timestep file (UCV)

This parameter controls the precision in TimestepFile .

```
SAVE_PrecisionTimestepFile = 8 # leads to output of the form 0.12345678E+01.
```

Default: SAVE_PrecisionTimestepFile = 5 (0.12345E+01)

SAVE_atEndOfTimestep

choose to save data for visualization at the end of time steps instead of at the start (UCV)

```
SAVE_atEndOfTimestep = 1
```

Default: SAVE_atEndOfTimestep = 0 (data is saved at the start of the time step)

Note: Any non-zero value will be treated as 1.

SCAN_ClustersOfConnectivity

(chamberwise) switch on cluster checking of MESHFREE point cloud by neighborhood connectivity (UCV)

SCAN_ClustersOfConnectivity = (10, 100)

Default: SCAN_ClustersOfConnectivity = (0, 100)

If switched on, MESHFREE determines each separate cluster of the point cloud and gives it a unique index. Clusters are formed by the neighborhood connectivities up to the given relative distance. The cluster index for each point is stored in %ind_cluster% .

entry	description
first value	If **>0** , it switches on the clustering of the point cloud. For *values larger than 1* , this denotes the minimum number of connected points required, to be considered its own cluster.
	If **<0** , it switches on the clustering of the point cloud only for postprocessing (saving of the results). For *absolute values larger than 1* , this denotes the minimum number of connected points required, to be considered its own cluster.
second value	The relative distance in percent of the local SMOOTH_LENGTH , for which two points are considered to be connected in the same cluster. Hence, 40 means points will be connected in the same cluster, if their distance is less than 0.4*H

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · STRESSTENSOR_Variante_Factor

STRESSTENSOR_Variante_Factor

factor in stress tensor time integration wrt the shear modulus (UCV)

STRESSTENSOR_Variante_Factor = 50.0

Default: STRESSTENSOR_Variante_Factor = 0.0

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · STRESSTENSOR_Variante

STRESSTENSOR_Variante

version of stress tensor time integration (UCV)

STRESSTENSOR_Variante = 7

Default: STRESSTENSOR_Variante = 3

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · V00_SmoothDivV

V00_SmoothDivV

Chorin projection: smooth the local values of div(v) before going into the correction pressure computation (UCV)

V00_SmoothDivV = 133

Default: V00_SmoothDivV = 000

entry	description
first digit	switch for projection of div(v)-values from boundary to interior
	0: no projection
	>0: projection, where the given value is the factor for the weight kernel that defines the distribution function
second digit	number of smoothing cycles
third digit	factor for the smoothing weight kernel

$$div(\boldsymbol{v})_i^{\text{smooth}} = \sum exp(-\text{SmoothDivV} \cdot r_{ij}) \cdot div(\boldsymbol{v})_j$$

Then, the Chorin correction pressure is established based on the PDE

$$div(\boldsymbol{v})_i^{\text{smooth}} = \nabla^T \left(\frac{\Delta t_{virt}}{\rho} \nabla c \right)$$

Note:

- This parameter is used to study conservation properties of MESHFREE .
- Surprisingly, it has bad effects on the smoothness of the velocity and pressure solutions. We observed transversal ripples for instance for the flow around and airfoil.

VOLUME_correction_FreeSurface

(chamberwise) parameter to correct volume by tiny global lifting of the free surface (UCV)

```
VOLUME_correction_FreeSurface = 0.001 # the volume must not be changed by more than 0.001*TotalVolume in a single time step.
```

Default: VOLUME_correction_FreeSurface = 0.0 (off)

The given value is the maximum allowed corrected volume per time step, based on the total volume of a chamber.

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER). If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.
If the volume correction for multiple chambers shall be different, use

```
VOLUME_correction_FreeSurface = 0.001
VOLUME_correction_FreeSurface (3) = 0.01
VOLUME_correction_FreeSurface (5) = 0.0
```

which sets the correction for all chambers first to 0.001, then it changes the values for chambers 3 and 5.

In general, for this type of volume correction, we first compute the potential displacement (distance D_{pot}) of the free surface by

$$D_{\text{pot}} = min\left(\alpha, \frac{V_{\text{target}} - V_{\text{current}}}{V_{\text{current}}}\right) \cdot \frac{V_{\text{current}}}{A_{\text{FreeSurface}}}$$

and then move, in every time cycle, the free surface artificially by the distance

$$D_{\text{move}} = \min\left(0.01 \cdot H, D_{\text{pot}}\right)$$

Here, α is equal to VOLUME_correction_FreeSurface.

If the RepresentativeMassAlgorithm is activated, the computation of the target volume is straight forward

$$V_{\text{target}} = \sum_{i=1}^{N} \frac{\widetilde{m}_i}{\rho_i}$$

If, moreover, the clustering of the point cloud is activated (see SCAN_ClustersOfConnectivity), the target volume and also the free surface corrections are computed clusterwise, i.e.

$$V_{\text{target}}^{k^{\text{cluster}}} = \sum_{i, i \in \Omega(k^{\text{cluster}})} \frac{\widetilde{m}_i}{\rho_i}$$

$$D_{\text{pot}}^{k^{\text{cluster}}} = min\left(\alpha, \frac{V_{\text{target}}^{k^{\text{cluster}}} - V_{\text{current}}^{k^{\text{cluster}}}}{V_{\text{current}}^{k^{\text{cluster}}}}\right) \cdot \frac{V_{\text{current}}^{k^{\text{cluster}}}}{A_{\text{FreeSurface}}^{k^{\text{cluster}}}}$$

In this case, the potential movement is displayed in the variable %ind_BNDfree_defect% , representing $\frac{D_{\text{pot}}^{k^{\text{cluster}}}}{H_i}$.

See VolumeCorrection for more information on volume correction.

VOLUME_correction

(chamberwise) parameter to correct volume by GLOBALLY adjusting the divergence of velocity term (UCV)

```
VOLUME_correction = 0.001 # the volume must not be changed by more than 0.001*TotalVolume in a single time step
```

Default: VOLUME_correction = 0.0 (off)

The given value is the maximum allowed corrected volume per time step, relative to the total volume of a chamber. MESHFREE will adjust div(v) in order to artificially provoke expanding or compressing flow to regain the correct, analytical volume.

Note:

- This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER). If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers. If the volume correction for multiple chambers shall be different, use

  ```
  VOLUME_correction = 0.001
  VOLUME_correction (3) = 0.01
  VOLUME_correction (5) = 0.0
  ```

  which sets the correction for all chambers first to 0.001, then it changes the values for chambers 3 and 5.
- The global volume correction will be turned off if the RepresentativeMass algorithm is turned on by RepresentativeMass_iData .
- See VolumeCorrection for more information on volume correction.

VOLUME_correction_local

(chamberwise) parameter to correct volume by LOCALLY adjusting the divergence of velocity term due to representative mass balance (UCV)

```
VOLUME_correction_local = 0.001
```

Default: VOLUME_correction_local = 0.0 (off)

This correction has an effect only if the representative mass algorithm is switched on, see RepresentativeMass_iData .

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

The idea of the correction is to impose additional divergence of velocity:
div_v_correction = min((Y %ind_r_rep% -Y %ind_r%)/Y %ind_r% , VOLUME_correction_local) / Y %ind_dt%

See VolumeCorrection for more information on volume correction.

VP0_VelocityCorrection

(chamberwise) switch to compute free surfaces (UCV)

```
VP0_VelocityCorrection = 1
```

Default: VP0_VelocityCorrection = 0

$$\boldsymbol{v}^{n+1} = \tilde{\boldsymbol{v}}^{n+1} - \frac{\tilde{\Delta t}_v}{\rho} \cdot \nabla c$$

By default, this correction (Chorin-correction) is switched off for the "vp-"-option, as we assume the velocity to be sufficiently close to its appropriate value of div(v). However, theoretically it is not wrong to perform the correction, see equation (24) in Meshfree_Methods_Proceeding_Paper_Jefferies_Kuhnert_17042014.pdf or equation (2.6) in DOCUMATH_ScalingOfLinearSystem_MxV.pdf .

There is one risk: if the correction pressure (%ind_c%) is corrupt, that will then also mess up the velocity.

---

compute_FS

(chamberwise) switch to compute free surfaces (UCV)

Decide whether or not to check for free surfaces.

```
compute_FS = 'NON' # do NOT check for free surfaces (default)
compute_FS = 'YES' # DO check for free surfaces
```

This parameter can also be set per chamber (see also KindOfProblem , CHAMBER)

```
compute_FS(1) = 'NON' # do NOT check for free surfaces, e.g. for air
compute_FS(2) = 'YES' # DO check for free surfaces, e.g. for water
```

Note: The same parameter can also be set in common_variables . Definitions in USER_common_variables are dominant.

Default: compute_FS = 'NON'

---

compute_phase_boundary

(obsolete) invoke detection of interface connections (UCV)

Obsolete, use PHASE_distinction instead.

---

damping_p_corr

(chamberwise) parameter to reduce the dynamic pressure as initial guess for the next time level (UCV)

```
damping_p_corr = 0.95
```

Default: damping_p_corr = 0.999

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER). If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

See v-- and vp- for details, especially look for $\tilde{p}^n_{dyn}$.

---

eps_T

precision in the breaking criterion for the linear systems of temperature (UCV)

```
eps_T = 1.0e-4
```

Default: eps_T = 1.0e-6

Details can be found in DOCUMATH_BreakingCriterionLinearSystems.pdf .

---

eps_p

precision in the breaking criterion for the linear systems of pressure (UCV)

```
eps_p = 1.0e-4
```

Default: eps_p = 1.0e-6

Details can be found in DOCUMATH_BreakingCriterionLinearSystems.pdf .

---

eps_phyd

precision in the breaking criterion for the linear systems of hydrostatic pressure (UCV)

```
eps_phyd = 1.0e-4
```

Default: eps_phyd = 1.0e-6

Details can be found in DOCUMATH_BreakingCriterionLinearSystems.pdf .

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · eps_v

eps_v

precision in the breaking criterion for the linear systems of velocity (UCV)

```
eps_v = 1.0e-3
```

Default: eps_v = 1.0e-4

Details can be found in DOCUMATH_BreakingCriterionLinearSystems.pdf .

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · max_N_stencil

max_N_stencil

maximum number of neighbor points accepted for stencil computation and numerics (UCV)

```
max_N_stencil = 25
```

Default: max_N_stencil = 40

This parameter defines the maximum number of accepted neighbor points for the pure numerics (stencil computation, differential operators). Out of the complete neighbor list, MESHFREE selects the **max_N_stencil** closest ones. This number is relevant for ALL points (interior + boundary).

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · ord_eval

ord_eval

define approximation order for refill points (UCV)

Define the approximation order for the approximation of all necessary values (velocity, temperature, pressure, etc.) of a newly created point during simulation. The approximation is done by using the MESHFREE least-squares operators. The order will be reduced or increased automatically if deemed necessary.

```
ord_eval = 2
```

Default: ord_eval = 3

MESHFREE · InputFiles · USER_common_variables · __Parameters__ · ord_gradient

ord_gradient

(chamberwise) approximation order of the gradient operators (UCV)

Define the approximation order for gradient approximation using the MESHFREE least-squares differential operators. The order will be reduced or increased automatically if deemed necessary.

The differential operators are introduced in DOCUMATH_DifferentialOperators.pdf , see especially section 2.2 for statements about the approximation order.

```
ord_gradient = 2
```

Default: ord_gradient = 3

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER). If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

Special feature:

```
ord_gradient = -2
```

In this case, the gradient operator is not computed directly, but retrieved from the Laplace operator in the following sense:

$$c_{ij}^{x} = \frac{1}{2} c_{ij}^{\Delta} \cdot (x_j - x_i)$$
$$c_{ij}^{y} = \frac{1}{2} c_{ij}^{\Delta} \cdot (y_j - y_i)$$
$$c_{ij}^{z} = \frac{1}{2} c_{ij}^{\Delta} \cdot (z_j - z_i)$$

ord_laplace

define approximation order of the Laplace operators (UCV)

Define the approximation order for Laplace approximation using the MESHFREE least-squares differential operators. The order will be reduced or increased automatically if deemed necessary.

The differential operators are introduced in DOCUMATH_DifferentialOperators.pdf , see especially section 2.2 for statements about the approximation order.

```
ord_laplace = 2
```

Default: ord_laplace = 3

radius_hole

relative allowed hole size (UCV)

A hole in a MESHFREE point cloud shall not be bigger than radius_hole *SmoothingLength. If a hole is bigger, it will be filled by a new MESHFREE point.

```
radius_hole = 0.40
```

Default: radius_hole = 0.45

rel_dist_bound

relative distance of neighboring points at boundaries for initial filling (UCV)

```
rel_dist_bound = 0.35
```

Default: rel_dist_bound = 0.38

This parameter is only effective for initial filling of boundary points. Refilling of boundary points during the simulation is performed depending on radius_hole .

restartnewBE_filling

(chamberwise) parameter to control filling of new boundary elements upon restart (UCV)

```
restartnewBE_filling = 'YES'
```

Default: restartnewBE_filling = 'NON' (off)

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

See also ExchangeBEOnRestart .

3.1.36. __overview_of_syntax_elements__

shows all possible syntax in USER_common_variables

On this page, all left hand side keywords are updated, which can be used in USER_common_variables .
By clicking on one of the keywords, a list of links is shown with the locations the given keyword appears in one or the other way.

As the documentation is dynamically growing, the links to the given keywords will grow appropriately,
which makes the navigation within the documentation more easy.

List of members:

AbaqusInterpolation	
absolute_pressure	
ActivateChamberAtTime	
ACTIVE	
AggregationKernel	
AllowContactToChambers	NOT USED, but planned
append{	append the INTEGRATION data to an existing .timestep-file of the same structure
BC_CNTFORCE	
BC_eps	
BC_k	
BC_p	
BC_S	
BC_SUBSON	
BC_SUPERSON	
BC_T	

BC_TearOffCriterion	
BC_v	
BC_WettingAngle	
BCON	
BCON_CNTCT	
BE_MAP	Define mapping from boundary points to BE
BE_MONITOR_ITEM	
begin_alias{	beginning alias definition
begin_boundary_elements{	beginning boundary elements definition
begin_CCC_seeds2D	
begin_CCC_seeds3D	
begin_CCC_seeds6D	
begin_construct_atRestart{	beginning construct variables definition (only) at restart
begin_construct{	beginning construct variables definition
begin_curve{	beginning curve definition
begin_equation{	beginning equation definition
begin_loop{	beginning loop definition
begin_material{	(deprecated) beginning material definition
begin_pointcloud{	beginning point cloud definition
begin_save{	begin of begin_save{ environment
begin_selection{	beginning selection definition
begin_timestepfile{	begin of timestep/integration file environment
BEmap_DefaultValue	Default value of BE_MAP (UCVO)
BreakageKernel	
BUBBLE_DoTheManagement	(chamberwise) switch regarding bubble analysis (UCVO)
BUBBLE_EnforceAveragePressure	fix average pressure for all bubbles (UCVO)
BUBBLE_forbidden	
BUBBLE_pOffset	define offset pressure for bubble pressure-on-volume analysis (UCVO)
case_else{	selection element
case{	selection element
CCC_clusterAllTriangles	
CCC_CuttingDistance	

CCC_maxSegmentLength	
CCC_minNewEdgeLength	
CCC_relativeEdgeLength	
CODI_A	
CODI_c	
CODI_D	
CODI_eq	
CODI_Integration	
CODI_min_max	
CODI_min_max_RejectLinearSolution	
CODI_Q	
CODI_rho	
CODI_V	
CODI_Vimplicit	
COEFF_dt	factor for computation of time step size (UCVO)
COEFF_dt_coll	time step criterion depending on %ind_d30% (DROPLETPHASE only) (UCVO)
COEFF_dt_d30	time step criterion depending on %ind_d30% (DROPLETPHASE only) (UCVO)
COEFF_dt_Darcy	define the virtual time step size for applications with Darcy (Brinkman) term (UCVO)
COEFF_dt_free	(experimental) factor for exaggerated movement of the free surface (UCVO)
COEFF_dt_SurfaceTension_A	time step criterion for surface tension, parameter A (UCVO)
COEFF_dt_SurfaceTension_B	time step criterion for surface tension, parameter B (UCVO)
COEFF_dt_SurfaceTension_C	(experimental) time step criterion for surface tension, parameter C (UCVO)
COEFF_dt_virt	(chamberwise) scaling factor for the virtual time step size (UCVO)
COEFF_mue	scaling factor for numerical viscosity (UCVO)
CoeffDtVirt	
COMP_CosEdgeAngle	(chamberwise) parameter to identify edges in geometry (UCVO)
COMP_DoOrganizeOnlyAfterHowManyCycles	do the point cloud organization only after how many time cycles (UCVO)
COMP_DropletphaseSubcycles	switch for subcycling in DROPLETPHASE (UCVO)
COMP_DropletphaseWithDisturbance	disturbance for DROPLETPHASE (UCVO)
COMP_dt_indep	parameter to switch on independent time stepping for two-phase LIQUID simulations with v-- and vp- (UCVO)

COMP_facSmooth_Eta	parameter for weight kernel definition for smoothing of viscosity (UCVO)
COMP_nbSmooth_Eta	number of smoothing cycles for effective and total viscosity (UCVO)
COMP_RemeshBoundary	parameter to control remeshing of IGES-files (UCVO)
COMP_TypeSmooth_Eta	type for smoothing of viscosity (UCVO)
COMP_TypeSmooth_Rho	type for smoothing of density (UCVO)
COMP_ViscosityCompensation	
compute_FS	(chamberwise) switch to compute free surfaces (UCVO)
compute_phase_boundary	(obsolete) invoke detection of interface connections (UCVO)
ConsistencyChecksAtStartup	
ContinuousPhase	
CONTROL_StopAfterReadingGeometry	stops the MESHFREE program after geometry is read (UCVO)
COORDTRANS	
CouplingBFT_DataRequest	
CouplingBFT_Synchronization	
CouplingBFT_TypeOfOfOtherSimulation	give the type of the other simulation
CouplingBFT_WorkingDirectoryOfOtherSimulation	working directory of another simulation to which couling has to be performed
cv	
damping_p_corr	(chamberwise) parameter to reduce the dynamic pressure as initial guess for the next time level (UCVO)
DarcyBasisVelocity	Define velocity of porous material
DarcyConstant	Define coupling parameter for porous material
DaughterParticleDistribution	
DaughterParticleProbability	
DELT_dt	maximum allowed time step size
DELT_dt_AddCond	defines a custom time step criterion
DELT_dt_start	time step size at the start of a simulation
DELT_dt_variable	let MESHFREE control the time step size
density	
DiffLaw	
DIFFOP_ConsistentGradient	consistent gradient in the sense d/dn = n*grad (UCVO)

DIFFOP_kernel_Gradient	(chamberwise) factor for the weight kernel for the least squares approximation stencils for gradients (UCVO)
DIFFOP_kernel_Laplace	(chamberwise) factor for the weight kernel for the least squares approximation stencils for the Laplacian (UCVO)
DIFFOP_kernel_Neumann	(chamberwise) factor for the weight kernel for the least squares approximation stencils for Neumann operators (UCVO)
DIFFOP_kernel_Transport	(chamberwise) factor for the weight kernel for the least squares approximation stencils for the transport operators (UCVO)
DIFFOP_laplace	type of least squares approximation stencils for the Laplacian (UCVO)
DIFFOP_Neumann_ExcludeBND	(chamberwise) parameter to exclude boundary points from the neighborhood for the computation of the Neumann operators (UCVO)
DIFFOP_WeightReductionInCaseOfDeactivation	(chamberwise) parameter to reduce the weight of a neighbor point in case of deactivation (UCVO)
divergenceV	
DovmmUntilTime_DovpmFromTime	parameter to control the execution of v-- and vp- solvers in two-phase LIQUID simulations wrt time
DovpmFromTime	parameter to control the execution of the vp- solver in two-phase LIQUID simulations wrt time
DP_UseOnlyRepulsiveContactForce	switch regarding attractive forces in spring-damper model (UCVO)
DropletSource	
end_alias	ending alias definition
end_boundary_elements	ending boundary elements definition
end_construct	ending construct variables definition
end_construct_atRestart	ending construct variables definition (only) at restart
end_curve	ending curve definition
end_equation	ending equation definition
end_loop	ending loop definition
end_material	(deprecated) ending material definition
end_pointcloud	ending point cloud definition
end_save	end of begin_save{ environment
end_selection	ending selection definition
end_timestepfile	end of timestep/integration file environment
ENFORCE_min_max	
ENFORCE_min_max_RejectLinearSolution	
eps_p	precision in the breaking criterion for the linear systems of pressure (UCVO)

eps_phyd	precision in the breaking criterion for the linear systems of hydrostatic pressure (UCVO)
eps_T	precision in the breaking criterion for the linear systems of temperature (UCVO)
eps_v	precision in the breaking criterion for the linear systems of velocity (UCVO)
eta	
EVENT	
EventMessage	
FLIQUID_ConsistentPressure_Version	version how to compute the consistent pressure (UCVO)
FOFTLIQUID_AdditionalCorrectionLoops	additional velocity correction loops (UCVO)
ForbidContactToChambers	NOT USED, but planned
ForchheimerConstant	Define coupling parameter for porous material
GenerateBubbleAtInflow	
gravity	
HEAT_EQ_1D	
HEAT_EQ_1D_TRANSFER_COEFF_EXTERNAL	
HEAT_EQ_1D_TRANSFER_COEFF_INTERNAL	
heatsource	
IGES_Accuracy	relative accuracy for consistency checks of IGES-faces (UCVO)
IGES_HealCorruptFaces	allow a certain depth of healing triangulation of IGES faces by refinement (UCVO)
include_CCC_curves	
include_CCC_seeds2D	
include_CCC_seeds3D	
include_CCC_seeds6D	
include_Ucv{	include a file in UCV-format
INITDATA	
INTEGRATION	
KindOfProblem	Model and Solver selection
KOP	Model and Solver selection
lambda	
latentheat	

LINEQN_scaling	choose the way how to scale/normalize the linear systems (UCVO)
LINEQN_solver	linear solver to be used for the coupled vp- or v-- system (UCVO)
LINEQN_solver_ScalarSystems	linear solver to be used for the scalar systems like pressure, temperature, etc. (UCVO)
max_N_stencil	maximum number of neighbor points accepted for stencil computation and numericss (UCVO)
max_vl	
MeanNumberDaughterDroplets	
MEMORIZE_Read	
MEMORIZE_ResetReadFlag	reset frequency for MEMORIZE_Read flag (UCVO)
MEMORIZE_Write	
min_vl	
MONITORPOINTS_CREATION	
MONITORPOINTS_CREATION_FunctionEvaluation	
MONITORPOINTS_DELETION	
MONITORPOINTS_STOP	
MOVE	
mue	
NumberOfDaughterParticles	
ODE	
ord_eval	define approximation order for refill points (UCVO)
ord_gradient	(chamberwise) approximation order of the gradient operators (UCVO)
ord_laplace	define approximation order of the Laplace operators (UCVO)
parameters{	give arguments/parameters to a include file (like calling subroutines or functions)
ParticleInteraction	defines the particle interaction model (attraction and repulsion) in a particle phase (DROPLETPHASE only)
ParticlePhase	
PBE_Developement	
PBE_Model_Alpha_Max	
PBE_Model_Alpha_Min	
PBE_Model_ContinuousDragSwitch	
PBE_Model_DiffusionSwitch	

PBE_Model_E_DropletSource	
PBE_Model_K_DropletSource	
PBE_Model_KEPS_DropletVisibilitySwitch	
PBE_Model_Vmax	
PBE_Model_Vmin	
PBE_SolverSetup	
PHASE_distinction	invoke detection of interface connections (UCVO)
PointCloudReduction	
PointDsplMethod	(experimental) Choice among different ways to move points in Lagrangian framework (UCVO)
POSTBND	
POSTVOL	
radius_hole	relative allowed hole size (UCVO)
Rconst	
rel_dist_bound	relative distance of neighboring points at boundaries (UCVO)
RelaxationTime	
RemeshBoundary_OrientationBuiltInComponents	
RemeshBoundary_RemoveTinyClusters	
RepairGeometry	enforce clustering of geometry nodes upon read-in (UCVO)
RepeatCurrentTimeStep	
RepeatCurrentTimeStep_AdditionalComputationsAfterDataTransfer	
RepeatCurrentTimeStep_ChangeCVconfiguration	
RepeatCurrentTimeStep_InitializeVariables	
RepeatCurrentTimeStep_SaveVariables	
RepresentativeMass_iData	(chamberwise) parameter for the RepresentativeMass algorithm (UCVO)
Restart	launch MESHFREE on the basis of a restart file
restart_additionalBE	include additional boundary elements file during restart
restart_copying	copy alias definition for additional boundary elements during restart

restart_file	Define file name of restart files
restart_path	Define path to restart files
restart_step_size	define after how many time cycles a restart file has to be generated
restart_toberemoved	remove pre-restart boundary elements during restart
restartnewBE_filling	(chamberwise) parameter to control filling of new boundary elements upon restart (UCVO)
RestartStepSize	define after how many time cycles a restart file has to be generated
RIGIDBODY_ExternalForces	pressure to apply on rigid bodies; if not given, hydrostatic and dynamic pressure are applied
RIGIDBODY_interaction	
RIGIDBODY_pressureToApplyOnBody	pressure to apply on rigid bodies; if not given, hydrostatic and dynamic pressure are applied
SAMG_Setupreuse	accelerates SAMG solver for quasi-stationary point clouds (UCVO)
SAVE_ABAQUS	
SAVE_atEndOfTimestep	choose to save data for visualization at the end of time steps instead of at the start (UCVO)
SAVE_BE_ITEM	
SAVE_BE_MONITOR_ITEM	
SAVE_BE_NODE_ITEM	
SAVE_choose_meth	save computational results in different formats
SAVE_CoordinateSystem	saving relative to specified coordinate system (movement)
SAVE_file	file name for the results
SAVE_filter	filter MESHFREE points to be saved in the result files
SAVE_first	control first save
SAVE_format	format to save simulation data
SAVE_format_skip	skipping cycle for SAVE_format
SAVE_interval	control saving frequency
SAVE_intervall	control saving frequency
SAVE_ITEM	
SAVE_list_of_var	
SAVE_MONITOR_ITEM	
SAVE_path	absolute or relative path for the simulation results
SAVE_PID_ITEM	

SAVE_PrecisionTimestepFile	choose the precision (number of digits) for values in the timestep file (UCVO)
SAVE_QUALITYCHECK_ITEM	
SAVE_ShareScalars	
SCAN_ClustersOfConnectivity	(chamberwise) switch on cluster checking of MESHFREE point cloud by neighborhood connectivity (UCVO)
shearmodulus	
sigma	
SMOOTH_LENGTH	
specificheat	
STRESSTENSOR_Variante	version of stress tensor time integration (UCVO)
STRESSTENSOR_Variante_Factor	factor in stress tensor time integration wrt the shear modulus (UCVO)
surfacetension	
tau	
TaylorQuinneyCoefficient	
Tend	maximum final time of simulation
thermalconduction	
TimeIntegration_N_final	maximum number of timesteps
TOUCH	
Tstart	initial time of simulation
USER_h_funct	choose either constant, locally variable, or adaptive smoothing length
USER_h_max	maximum allowed smoothing length
USER_h_min	minimum allowed smoothing length
V00_SmoothDivV	Chorin projection: smooth the local values of div(v) before going into the correction pressure computation (UCVO)
v_transport	
viscosity	
VOLUME_correction	(chamberwise) parameter to correct volume by GLOBALLY adjusting the divergence of velocity term (UCVO)
VOLUME_correction_FreeSurface	(chamberwise) parameter to correct volume by tiny global lifting of the free surface (UCVO)
VOLUME_correction_local	(chamberwise) parameter to correct volume by LOCALLY adjusting the divergence of velocity term due to representative mass balance (UCVO)
VP0_VelocityCorrection	(chamberwise) switch to compute free surfaces (UCVO)
x_p1	

x_p

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · BE_MAP

BE_MAP

Define mapping from boundary points to BE

See BE_MAP

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · BEmap_DefaultValue

BEmap_DefaultValue

Default value of BE_MAP (UCVO)

See BEmap_DefaultValue .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · BUBBLE_DoTheManagement

BUBBLE_DoTheManagement

(chamberwise) switch regarding bubble analysis (UCVO)

See BUBBLE_DoTheManagement .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · BUBBLE_EnforceAveragePressure

BUBBLE_EnforceAveragePressure

fix average pressure for all bubbles (UCVO)

See BUBBLE_EnforceAveragePressure .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · BUBBLE_pOffset

BUBBLE_pOffset

define offset pressure for bubble pressure-on-volume analysis (UCVO)

See BUBBLE_pOffset .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COEFF_dt_Darcy

COEFF_dt_Darcy

define the virtual time step size for applications with Darcy (Brinkman) term (UCVO)

See COEFF_dt_Darcy .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COEFF_dt_SurfaceTension_A

COEFF_dt_SurfaceTension_A

time step criterion for surface tension, parameter A (UCVO)

See COEFF_dt_SurfaceTension_A .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COEFF_dt_SurfaceTension_B

COEFF_dt_SurfaceTension_B

time step criterion for surface tension, parameter B (UCVO)

See COEFF_dt_SurfaceTension_B .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COEFF_dt_SurfaceTension_C

COEFF_dt_SurfaceTension_C

(experimental) time step criterion for surface tension, parameter C (UCVO)

See COEFF_dt_SurfaceTension_C .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COEFF_dt

COEFF_dt

factor for computation of time step size (UCVO)

See COEFF_dt .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COEFF_dt_coll

COEFF_dt_coll

time step criterion depending on %ind_d30% (DROPLETPHASE only) (UCVO)

See COEFF_dt_coll .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COEFF_dt_d30

COEFF_dt_d30

time step criterion depending on %ind_d30% (DROPLETPHASE only) (UCVO)

See COEFF_dt_d30 .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COEFF_dt_free

COEFF_dt_free

(experimental) factor for exaggerated movement of the free surface (UCVO)

See COEFF_dt_free .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COEFF_dt_virt

COEFF_dt_virt

(chamberwise) scaling factor for the virtual time step size (UCVO)

See COEFF_dt_virt .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COEFF_mue

COEFF_mue

scaling factor for numerical viscosity (UCVO)

See COEFF_mue .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COMP_CosEdgeAngle

COMP_CosEdgeAngle

(chamberwise) parameter to identify edges in geometry (UCVO)

See COMP_CosEdgeAngle .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COMP_DoOrganizeOnlyAfterHowManyCycles

COMP_DoOrganizeOnlyAfterHowManyCycles

do the point cloud organization only after how many time cycles (UCVO)

See COMP_DoOrganizeOnlyAfterHowManyCycles .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COMP_DropletphaseSubcycles

COMP_DropletphaseSubcycles

switch for subcycling in DROPLETPHASE (UCVO)

See COMP_DropletphaseSubcycles .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COMP_DropletphaseWithDisturbance

COMP_DropletphaseWithDisturbance

disturbance for DROPLETPHASE (UCVO)

See COMP_DropletphaseWithDisturbance .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COMP_RemeshBoundary

COMP_RemeshBoundary

parameter to control remeshing of IGES-files (UCVO)

See COMP_RemeshBoundary .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COMP_TypeSmooth_Eta

COMP_TypeSmooth_Eta

type for smoothing of viscosity (UCVO)

See COMP_TypeSmooth_Eta .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COMP_TypeSmooth_Rho

COMP_TypeSmooth_Rho

type for smoothing of density (UCVO)

See COMP_TypeSmooth_Rho .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COMP_dt_indep

COMP_dt_indep

parameter to switch on independent time stepping for two-phase LIQUID simulations with v-- and vp- (UCVO)

See COMP_dt_indep .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COMP_facSmooth_Eta

COMP_facSmooth_Eta

parameter for weight kernel definition for smoothing of viscosity (UCVO)

See COMP_facSmooth_Eta .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · COMP_nbSmooth_Eta

COMP_nbSmooth_Eta

number of smoothing cycles for effective and total viscosity (UCVO)

See COMP_nbSmooth_Eta .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · CONTROL_StopAfterReadingGeometry

CONTROL_StopAfterReadingGeometry

stops the MESHFREE program after geometry is read (UCVO)

See CONTROL_StopAfterReadingGeometry .

CouplingBFT_TypeOfOfOtherSimulation

give the type of the other simulation

See CouplingBFT_TypeOfOtherSimulation .

CouplingBFT_WorkingDirectoryOfOtherSimulation

working directory of another simulation to which couling has to be performed

See CouplingBFT_WorkingDirectoryOfOtherSimulation .

DELT_dt_AddCond

defines a custom time step criterion

See DELT_dt_AddCond .

DELT_dt

maximum allowed time step size

See DELT_dt .

DELT_dt_start

time step size at the start of a simulation

See DELT_dt_start .

DELT_dt_variable

let MESHFREE control the time step size

See DELT_dt_variable .

DIFFOP_ConsistentGradient

*consistent gradient in the sense d/dn = n*grad (UCVO)*

See DIFFOP_ConsistentGradient .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · DIFFOP_Neumann_ExcludeBND

DIFFOP_Neumann_ExcludeBND

(chamberwise) parameter to exclude boundary points from the neighborhood for the computation of the Neumann operators (UCVO)

See DIFFOP_Neumann_ExcludeBND .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · DIFFOP_WeightReductionInCaseOfDeactivation

DIFFOP_WeightReductionInCaseOfDeactivation

(chamberwise) parameter to reduce the weight of a neighbor point in case of deactivation (UCVO)

See DIFFOP_WeightReductionInCaseOfDeactivation .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · DIFFOP_kernel_Gradient

DIFFOP_kernel_Gradient

(chamberwise) factor for the weight kernel for the least squares approximation stencils for gradients (UCVO)

See DIFFOP_kernel_Gradient .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · DIFFOP_kernel_Laplace

DIFFOP_kernel_Laplace

(chamberwise) factor for the weight kernel for the least squares approximation stencils for the Laplacian (UCVO)

See DIFFOP_kernel_Laplace .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · DIFFOP_kernel_Neumann

DIFFOP_kernel_Neumann

(chamberwise) factor for the weight kernel for the least squares approximation stencils for Neumann operators (UCVO)

See DIFFOP_kernel_Neumann .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · DIFFOP_kernel_Transport

DIFFOP_kernel_Transport

(chamberwise) factor for the weight kernel for the least squares approximation stencils for the transport operators (UCVO)

See DIFFOP_kernel_Transport .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · DIFFOP_laplace

DIFFOP_laplace

type of least squares approximation stencils for the Laplacian (UCVO)

See DIFFOP_laplace .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · DP_UseOnlyRepulsiveContactForce

DP_UseOnlyRepulsiveContactForce

switch regarding attractive forces in spring-damper model (UCVO)

See DP_UseOnlyRepulsiveContactForce .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · DarcyBasisVelocity

DarcyBasisVelocity

Define velocity of porous material

See DarcyBasisVelocity

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · DarcyConstant

DarcyConstant

Define coupling parameter for porous material

See DarcyConstant

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · FLIQUID_ConsistentPressure_Version

FLIQUID_ConsistentPressure_Version

version how to compute the consistent pressure (UCVO)

See FLIQUID_ConsistentPressure_Version .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · FOFTLIQUID_AdditionalCorrectionLoops

FOFTLIQUID_AdditionalCorrectionLoops

additional velocity correction loops (UCVO)

See FOFTLIQUID_AdditionalCorrectionLoops .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · ForchheimerConstant

ForchheimerConstant

Define coupling parameter for porous material

See ForchheimerConstant

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · IGES_Accuracy

IGES_Accuracy

relative accuracy for consistency checks of IGES-faces (UCVO)

See IGES_Accuracy .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · IGES_HealCorruptFaces

IGES_HealCorruptFaces

allow a certain depth of healing triangulation of IGES faces by refinement (UCVO)

See IGES_HealCorruptFaces .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · KOP

KOP

Model and Solver selection

See KindOfProblem .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · KindOfProblem

KindOfProblem

Model and Solver selection

See KindOfProblem .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · LINEQN_scaling

LINEQN_scaling

choose the way how to scale/normalize the linear systems (UCVO)

See LINEQN_scaling .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · LINEQN_solver_ScalarSystems

LINEQN_solver_ScalarSystems

linear solver to be used for the scalar systems like pressure, temperature, etc. (UCVO)

See LINEQN_solver_ScalarSystems .

LINEQN_solver

linear solver to be used for the coupled vp- or v-- system (UCVO)

See LINEQN_solver .

MEMORIZE_ResetReadFlag

reset frequency for MEMORIZE_Read flag (UCVO)

See MEMORIZE_ResetReadFlag .

PHASE_distinction

invoke detection of interface connections (UCVO)

See PHASE_distinction .

PointDsplMethod

(experimental) Choice among different ways to move points in Lagrangian framework (UCVO)

See PointDsplMethod .

RIGIDBODY_ExternalForces

pressure to apply on rigid bodies; if not given, hydrostatic and dynamic pressure are applied

```
RIGIDBODY_ExternalForces (i) = ( x, y, z, Fx, Fy, Fz )
```

default: RIGIDBODY_ExternalForces (i) = (0,0,0, 0,0,0)

RIGIDBODY_pressureToApplyOnBody

pressure to apply on rigid bodies; if not given, hydrostatic and dynamic pressure are applied

```
RIGIDBODY_pressureToApplyOnBody ($MOVEitem$) = ( [ equation for pHydrostatic ] , [ equation for pDynamic] )
```

default: ([Y %ind_p%] , [Y %ind_p_dyn%])

RepairGeometry

enforce clustering of geometry nodes upon read-in (UCVO)

See RepairGeometry .

RepresentativeMass_iData

(chamberwise) parameter for the RepresentativeMass algorithm (UCVO)

See RepresentativeMass_iData .

RestartStepSize

define after how many time cycles a restart file has to be generated

See RestartStepSize .

Restart

launch MESHFREE on the basis of a restart file

See LaunchRestart .

SAMG_Setupreuse

accelerates SAMG solver for quasi-stationary point clouds (UCVO)

See SAMG_Setupreuse .

SAVE_CoordinateSystem

saving relative to specified coordinate system (movement)

See SAVE_CoordinateSystem .
See SAVE_CoordinateSystem .

SAVE_PrecisionTimestepFile

choose the precision (number of digits) for values in the timestep file (UCVO)

See SAVE_PrecisionTimestepFile .

SAVE_atEndOfTimestep

choose to save data for visualization at the end of time steps instead of at the start (UCVO)

See SAVE_atEndOfTimestep .

SAVE_choose_meth

save computational results in different formats

See SAVE_choose_meth .
See SAVE_choose_meth .

SAVE_file

file name for the results

See SAVE_file .
See SAVE_file .

SAVE_first

control first save

See SAVE_first .
See SAVE_first .

SAVE_format

format to save simulation data

See SAVE_format .
See SAVE_format .

SAVE_format_skip

skipping cycle for SAVE_format

See SAVE_format_skip .

SAVE_interval

control saving frequency

See SAVE_interval .
See SAVE_interval .

SAVE_intervall

control saving frequency

See SAVE_interval .
See SAVE_interval .

SAVE_path

absolute or relative path for the simulation results

See SAVE_path .
See SAVE_path .

SCAN_ClustersOfConnectivity

(chamberwise) switch on cluster checking of MESHFREE point cloud by neighborhood connectivity (UCVO)

See SCAN_ClustersOfConnectivity .

STRESSTENSOR_Variante_Factor

factor in stress tensor time integration wrt the shear modulus (UCVO)

See STRESSTENSOR_Variante_Factor .

STRESSTENSOR_Variante

STRESSTENSOR_Variante

version of stress tensor time integration (UCVO)

See STRESSTENSOR_Variante .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · Tend

Tend

maximum final time of simulation

See Tend .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · TimeIntegration_N_final

TimeIntegration_N_final

maximum number of timesteps

See TimeIntegration_N_final .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · Tstart

Tstart

initial time of simulation

See Tstart .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · USER_h_funct

USER_h_funct

choose either constant, locally variable, or adaptive smoothing length

See USER_h_funct .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · USER_h_max

USER_h_max

maximum allowed smoothing length

See USER_h_max .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · USER_h_min

USER_h_min

minimum allowed smoothing length

See USER_h_min .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · V00_SmoothDivV

V00_SmoothDivV

Chorin projection: smooth the local values of div(v) before going into the correction pressure computation (UCVO)

See V00_SmoothDivV .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · VOLUME_correction_FreeSurface

VOLUME_correction_FreeSurface

(chamberwise) parameter to correct volume by tiny global lifting of the free surface (UCVO)

See VOLUME_correction_FreeSurface .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · VOLUME_correction

VOLUME_correction

(chamberwise) parameter to correct volume by GLOBALLY adjusting the divergence of velocity term (UCVO)

See VOLUME_correction .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · VOLUME_correction_local

VOLUME_correction_local

(chamberwise) parameter to correct volume by LOCALLY adjusting the divergence of velocity term due to representative mass balance (UCVO)

See VOLUME_correction_local .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · VP0_VelocityCorrection

VP0_VelocityCorrection

(chamberwise) switch to compute free surfaces (UCVO)

See VP0_VelocityCorrection .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · append{

append{

append the INTEGRATION data to an existing .timestep-file of the same structure

See AppendDataToExistingFiles

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ ·

begin_alias{

beginning alias definition

See ALIAS .

List of members:	
ACTIVE	define the activation behavior of the boundary elements of this part
BC_PASSON	for deactivated/disappearing boundary elements: give BC-flag to released MESHFREE points
BC	define flag for boundary conditions
BOUNDARYFILLING	possibility to request reduced filling behavior for MESHFREE points for parts of the boundary
CHAMBER	define the chamber index for the geometry entities
COORDTRANS	define coordinate transformation to mathematically transform long thin geometries into short thick ones
IDENT_PASSON	for deactivated/disappearing boundary elements: give IDENT-information to released MESHFREE points
IDENT	how to handle the geometry part during point cloud organization
IGNORE	ignore this geometry item upon reading from geometry file
LAYER	define layer index
MAT	define the material flag to be used, when the geometry part fills new points(mostly for initial filling)
METAPLANE	define a cutting plane for MESHFREE points
MOVE_PASSON	for deactivated/disappearing boundary elements: give MOVE-flag to released MESHFREE points
MOVE	provide a flag for the definition of boundary movement
POSTPROCESS	define flag for postprocessing/integration
REV_ORIENT	flip around orientation of boundary parts upon read-in of geometry files
SMOOTH_LENGTH__Ucv__	define flag for smoothing length definition
SMOOTH_N	invoke smoothing of the boundary
SYMMETRYFACE	trigger the geometry part as symmetryface which changes the way of distance computation
TOUCH	define the wetting/activation behavior of MESHFREE points along the given boundary part

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_alias{ · ACTIVE

ACTIVE

define the activation behavior of the boundary elements of this part

See ACTIVE .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_alias{ · BC

BC

define flag for boundary conditions

See BC .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_alias{ · BC_PASSON

BC_PASSON

for deactivated/disappearing boundary elements: give BC-flag to released MESHFREE points

See BC_PASSON .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_alias{ · BOUNDARYFILLING

BOUNDARYFILLING

possibility to request reduced filling behavior for MESHFREE points for parts of the boundary

See BOUNDARYFILLING .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_alias{ · CHAMBER

CHAMBER

define the chamber index for the geometry entities

See CHAMBER .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_alias{ · COORDTRANS

COORDTRANS

define coordinate transformation to mathematically transform long thin geometries into short thick ones

See COORDTRANS .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_alias{ · IDENT

IDENT

how to handle the geometry part during point cloud organization

See IDENT .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_alias{ · IDENT_PASSON

IDENT_PASSON

for deactivated/disappearing boundary elements: give IDENT-information to released MESHFREE points

See IDENT_PASSON .

IGNORE

ignore this geometry item upon reading from geometry file

See IGNORE .

LAYER

define layer index

See LAYER .

MAT

define the material flag to be used, when the geometry part fills new points(mostly for initial filling)

See MAT .

METAPLANE

define a cutting plane for MESHFREE points

See METAPLANE .

MOVE

provide a flag for the definition of boundary movement

See MOVE .

MOVE_PASSON

for deactivated/disappearing boundary elements: give MOVE-flag to released MESHFREE points

See MOVE_PASSON .

POSTPROCESS

define flag for postprocessing/integration

See MOVE .

REV_ORIENT

flip around orientation of boundary parts upon read-in of geometry files

See REV_ORIENT .

SMOOTH_LENGTH__Ucv__

define flag for smoothing length definition

See SMOOTH_LENGTH .

SMOOTH_N

invoke smoothing of the boundary

See SMOOTH_N .

SYMMETRYFACE

trigger the geometry part as symmetryface which changes the way of distance computation

See SYMMETRYFACE .

TOUCH

define the wetting/activation behavior of MESHFREE points along the given boundary part

See TOUCH .

begin_boundary_elements{

beginning boundary elements definition

See BoundaryElements .

List of members:

include{	definition of a geometry file to be read by MESHFREE
BND_cube	create an independent rectangular cuboid (box)
BND_cylinder	create a cylinder
BND_line	create an independent line
BND_node	create an independent node for use in other boundary entity definitions
BND_plane	
BND_point	create an independent point
BND_quad	create an independent quadrilateral
BND_tria	create an independent triangle
BND_tria6N	create an independent 6-node triangle
include_CCC_curves{	define the geometry file containing cutting curves for clustering
manipulate{	manipulate (move, rotate, ...) the geometry belonging to an alias-group
delete{	delete all the geometry belonging to a given alias-group
BNDpoints_ExtractFromNodes{	create BND_points from existing geometry nodes

BND_cube

create an independent rectangular cuboid (box)

See BND_cube .

BND_cylinder

create a cylinder

See BND_cylinder .

BND_line

create an independent line

See BND_line .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_boundary_elements{ · BND_node

BND_node

create an independent node for use in other boundary entity definitions

See BND_node .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_boundary_elements{ · BND_plane

BND_plane

See BND_plane .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_boundary_elements{ · BND_point

BND_point

create an independent point

See BND_point .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_boundary_elements{ · BND_quad

BND_quad

create an independent quadrilateral

See BND_quad .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_boundary_elements{ · BND_tria

BND_tria

create an independent triangle

See BND_tria .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_boundary_elements{ · BND_tria6N

BND_tria6N

create an independent 6-node triangle

See BND_tria6N .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_boundary_elements{ · BNDpoints_ExtractFromNodes{

BNDpoints_ExtractFromNodes{

create BND_points from existing geometry nodes

See BNDpoints_ExtractFromNodes{ .

---

delete{

delete all the geometry belonging to a given alias-group

See delete{ .

---

include_CCC_curves{

define the geometry file containing cutting curves for clustering

See include_CCC_curves .

---

include{

definition of a geometry file to be read by MESHFREE

See include{ .

List of members:	
applyAlias{	Rename BoundaryElements with the given alias name
coarsenGeometry{	coarsen the triangulation of the specified part of the geometry
duplicate{	Duplicate part of the geometry and apply a new alias
layerByCluster	assign the layer-property of a geometryical entity, possibly overrides the user given value form the ALIAS block
mirror{	generalized mirroring across a plane
offset{	shift the given geometry by a vector
removeCluster{	removes cluster(s) of the current geometry subset due to given conditions
removeIsolatedCluster{	remove clusters who have less than a given number of single geometry elements (triangles, quads, etc.)
removeOuterShell{	for shell geometry given by two closed surfaces, remove outer surface
removeTinyClusters{	remove tiny parts from a geometrical entity
reorientation{	reorientation (inside/outside) of parts of the geometry
revOrient{	Invert orientation of boundary elements
rotate{	rotate the given geometry about a point with a rotation axis and angle
scale{	scale the given geometry about the origin
symmetryfaceByCluster{	automatic distribution of SYMMETRYFACE-flags to geometry components
thickenabs{	move a given part of the geometry by an absolute value of distance
thickenexp{	move the given part of the boundary by a relative value, correlated to the locally given smoothing length
turn_6NodeTriangles_into_3NodeTriangles{	Turn 6-node triangles into 3-node triangles

applyAlias{

Rename BoundaryElements with the given alias name

See applyAlias{ .

coarsenGeometry{

coarsen the triangulation of the specified part of the geometry

See coarsenGeometry{ .

duplicate{

Duplicate part of the geometry and apply a new alias

See duplicate{ .

---

layerByCluster

assign the layer-property of a geometryical entity, possibly overrides the user given value form the ALIAS block

See layerByCluster{ .

---

mirror{

generalized mirroring across a plane

See mirror{ .

---

offset{

shift the given geometry by a vector

See offset{ .

---

removeCluster{

removes cluster(s) of the current geometry subset due to given conditions

See removeCluster{ .

---

removeIsolatedCluster{

remove clusters who have less than a given number of single geometry elements (triangles, quads, etc.)

See removeIsolatedClusters{ .

---

removeOuterShell{

for shell geometry given by two closed surfaces, remove outer surface

See removeOuterShell{ .

---

removeTinyClusters{

remove tiny parts from a geometrical entity

See removeTinyClusters{ .

---

reorientation{

reorientation (inside/outside) of parts of the geometry

See reorientation{ .

---

revOrient{

Invert orientation of boundary elements

See revOrient{ .

---

rotate{

rotate the given geometry about a point with a rotation axis and angle

See rotate{ .

---

scale{

scale the given geometry about the origin

See scale{ .

---

symmetryfaceByCluster{

automatic distribution of SYMMETRYFACE-flags to geometry components

See symmetryfaceByCluster{ .

---

thickenabs{

move a given part of the geometry by an absolute value of distance

See thickenabs{ .

thickenexp{

move the given part of the boundary by a relative value, correlated to the locally given smoothing length

See thickenexp{ .

turn_6NodeTriangles_into_3NodeTriangles{

Turn 6-node triangles into 3-node triangles

See turn_6NodeTriangles_into_3NodeTriangles{ .

manipulate{

manipulate (move, rotate, ...) the geometry belonging to an alias-group

See manipulate{ .

begin_construct_atRestart{

beginning construct variables definition (only) at restart

See ConstructClause .

begin_construct{

beginning construct variables definition

See ConstructClause .

List of members:

CONSTRUCT	mathematical construction of scalars and vectors

CONSTRUCT

mathematical construction of scalars and vectors

See ConstructClause .

---

begin_curve{

beginning curve definition

See Curves .

List of members:

depvar_default{	defines the index for the independent variable in 1D curves
depvar_horizontal{	defines the index for the horizontal independent variable in 2D curves
depvar_vertical{	defines the index for the vertical independent variable in 2D curves

---

depvar_default{

defines the index for the independent variable in 1D curves

See depvar_default .

---

depvar_horizontal{

defines the index for the horizontal independent variable in 2D curves

See depvar_horizontal .

---

depvar_vertical{

defines the index for the vertical independent variable in 2D curves

See depvar_vertical .

---

begin_equation{

beginning equation definition

See Equations .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_loop{

begin_loop{

beginning loop definition

See Loops .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_pointcloud{

begin_pointcloud{

beginning point cloud definition

See ReadInPointCloud .
See ReadInPointCloud .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_save{

begin_save{

begin of begin_save{ environment

See begin_save{

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_selection{

begin_selection{

beginning selection definition

See Selection for details.

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · begin_timestepfile{

begin_timestepfile{

begin of timestep/integration file environment

See TimestepFile .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · case_else{

case_else{

selection element

See Selection for details.

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ ·

case{

case{

selection element

See Selection for details.

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · compute_FS

compute_FS

(chamberwise) switch to compute free surfaces (UCVO)

See compute_FS .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · compute_phase_boundary

compute_phase_boundary

(obsolete) invoke detection of interface connections (UCVO)

Obsolete, use PHASE_distinction instead.

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · damping_p_corr

damping_p_corr

(chamberwise) parameter to reduce the dynamic pressure as initial guess for the next time level (UCVO)

See damping_p_corr .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · end_alias

end_alias

ending alias definition

See ALIAS .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · end_boundary_elements

end_boundary_elements

ending boundary elements definition

See BoundaryElements .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · end_construct

end_construct

ending construct variables definition

See ConstructClause .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · end_construct_atRestart

end_construct_atRestart

ending construct variables definition (only) at restart

See ConstructClause .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · end_curve

end_curve

ending curve definition

See Curves .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · end_equation

end_equation

ending equation definition

See Equations .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · end_loop

end_loop

ending loop definition

See Loops .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · end_pointcloud

end_pointcloud

ending point cloud definition

See ReadInPointCloud .
See ReadInPointCloud .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · end_save

end_save

end of begin_save{ environment

See begin_save{

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ ·

end_selection

end_selection

ending selection definition

See Selection for details.

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · end_timestepfile

end_timestepfile

end of timestep/integration file environment

See TimestepFile .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · eps_T

eps_T

precision in the breaking criterion for the linear systems of temperature (UCVO)

See eps_T .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · eps_p

eps_p

precision in the breaking criterion for the linear systems of pressure (UCVO)

See eps_p .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · eps_phyd

eps_phyd

precision in the breaking criterion for the linear systems of hydrostatic pressure (UCVO)

See eps_phyd .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · eps_v

eps_v

precision in the breaking criterion for the linear systems of velocity (UCVO)

See eps_v .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · max_N_stencil

max_N_stencil

maximum number of neighbor points accepted for stencil computation and numericss (UCVO)

See max_N_stencil .

---

ord_eval

define approximation order for refill points (UCVO)

See ord_eval .

---

ord_gradient

(chamberwise) approximation order of the gradient operators (UCVO)

See ord_gradient .

---

ord_laplace

define approximation order of the Laplace operators (UCVO)

See ord_laplace .

---

radius_hole

relative allowed hole size (UCVO)

See radius_hole .

---

rel_dist_bound

relative distance of neighboring points at boundaries (UCVO)

See rel_dist_bound .

---

restart_additionalBE

include additional boundary elements file during restart

See ExchangeBEOnRestart .

---

restart_copying

restart_copying

copy alias definition for additional boundary elements during restart

See ExchangeBEOnRestart .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · restart_file

restart_file

Define file name of restart files

For an explanation of this option see RestartPath .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · restart_path

restart_path

Define path to restart files

For an explanation of this option see RestartPath .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · restart_step_size

restart_step_size

define after how many time cycles a restart file has to be generated

See RestartStepSize .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · restart_toberemoved

restart_toberemoved

remove pre-restart boundary elements during restart

See ExchangeBEOnRestart .

MESHFREE · InputFiles · USER_common_variables · __overview_of_syntax_elements__ · restartnewBE_filling

restartnewBE_filling

(chamberwise) parameter to control filling of new boundary elements upon restart (UCVO)

See restartnewBE_filling .

MESHFREE · InputFiles · USER_common_variables · include_Ucv{

3.1.37. include_Ucv{

include a file in UCV-format

```
...
include_Ucv{ FileName }
...
```

List of members:

parameters{	use include-UCV-file as subroutine and define the parameters

parameters{

use include-UCV-file as subroutine and define the parameters

```
...
include_Ucv{ FileName.dat } parameters{ $SomeAcronym_local$=$SomeAcronym_global$ , ... ,
&SomeALias_local&=&SomeALias_global&}
...
```

MESHFREE will replace all occurances of **$SomeAcronym_local$** ... **&SomeALias_local&** in the local file FileName.dat by their global representations **$SomeAcronym_global$** ... **&SomeALias_global&** who need to have a meaning in the "calling" UCV-file.
The replacement is performed during read-in, the files on disk are not affected.

Special hint:
Actually, the parameters{ }-functionality works simply by character or string replacement during read-in of the file FileName.dat
In that aspect, one can also omit the control characters "$" or "&".
However, if you want to use a specific numerical value for a quantity, that is used as &AliasOfSomeName& in FileName.dat,
then one would have to write

```
...
include_Ucv{ FileName.dat } parameters{ ..., &AliasUsedInFileName&=12.34567879, ... }
...
```


3.2. common_variables

input file for development and debugging purposes

The file common_variables.dat (CV) contains mostly numerical parameters that are used for development and debugging. Currently, efforts are made to reduce the number of mandatory parameters to a minimum (aim is none).

Note:

- Some CV-parameters can also be set in USER_common_variables (UCV). The UCV-definition is dominant and overwrites the
  CV-definition (see warnings file in the simulation folder). The Ucv parameters can be found here .
- Some CV-parameters can be set chamberwise, which can be necessary for multi-phase simulations. If such a parameter is
  not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

List of members:

additionalPoint_approximation	(experimental) in EULERIMPL and EULEREXPL setting
AdvancedFreeSurfaceAtTimeStep	advanced checking of free surface point (Delaunay based) starting at which time cycle
alpha_O1	
AMFPMJ_CommonAdministrationDirectory	define a directory that MESHFREE uses for synchronization of multiple MESHFREE jobs
APPROXIMATENEWPOINTS_HowToApproximateKEPS	
APPROXIMATENEWPOINTS_SeparateInteriorAndBoundaryPoints	
APPROXIMATENEWPOINTS_SeparateInteriorBoundary	
BCGSL_ell	
BE_CleanUp_STL	choose whether and when to clean up STL geometries
BE_COLLAPSE_collapsebeforeflip	
BE_COLLAPSE_specifycollapse	
BE_COLLAPSE_tocollapse	
BE_COLLAPSE_tolerance	
BEmap_DefaultValue	Default value of BE_MAP (CV)
BETA_FOR_LIMITER	parameter for controlling the Sweby limiter
BND_alpha	
BND_beta	
BND_gamma	
BND_SearchTreeAdministration_NbTimeStepsUntilFirstSkip	skip the construction of the boundary element search tree after this many time cycles
BND_SearchTreeAdministration_RefreshTreeAfterHowManyCycles	refresh the boundary element search tree after this many time cycles
BUBBLE_DoTheManagement	(chamberwise) switch regarding bubble analysis (CV)
BUBBLE_EdgeValue	(experimental) edge value limit for the detection of open edges
BUBBLE_EnforceAveragePressure	fix average pressure for all bubbles (CV)
BUBBLE_fac_pHydrostatic	(experimental) numerical relaxation parameter for pHydrostatic in case of bubbles
BUBBLE_pOffset	define offset pressure for bubble pressure-on-volume analysis (CV)
BUBBLE_UseTopologyConstraint	(chamberwise) parameter to use topology analysis for bubble-volume computation
CLUSTER_Size	
CoarsenGeometry	

COEFF_Abaqus_H	factor for abaqus mesh interpolation
COEFF_dt	(chamberwise) factor for computation of time step size (CV)
COEFF_dt_coll	time step criterion from interaction model (DROPLETPHASE only) (CV)
COEFF_dt_d30	time step criterion depending on %ind_d30% (DROPLETPHASE only) (CV)
COEFF_dt_Darcy	define the virtual time step size for applications with Darcy (Brinkman) term (CV)
COEFF_dt_free	(experimental) factor for exaggerated movement of the free surface (CV)
COEFF_dt_SurfaceTension_A	time step criterion for surface tension, parameter A (CV)
COEFF_dt_SurfaceTension_B	time step criterion for surface tension, parameter B (CV)
COEFF_dt_SurfaceTension_C	(experimental) time step criterion for surface tension, parameter C (CV)
COEFF_dt_virt	(chamberwise) scaling factor for the virtual time step size (CV)
COEFF_Iopp_Repair	
COEFF_mue	scaling factor for numerical viscosity (CV)
COEFF_p_divV	factor to switch on and control the p*div(v) term in temperature equation
COEFF_penalty	
COMP_AddBoundaryParticles	this parameter rules how to add points at regular boundaryies (walls, inflow etc)
COMP_AddInteriorParticles	
COMP_AdjustEtaEff	invoke more stability by controlling the total viscosity
COMP_CheckConservationDuringOrganization	
COMP_CosEdgeAngle	(chamberwise) parameter to identify edges in geometry (CV)
COMP_CosOpenEdge	specify how the boundary continues at an open edge
COMP_DeflationLevel	
COMP_DoOrganizeOnlyAfterHowManyCycles	do the point cloud organization only after how many time cycles (CV)
COMP_DoOrganizePointsUntil	
COMP_DropletphaseSubcycles	switch for subcycling in DROPLETPHASE (CV)
COMP_DropletphaseWithDisturbance	disturbance for DROPLETPHASE (CV)
COMP_dt_indep	parameter to switch on independent time stepping for two-phase LIQUID simulations with v-- and vp- (CV)
COMP_EtaGrad_Version	define the way of numerically modelling the property-times-gradient operator
COMP_evoid	
COMP_facSmooth_Eta	parameter for weight kernel definition for smoothing of viscosity (CV)
COMP_FastBoundaryRefill	

COMP_FillEdges	fill additional points to the edges of an inflow area
COMP_GradtEtaGrad_Version	define the way of numerically modeling the diffusion operator
COMP_HydrostaticPressure	
COMP_IsolatedParticles_MinNbOfInteriorNeigh	minimum number of interior neighbors a points should have
COMP_IsolatedParticles_MinNbOfNeigh	(chamberwise) parameter for the minimum number of (total) neighbors a points should have
COMP_ManifoldContacts	(Experimental) Determines whether or not contact should be checked (manifold phase only)
COMP_MaxSubCycle	
COMP_MinSubCycle	
COMP_nbSmooth_Darcy	
COMP_nbSmooth_Eta	number of smoothing cycles for effective and total viscosity (CV)
COMP_nbSmooth_pCorr	smooth heat conductivity
COMP_OppositePoints_NoFreeSurface	
COMP_RandomizedFilling	
COMP_ReduceSn	
COMP_RemeshBoundary	parameter to control remeshing of IGES-files (CV)
COMP_RemoveBoundaryParticles	
COMP_RemoveInteriorParticles	
COMP_SharedMemoryForBE	turn on use of MPI shared memory for boundary geometry if available
COMP_SharedMemoryForGT2	turn on use of MPI shared memory for GEOTREE2 if available
COMP_SkipHighVelocities	for how many consecutive cycles a corrupt solution of velocity is accepted, before MESHFREE stops
COMP_SortBEintoBoxes_Version	version how to organize/prepare boundary elements for efficient computation
COMP_StressRelaxAtFreeAndSlipSurface	
COMP_TimeCheck	switch on time measurements for the main tasks of MESHFREE
COMP_TypeSmooth_Eta	type for smoothing of viscosity (CV)
COMP_TypeSmooth_Rho	type for smoothing of density (CV)
COMP_WettingAngleVariante	How to incorporate the contact angle between free surface and wall
COMP_WettingAngleWeight	
CompDistToBoundary_Acc	threshold of distance until which the distance to different BE is treated as equal

CompDistToBoundary_EffectiveSearchRadius	
compute_FS	(chamberwise) switch to compute free surfaces (CV)
compute_LAYER	(experimental) influence to Neighbor Filtering over Layers
compute_phase_boundary	(obsolete) invoke detection of interface connections (CV)
CONTROL_DirectTesting	Instead of launching the computation, MESHFREE goes into a separate testing branch for different tasks
CONTROL_DirectTesting_Param1	additional parameter for the testing environment
CONTROL_DirectTesting_Param2	additional parameter for the testing environment
CONTROL_DirectTesting_Param3	additional parameter for the testing environment
CONTROL_StopAfterReadingGeometry	stops the MESHFREE program after geometry is read (CV)
CONTROL_writeUcvLines	write out the Ucv-lines read during startup (debugging feature)
correct_CONS	
correction_pressure	
damping_p_corr	(chamberwise) parameter to reduce the dynamic pressure as initial guess for the next time level (CV)
Darcy_PrimaryDirection	
Darcy_PrimaryDirectionFactor	
DEBUG_Check_CCOR	generate control writeout for correction pressure
DEBUG_Check_PDYN	generate control writeout for dynamic pressure
DEBUG_Check_PHYD	generate control writeout for hydrostatic pressure
DEBUG_Check_VELO	generate control writeout for velocity
DEBUG_DefaultRescue	
DEBUG_GeneralParameter	General list of debug parameters at the developpers disposal
DEBUG_SHM_MPIwindow	GASDYN parameter for FPM2
DEL_rel_dist_shuffle	
delaunay_reduction	switch for delaunay reduction procedure
delta_uw	
delta_uw_bp	
DIFFOP_ConsistentGradient	consistent gradient in the sense d/dn = n*grad (CV)
DIFFOP_gradient	type of least squares approximation stencils for gradients
DIFFOP_kernel_Gradient	(chamberwise) factor for the weight kernel for the least squares approximation stencils for gradients (CV)

DIFFOP_kernel_Laplace	(chamberwise) factor for the weight kernel for the least squares approximation stencils for the Laplacian (CV)
DIFFOP_kernel_Neumann	(chamberwise) factor for the weight kernel for the least squares approximation stencils for Neumann operators (CV)
DIFFOP_kernel_Transport	(chamberwise) factor for the weight kernel for the least squares approximation stencils for the transport operators (CV)
DIFFOP_laplace	type of least squares approximation stencils for the Laplacian (CV)
DIFFOP_Neumann_ExcludeBND	(chamberwise) parameter to exclude boundary points from the neighborhood for the computation of the Neumann operators (CV)
DIFFOP_PPI_Gradient	
DIFFOP_PPI_Laplace	
DIFFOP_PPI_Neumann	
DIFFOP_Version	version of least squares operators
DIFFOP_WeightReductionInCaseOfDeactivation	(chamberwise) parameter to reduce the weight of a neighbor point in case of deactivation (CV)
dist_aip	initial relative distance to boundary of a newly injected MESHFREE point (aip = add injected points)
dist_between_phases	
dist_FS_from_BND	define hole size for the free surface detection
dist_FS_new_part	
dist_LayerThickness	minimal thickness for degenerated 3D phase
dist_merge_opp_face	
dist_rab	relative allowed minimum distance of MESHFREE points to boundary (rab = remove at boundary)
dist_rip	relative allowed minimum distance between MESHFREE points (rip = remove interior points)
DP_UseOnlyRepulsiveContactForce	switch regarding attractive forces in spring-damper model (CV)
ELASTOPLASTIC_FadeOut_divS_gradP	
EPS_global	
eps_p	precision in the breaking criterion for the linear systems of pressure (CV)
eps_phyd	precision in the breaking criterion for the linear systems of hydrostatic pressure (CV)
eps_T	precision in the breaking criterion for the linear systems of temperature (CV)
eps_v	precision in the breaking criterion for the linear systems of velocity (CV)
FLIQUID_AssignPenalties_EpsilonP	vp- coupled linear system: lower bound for ratio between pressure and velocity entries, PRESSURE EQUATION

FLIQUID_AssignPenalties_EpsilonV	vp- coupled linear system: upper bound for ratio between velocity and pressure entries, VELOCITY EQUATION
FLIQUID_ConsistentPressure_CoeffJOKER	TEMPORARY: factor to study consistent pressure version 2
FLIQUID_ConsistentPressure_CoeffMM	TEMPORARY: factor to study consistent pressure version 2
FLIQUID_ConsistentPressure_CoeffNN	TEMPORARY: factor to study consistent pressure version 2
FLIQUID_ConsistentPressure_CoeffTT	TEMPORARY: factor to study consistent pressure version 2
FLIQUID_ConsistentPressure_CoeffWEIGHT	TEMPORARY: factor to study consistent pressure version 2
FLIQUID_ConsistentPressure_UseDivV	(chamberwise) parameter to use numerical approximations of div(v) in direct computation of dynamic pressure (i.e. consistent pressure)
FLIQUID_ConsistentPressure_Version	version how to compute the consistent pressure (CV)
FOFTLIQUID_AdditionalCorrectionLoops	additional velocity correction loops (CV)
FPM_LICENSE_FILE	overwrite the environment variable
GASDYN_CorrectEnergy	correct total energy in GASDYN application
GASDYN_CorrectMass	correct mass in GASDYN application
GASDYN_FPM2_alpha	GASDYN parameter for FPM2
GASDYN_FPM2_beta	GASDYN parameter for FPM2
GASDYN_p_gain	limit the pressure gain in GASDYN-applications
GASDYN_p_loss	limit the pressure drop in GASDYN-applications
GASDYN_r_gain	limit the density gain in GASDYN-applications
GASDYN_r_loss	limit the density drop in GASDYN-applications
GASDYN_T_gain	limit the temperature gain in GASDYN-applications
GASDYN_T_loss	limit the temperature drop in GASDYN-applications
GASDYN_Upwind2ndOrder	DEPRECATED!!! (GASDYN parameter for FPM1)
GASDYN_Upwind_Lbeta	(chamberwise) GASDYN parameter for FPM1 and FPM3
GASDYN_Upwind_Lgamma	(chamberwise) GASDYN parameter for FPM1 and FPM3
GASDYN_UpwindOffset	(chamberwise) GASDYN parameter for FPM1
GASDYN_Version	(chamberwise) GASDYN parameter to choose FPM1 or FPM2
GEOTREE2_BND_FinalBoxDimension	relative size extent of GEOTREE2 leaves
GEOTREE2_BND_FinalBoxSize	number of triangles in a GEOTREE2 leave

GEOTREE2_EstablishCON_Version	parameter for the bintree-search of the neighborhood of MESHFREE points
GEOTREE2_FinalBoxSize	parameter for the bintree-search of the neighborhood of MESHFREE points
GEOTREE2_IntListMargin	parameter for the bintree-search of the neighborhood of MESHFREE points
GEOTREE2_MaximumBoxSize	parameter for the bintree-search of the neighborhood of MESHFREE points
GEOTREE2_SizeOfSearchBox	parameter for the bintree-search of the neighborhood of MESHFREE points
GLOBAL_eps_mass	
GLOBAL_eps_momentum	
GLOBAL_N_iterations	
HowToTreatPause	
iFPM_process_ID	give a maximum 16-digit MESHFREE process ID
IGES_Accuracy	relative accuracy for consistency checks of IGES-faces (CV)
IGES_HealCorruptFaces	allow a certain depth of healing triangulation of IGES faces by refinement (CV)
initial_particles	
int_BND_part_add	boundary point addition interval
int_BND_part_remove	boundary point removal interval
int_part_add	interior point addition interval
int_part_cross_BND	
int_part_remove	interior point removal interval
int_part_smooth	
INTEGRATION_ReopenTimestpFilesAfterHowManyCycles	*.timestep-Files close and reopen again after how many cycles (debug reasons)
IS_GPU	
ISOLATEDPOINTS_ClusterOnResultingVolume	threshold to cluster two isolated points into one
ISOLATEDPOINTS_ProduceVolumePackage	threshold to turn isolated points into volume packages
ITERATION_EstimatedFutureStressTensor	
ITERATION_evoid	
ITWMESI_PressureMapping_Filter	coupling ITWMESI filter for mapping the pressure solution to the boundary elements
ITWMESI_PressureMapping_WeightPdyn	coupling ITWMESI weight for mapping dynamic pressure
ITWMESI_PressureMapping_WeightPhyd	coupling ITWMESI weight for mapping hydrostatic pressure

ITWMESI_ShearForceMapping_BasedOnStresses	coupling ITWMESI: decide whether the shear forces be projected as stress values (N/m^2) or as forces (N)
ITWMESI_ShearForceMapping_Filter	coupling ITWMESI filter for mapping the shear force solution to the VPS boundary elements
ITWMESI_ShearForceMapping_Weight	coupling ITWMESI weight for mapping the shear forces
ITWMMpCCI_PressureMapping_WeightPdyn	coupling ITWMESI weight for mapping dynamic pressure
ITWMMpCCI_PressureMapping_WeightPhyd	coupling ITWMESI weight for mapping hydrostatic pressure
kind_of_method	
LIMITER	slope limiter for controlling numerical diffusion in MUSCL-reconstruction scheme in EULERIMPL and EULEREXPL setting
LINEQN_scaling	choose the way how to scale/normalize the linear systems (CV)
LINEQN_solver	linear solver to be used for the coupled vp- or v-- system (CV)
LINEQN_solver_ScalarSystems	linear solver to be used for the scalar systems like pressure, temperature, etc. (CV)
MASS_correction_DivergenceVelocity	Mass Correction for weakly compressible flow problems
max_N_stencil	maximum number of neighbor points accepted for stencil computation and numerics (CV)
max_N_stencil_INTERIOR	max. number of neighbors accepted for stencil computation and numerics only for interior pooints
MaximumNumberOfPointsDuringComputation	
MEMORIZE_ResetReadFlag	reset frequency for MEMORIZE_Read flag (CV)
MESHFREE_LICENSE_FILE	overwrite the environment variable
MPI_CommunicationMethod	
MPI_ExcludeDirectionFromBisection	
MPI_WeightingMethodForBisection	how to give weights to points for the MPI-bisection process
MULTIGRID_CutOff	
N_addvar	definition of the number of %ind_addvar% to be used (legacy code)
Nb_InflowLayers	Nb_InflowLayers
NB_OF_ACCEPTED_REPETITIONS	number of permitted repetitions of substep in EULERIMPL setting
NB_POINTS_BC_HEAT_EQUATION_1D	number of points for 1D heat equation for temperature boundary condition
NEIGHBOR_filter_level	

NEIGHBOR_FilterMethod	choose how to exclude neighbors from MESHFREE points at critical geometry parts
nue	
OBJ_ConvertQuadToTria	convert quads into triangles upon read-in
OPTIMIZATION_InitialGuessOfVi_Fast	
ord_eval	define approximation order for refill points (CV)
ord_gradient	(chamberwise) approximation order of the gradient operators (CV)
ord_laplace	define approximation order of the Laplace operators (CV)
ORGANIZE_ActivateBNDpoints_Version	define version number for the boundary point activation
ORGANIZE_BE_ClusterNodesPoints_Version	define version number for clusterig of geometry node points after geometry is read in from file (such as stl-files)
ORGANIZE_BringNewPointToFreeSurface	define maximum distance a newly created point at the free surface can be moved in order to perfectly fit the free surface
ORGANIZE_CheckAllPointsForFreeSurfaceUntilTimeStep	consider all points as candidates for free surface until a given time step
ORGANIZE_CheckFreeSurface_Version	define version number for the free-surface-check
ORGANIZE_CheckPointsAtFS_PerformPreCheck	invoke additional algorithm in order to find candidates for free surface detection
ORGANIZE_DevelopperCheck_Version	version of the debugging routine ORGANIZE_DevelopperCheck
ORGANIZE_DistanceToBoundary_Version	define version number for distance-to-boundary computations
ORGANIZE_ForceInsideCheckForAllParticles	inside-check for all MESHFREE points
ORGANIZE_ForceInsideCheckForNewParticles	inside-check for new MESHFREE points
ORGANIZE_ForceTouchCheckAtWalls	touch-check for MESHFREE points at walls
ORGANIZE_FuzzyMPIFilling	(chamberwise) parameter to allow MPI processes to fill points outside their own domain
ORGANIZE_OppositePoints_Version	define version number for detecting points of the other phase to be coupled (opposite points)
ORGANIZE_PreAllocationSize	define version number for distance-to-boundary computations
ORGANIZE_PSTOneReductionStep_Version	version how to reduce MESHFREE points if they come to close to each other

ORGANIZE_PSTOneRefillStep3_UseFromWhichTime	use the new implementation of PST_OneRefillStep_3 from which time
ORGANIZE_PSTOneRefillStep3_UseFromWhichTimeStep	use the new implementation of PST_OneRefillStep_3 from which time step
ORGANIZE_QualityCheck_ListNbOfNeighbors	number of neighbors per point for which the quality check has to be performed
ORGANIZE_ReducedFillingOfWalls	(chamberwise) parameter for reduced filling of boundaries marked as walls
ORGANIZE_ReducedFillingOfWallsIgnoreNofillForStartup	
ORGANIZE_RefillOnlyForActiveBoundaryParticles	(chamberwise) parameter to trigger the point refilling procedure along the boundary only for active boundary points
ORGANIZE_ToleranceForGapAnalysisOfRegularBoundary	
ORGANIZE_USER_update_boundary_particles_Version	version of USER_update_boundary_particles.f90 to be used
PARTPHASE_elasticity	
PARTPHASE_friction	
PARTPHASE_wall_friction	
pBubble_Offset	define offset pressure for bubble pressure-on-volume analysis
PHASE_distinction	invoke detection of interface connections (CV)
PointDsplMethod	(experimental) Choice among different ways to move points in Lagrangian framework (CV)
prec_seek_holes	number of test points created for hole search
pure_TRANSPORT	(experimental) choice of spatial discretization scheme for transport terms in EULERIMPL and EULEREXPL setting
QUICKVIEW_SaveHowOften	
QUICKVIEW_VariableList	
QUICKVIEW_Version	
QUICKVIEW_WhichParticles	
radius_hole	relative allowed hole size (CV)
rel_dist_bound	relative distance of neighboring points at boundaries (CV)
rel_dist_edge	relative distance of neighboring points at edges of the geometry
RepairGeometry	enforce clustering of geometry nodes upon read-in (CV)
RepresentativeMass_iData	(chamberwise) parameter for the RepresentativeMass algorithm (CV)
RESTART_useSTREAMfile	use the STREAM inp/output for restart files

restartnewBE_filling	(chamberwise) parameter to control filling of new boundary elements upon restart (CV)
RIGIDBODY_TimeIntegrationDamping	Numerically damping of the time integration
RIGIDBODY_TimeIntegrationPPI	Tichonov-regularization parameter for rigid bodies with links or intersections
RIGIDBODY_TimeIntegrationVersion	choose time integration version (still experimental)
RIGIDBODY_UseCollisionModel	switch on the collision model for rigs bodies (rigid-wall and rigid-rigid)
SAMG_Setupreuse	accelerates SAMG solver for quasi-stationary point clouds (CV)
SAVE_ASCII_split	splits ASCII output files if larger than 2GB
SAVE_atEndOfTimestep	choose to save data for visualization at the end of time steps instead of at the start (CV)
SAVE_PrecisionTimestepFile	choose the precision (number of digits) for values in the timestep file (CV)
SAVE_QuickView	
SaveRestartOnInit	
SCAN_ClustersOfConnectivity	(chamberwise) switch on cluster checking of MESHFREE point cloud by neighborhood connectivity (CV)
SIGNAL_LaunchComputationalSteering	Switch between the two options of computational steering
SimCut	(chamberwise) parameter to stop filling of geometry by MESHFREE points after a certain number of filling cycles
SimCutBoundary	(chamberwise) parameter to stop filling of boundary by MESHFREE points after a certain number of filling cycles
SkipMarkingPointsLayer2	(experimental) switch for marking the second layer near the boundary in EULERIMPL setting
smooth_BND_movement	
smooth_BND_normal	
smooth_FS	
smooth_FS_SurfaceTension	
SOLVEV_N_iterations	
SPAI_eps	
SPAI_first	
SPAI_maxentries	
SPAI_maxiter	
SPAI_precond_method	
SPAI_precond_preparation	
SPAI_restart	

SPAI_smax	
SpecialBNDtreatmentEULERIMPL	(experimental) switch for special boundary treatment for MUSCL reconstruction in EULERIMPL scheme
SPM_matrix_times_vector_Version	version for the matrix-times-verctor operations for sparse linear systems
SPM_N_iterations	maximum number of iterations in linear system solver
SPM_Regularization_Epsilon	adjust numerical parameter epsilon for the matrix regularizations
SPM_Regularization_Type	regularization type if all boundaries are Neumann-type
StencilOrderReductionNearBND_forEULERIMPL	(experimental) switch for order reduction of x,y,z-derivative stencils in EULERIMPL setting
STRESSTENSOR_NumberSubcycles	
STRESSTENSOR_Variante	version of stress tensor time integration (CV)
STRESSTENSOR_Variante_Factor	factor in stress tensor time integration wrt the shear modulus (CV)
stretch_search	
SUBSTEPS_EXPL	number of explicit substeps for solving TRANSPORT part in EULEREXPL setting
SUBSTEPS_IMPL	number of implicit substeps with constant time step size in EULERIMPL setting
SURFACETENSION_FacSmooth	
SURFACETENSION_NbSmooth	
SurfaceTesselationActiveBoundary_cRadius	radius of the basic disc for the surface tesselation cells on active boundary, including free surface, excluding inactive points
SurfaceTesselationRegularBoundary_cRadius	radius of the basic disc for the surface tesselation cells on regular boundary
time_integration_expl	order of explicit time integration scheme in EULEREXPL setting
time_integration_impl	order of implicit time integration scheme in EULERIMPL setting
time_integration_impl_solve_v	order of implicit time integration scheme for velocity only (EULERIMPL)
time_step_gain	relative amount by which new timestep size can increase at maximum compared to old timestep size
time_step_loss	relative amount by which new timestep size can decrease at maximum compared to old timestep size (adaptive timestep size)
TIMECHECK_Level	time check only up to a given level
TOL_keps	(control of time step size) error tolerance for computing the k-epsilon model using SDIRK2 method in EULERIMPL setting
TOL_T	(control of time step size) error tolerance for computing the temperature using SDIRK2 method in EULERIMPL setting
TOL_v	(control of time step size) error tolerance for computing the velocity using SDIRK2 method in EULERIMPL setting

TRANSPORT_ODE_fct_evaluation	(experimental) switch for additional function evaluation within the implicit time integration scheme in EULERIMPL setting
tryMaikesTriangulation	
turn_down_BND_order	(chamberwise) parameter to automatically reduce the approximation order of a boundary point
use_BubbleManagement	(chamberwise) switch regarding bubble analysis
UseBoxSystemVersion	force MESHFREE to use a certain tree algorithm for the MESHFREE point neighbor search
USER_curve_interpol_cache	turn on caching in USER_curve_interpol_3
V00_SmoothDivV	Chorin projection: smooth the local values of div(v) before going into the correction pressure computation (CV)
vel_dim	
VOLUME_correction	(chamberwise) parameter to correct volume by GLOBALLY adjusting the divergence of velocity term (CV)
VOLUME_correction_FreeSurface	(chamberwise) parameter to correct volume by tiny global lifting of the free surface (CV)
VOLUME_correction_local	(chamberwise) parameter to correct volume by LOCALLY adjusting the divergence of velocity term due to representative mass balance (CV)
VOLUME_correction_ResetOnRestart	(experimental) resets the volume correction quantities of each chamber to the current values
VP0_VelocityCorrection	invoke velocity correction based on correction pressure (%ind_c%) for vp- solver (CV)
WallLayer	Turbulent wall layer thickness
WARNINGS_BND_Integrate	flag controlling the warnings in BND_Integrate
WARNINGS_USER_parse_IsConditionStringFulfilledByBE	flag controlling the warnings in USER_parse_IsConditionStringFulfilledByBE
WARNINGS_USER_parse_IsConditionSubstringFulfilledByBE	flag controlling the warnings in USER_parse_IsConditionSubstringFulfilledByBE
WhichIndexingMethod	
who_am_I	
write_debug	
WRITEOUTPUT_Level_Organize	

3.2.1. AMFPMJ_CommonAdministrationDirectory

define a directory that MESHFREE uses for synchronization of multiple MESHFREE jobs

```
AMFPMJ_CommonAdministrationDirectory = '/home/WhoAmI/FPMsynchro/' # for example Linux
AMFPMJ_CommonAdministrationDirectory = 'D:\home\WhoAmI\FPMsynchro\' # for example in WINDOWS
```

puts a file with the name "FPM_ID=??????????????????????" into the directory given in AMFPMJ_CommonAdministrationDirectory .
This file contains information about job index, requested resources as well as start time and last report time.
The last report time is updated each time MESHFREE starts a new time cycle. This information is used to decide, which MESHFREE
has now priority and which one has to sleep until ressources are available.

3.2.5. AdvancedFreeSurfaceAtTimeStep

advanced checking of free surface point (Delaunay based) starting at which time cycle

```
AdvancedFreeSurfaceAtTimeStep = 0
```

Default: AdvancedFreeSurfaceAtTimeStep = 3

3.2.6. BCGSL_ell

Choose parameter l for linear solver BiCGstab(l). Default value is 4.

```
LINEQN_solver = 'BCGL'
BCGSL_ell = 3
```


3.2.7. BETA_FOR_LIMITER

parameter for controlling the Sweby limiter

```
BETA_FOR_LIMITER = 1.5
```

Default: BETA_FOR_LIMITER = 1.9

It controls the numerical diffusion of the Sweby limiter.

- BETA_FOR_LIMITER = 1.0 -> yields Minmod limiter
- BETA_FOR_LIMITER = 2.0 -> yields Superbee limiter

See LIMITER .

3.2.12. BE_CleanUp_STL

choose whether and when to clean up STL geometries

STL files contain the node coordinates for each triangle, so node points that belong to multiple triangles are saved repeatedly. During cleanup, node points are clustered if they are very close to each other and afterwards, degenerate triangles are deleted. If there are other geometry items have been loaded before (e.g. via include{} or as PlainBoundaryElements), then this can lead to problems.

Thus, by default, the STL clean up is only done as long as no other geometry has been loaded. To change the behavior, adapt this parameter.

```
BE_CleanUp_STL = 0 # never clean up STL geometries
BE_CleanUp_STL = 1 # only clean up STL geometries
# as long as no other geometries have been included
BE_CleanUp_STL = 2 # default: always clean up STL geometries
# (may lead to problems in some cases)
```

MESHFREE · InputFiles · common_variables · BEmap_DefaultValue

3.2.13. BEmap_DefaultValue

Default value of BE_MAP (CV)

See BEmap_DefaultValue . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · BND_SearchTreeAdministration_NbTimeStepsUntilFirstSkip

3.2.14. BND_SearchTreeAdministration_NbTimeStepsUntilFirstSkip

skip the construction of the boundary element search tree after this many time cycles

```
BND_SearchTreeAdministration_NbTimeStepsUntilFirstSkip = 10
```

Default: 10000000
Additional parameter (optional) to be set when defining COMP_SortBEintoBoxes_Version .
In the first BND_SearchTreeAdministration_NbTimeStepsUntilFirstSkip MESHFREE time cycles, the boundary element search tree is established at the beginning
of the time cycle. After this, the period of establishing the search tree is given by the variable BND_SearchTreeAdministration_RefreshTreeAfterHowManyCycles .

MESHFREE · InputFiles · common_variables · BND_SearchTreeAdministration_RefreshTreeAfterHowManyCycles

3.2.15. BND_SearchTreeAdministration_RefreshTreeAfterHowManyCycles

refresh the boundary element search tree after this many time cycles

```
BND_SearchTreeAdministration_RefreshTreeAfterHowManyCycles = 50
```

Default: 1
Additional parameter (optional) to be set when defining COMP_SortBEintoBoxes_Version .
See also BND_SearchTreeAdministration_NbTimeStepsUntilFirstSkip .

MESHFREE · InputFiles · common_variables · BUBBLE_DoTheManagement

3.2.19. BUBBLE_DoTheManagement

(chamberwise) switch regarding bubble analysis (CV)

See BUBBLE_DoTheManagement . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · BUBBLE_EdgeValue

3.2.20. BUBBLE_EdgeValue

(experimental) edge value limit for the detection of open edges

This parameter is currently experimental.

MESHFREE · InputFiles · common_variables · BUBBLE_EnforceAveragePressure

3.2.21. BUBBLE_EnforceAveragePressure

fix average pressure for all bubbles (CV)

See BUBBLE_EnforceAveragePressure . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · BUBBLE_UseTopologyConstraint

3.2.22. BUBBLE_UseTopologyConstraint

(chamberwise) parameter to use topology analysis for bubble-volume computation

```
BUBBLE_UseTopologyConstraint = 1
```

Default: BUBBLE_UseTopologyConstraint = 0

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

If the volume is positive, it is a true bubble, if negative, it is a droplet. Sometimes, the MESHFREE point configuration is disadvantageous, such that the measured volume might change the sign. However, if there was no topology change, i.e. no splitting or merging, the pressure and volume changes are ignored for the current time step, if the sign of mesured volume flipped.

BUBBLE_UseTopologyConstraint = 1 : topology check for both way (minus->plus and plus->minus)
BUBBLE_UseTopologyConstraint = 2 : topology check for bubble (plus->minus)
BUBBLE_UseTopologyConstraint = 3 : topology check for droplet (minus->plus)

MESHFREE · InputFiles · common_variables · BUBBLE_fac_pHydrostatic

3.2.23. BUBBLE_fac_pHydrostatic

(experimental) numerical relaxation parameter for pHydrostatic in case of bubbles

This parameter is currently experimental.

3.2.24. BUBBLE_pOffset

define offset pressure for bubble pressure-on-volume analysis (CV)

See BUBBLE_pOffset . Definitions in USER_common_variables are dominant.

3.2.26. COEFF_Abaqus_H

factor for abaqus mesh interpolation

```
COEFF_Abaqus_H = 1.0
```

Default: COEFF_Abaqus_H = 1.0

Defines the radius in relative to the smoothing length that is used for abaqus mesh interpolation.

If COEFF_Abaqus_H = 1.0, all MESHFREE points within a perimeter of size H arround an abaqus node or midpoint are used for interpolation
of information of the simulation onto the abaqus mesh.

3.2.28. COEFF_dt_Darcy

define the virtual time step size for applications with Darcy (Brinkman) term (CV)

See COEFF_dt_Darcy . Definitions in USER_common_variables are dominant.

3.2.29. COEFF_dt_SurfaceTension_A

time step criterion for surface tension, parameter A (CV)

See COEFF_dt_SurfaceTension_A . Definitions in USER_common_variables are dominant.

3.2.30. COEFF_dt_SurfaceTension_B

time step criterion for surface tension, parameter B (CV)

See COEFF_dt_SurfaceTension_B . Definitions in USER_common_variables are dominant.

3.2.31. COEFF_dt_SurfaceTension_C

(experimental) time step criterion for surface tension, parameter C (CV)

See COEFF_dt_SurfaceTension_C . Definitions in USER_common_variables are dominant.

3.2.32. COEFF_dt

(chamberwise) factor for computation of time step size (CV)

See COEFF_dt . Definitions in USER_common_variables are dominant.

3.2.33. COEFF_dt_coll

time step criterion from interaction model (DROPLETPHASE only) (CV)

See COEFF_dt_coll . Definitions in USER_common_variables are dominant.

3.2.34. COEFF_dt_d30

time step criterion depending on %ind_d30% (DROPLETPHASE only) (CV)

See COEFF_dt_d30 . Definitions in USER_common_variables are dominant.

3.2.35. COEFF_dt_free

(experimental) factor for exaggerated movement of the free surface (CV)

See COEFF_dt_free . Definitions in USER_common_variables are dominant.

3.2.36. COEFF_dt_virt

(chamberwise) scaling factor for the virtual time step size (CV)

See COEFF_dt_virt . Definitions in USER_common_variables are dominant.

3.2.37. COEFF_mue

scaling factor for numerical viscosity (CV)

See COEFF_mue . Definitions in USER_common_variables are dominant.

3.2.38. COEFF_p_divV

*factor to switch on and control the p*div(v) term in temperature equation*

```
COEFF_p_divV = 1.0
```

Default: COEFF_p_divV = 0.0

This term is switched off by default because for incompressible flow problems it is zero.
But for weakly compressible flow problems it becomes more important so that it is needed to use COEFF_p_divV > 0 for switching on this term.

COEFF_p_divV = 1.0 means the term $p\,(\nabla \cdot \mathbf{v})$ is multiplied by one, thus the temperature equation

$$(\varrho\, c_v)\left(\frac{\partial T}{\partial t} + \mathbf{v}\cdot\nabla T\right) = \nabla\cdot(\lambda\,\nabla T) - p\,(\nabla\cdot\mathbf{v}) + \Phi$$

is used, whereby Φ is the dissipation function.

3.2.40. COMP_AddBoundaryParticles

this parameter rules how to add points at regular boundaryies (walls, inflow etc)

```
COMP_AddBoundaryParticles = (3,2) # refill every 3 time cycles, perform 2 filling iterations
```

Default: COMP_AddBoundaryParticles = (3,1)

first digit: refill of boundary points after this many time cycles
second digit: number of iteration loops performed for a refilling instance

OPTIONAL third digit:
COMP_AddBoundaryParticles = (3,1, 2)
third digit: perform more "aggressive" filling of boundaries in the vicinity of thin/degenerated liquid layers

3.2.42. COMP_AdjustEtaEff

invoke more stability by controlling the total viscosity

```
COMP_AdjustEtaEff = 8
```

Default: COMP_AdjustEtaEff = 1

$$\hat{\eta} := \max\left(\hat{\eta}, C\cdot\Delta t\ \|\boldsymbol{S}_s\|_{\text{Mises}}\right)$$

COMP_AdjustEtaEff defines the parameter C in the equation above.

3.2.44. COMP_CosEdgeAngle

(chamberwise) parameter to identify edges in geometry (CV)

See COMP_CosEdgeAngle . Definitions in USER_common_variables are dominant.

3.2.45. COMP_CosOpenEdge

specify how the boundary continues at an open edge

```
COMP_CosOpenEdge = -0.5
```

Default: COMP_CosOpenEdge = -0.3
If a boundary ends in an open edge (triangle edge with no topological connectivity to another, adjacent triangle), then with COMP_CosOpenEdge a virtual continuation direction of the boundary is given. The condition is cos(OpenEdge) = t*n,
where t is the virtual continuation tangent, and n is the boundary normal of the present boundary element.
A given point x, which projects onto the open edge, is considered to be inside if

(x-x_proj)*n/norm(x-x_proj) > COMP_CosOpenEdge
otherwise it is outside.

3.2.47. COMP_DoOrganizeOnlyAfterHowManyCycles

do the point cloud organization only after how many time cycles (CV)

See COMP_DoOrganizeOnlyAfterHowManyCycles . Definitions in USER_common_variables are dominant.

3.2.49. COMP_DropletphaseSubcycles

switch for subcycling in DROPLETPHASE (CV)

See COMP_DropletphaseSubcycles . Definitions in USER_common_variables are dominant.

3.2.50. COMP_DropletphaseWithDisturbance

disturbance for DROPLETPHASE (CV)

See COMP_DropletphaseWithDisturbance . Definitions in USER_common_variables are dominant.

3.2.51. COMP_EtaGrad_Version

define the way of numerically modelling the property-times-gradient operator

defines the way how to numerically model terms of the form $\eta\nabla u$ where η is a material property, that might have discontinuities.

```
COMP_EtaGrad_Version = %EtaGrad_Identity%
```

Default: COMP_EtaGrad_Version = %EtaGrad_Classical%
Possible options:

1.) %EtaGrad_Classical% :: estbalish the numerical operator exactly as it is: $\eta\nabla u = \eta_i \cdot \sum c_{ij}^{\nabla} u_j$

2.) %EtaGrad_Identity% :: estbalish the numerical operator as:

$$\begin{aligned}\eta\nabla u &= \eta\nabla(u-u_0)\\ &= \nabla\left(\eta(u-u_0)\right) - (u-u_0)\nabla\eta\\ &= \nabla\left(\eta(u-u_0)\right)\\ &= \sum c_{ij}^{\nabla}\left(\eta_j(u_j-u_i)\right)\end{aligned}$$

This option might improve tha numerical solution if the material property has jumps.

This has impact, most of all, on the term $\frac{1}{\rho}\nabla p$ occuring in the equation nof momentum, see EquationsToSolve .

3.2.53. COMP_FillEdges

fill additional points to the edges of an inflow area

If your setting contains an inflow that is not connected to the rest of the geometry, the inflow precision can be improved by adding additional discretisation points to the inflow boundary. This is switched on by

```
COMP_FillEdges = 1
```

Default: COMP_FillEdges = 0
In tut3d_10 COMP_FillEdges = 1 is used to improve the modeling of the inflow.

3.2.54. COMP_GradtEtaGrad_Version

define the way of numerically modeling the diffusion operator

defines the way how to numerically model the diffusion term $\nabla^T \cdot (\eta\nabla)$
with η being any physical property such as viscosity, heat conductivity, etc.
This will be important if the physical property has jumps or steep gradients.

```
COMP_GradtEtaGrad_Version = %GradtEtaGrad_DirectApproximation%
```

Default: COMP_GradtEtaGrad_Version = %GradtEtaGrad_Identity%
Possible options:

1.) %GradtEtaGrad_DirectApproximation% :: estbalish the numerical operator by direct least-squares approximation under stability optimization, i.e. utmost diagonal dominance (takes additional computation time)

2.) %GradtEtaGrad_Identity% :: using the identity $\nabla^T \cdot (\eta\nabla) = \nabla\eta^T \cdot \nabla + \eta\Delta$ and employ the already existing

operators for Gradient and Laplacian
3.) %GradtEtaGrad_None% :: #only for testing, as it is mathematically wrong: establish simply set $\nabla^T \cdot (\eta \nabla) = \eta \Delta$ and use the already computed Laplacian operator

3.2.56. COMP_IsolatedParticles_MinNbOfInteriorNeigh

minimum number of interior neighbors a points should have

Any MESHFREE point is allowed to have not less than this given number of interior neighbors.
Otherwise, the point will be deleted.

```
COMP_IsolatedParticles_MinNbOfInteriorNeigh = 0 # this will allow thin layer structure that do not contain any interior point
```

Default: COMP_IsolatedParticles_MinNbOfInteriorNeigh = 1

See also COMP_IsolatedParticles_MinNbOfNeigh .

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

Example:
COMP_IsolatedParticles_MinNbOfInteriorNeigh (iChamber) = 8

3.2.57. COMP_IsolatedParticles_MinNbOfNeigh

(chamberwise) parameter for the minimum number of (total) neighbors a points should have

Any MESHFREE point is allowed to have not less than this given number of neighbors (no matter if interior or boundary).
Otherwise, the point will be deleted.

```
COMP_IsolatedParticles_MinNbOfNeigh = 0 # will provoke that the MESHFREE point might be isolated
```

Default: COMP_IsolatedParticles_MinNbOfNeigh = 6
See also COMP_IsolatedParticles_MinNbOfInteriorNeigh .

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

Example:
COMP_IsolatedParticles_MinNbOfNeigh (iChamber) = 8

3.2.58. COMP_ManifoldContacts

(Experimental) Determines whether or not contact should be checked (manifold phase only)

For example,

```
COMP_ManifoldContacts = 1
```

Default: 0

Options available:
0 : Do not check for contacts or penetration
1 : Check only for contacts/penetration with other chambers
2 : Check only for contacts/penetration with other parts of the same chamber
NOTE: Both of the above together is not yet possible

3.2.64. COMP_RemeshBoundary

parameter to control remeshing of IGES-files (CV)

See COMP_RemeshBoundary . Definitions in USER_common_variables are dominant.

3.2.67. COMP_SharedMemoryForBE

turn on use of MPI shared memory for boundary geometry if available

Turns on the use of MPI shared memory for the boundary geometry if available. This will reduce the memory footprint as the geometry will only be stored once per physical compute node.

```
COMP_SharedMemoryForBE = true
```

Default: COMP_SharedMemoryForBE = false

Note: Not all executables support this feature. There will be a warning if it is not available.

3.2.68. COMP_SharedMemoryForGT2

turn on use of MPI shared memory for GEOTREE2 if available

Turns on the use of MPI shared memory for GEOTREE2 if available. This will reduce the memory footprint for COMP_SortBEintoBoxes_Version=21. MESHFREE might crash when use with other versions.

```
COMP_SharedMemoryForGT2 = true
```

Default: COMP_SharedMemoryForGT2 = false

Note: Not all executables support this feature. There will be a warning if it is not available.

3.2.69. COMP_SkipHighVelocities

for how many consecutive cycles a corrupt solution of velocity is accepted, before MESHFREE stops

```
COMP_SkipHighVelocities = NumberOfTimeCycles
```

Default: COMP_SkipHighVelocities = 10

If the solution to the linear system of the velocity fails (no convergence of iterative solver or
production of unphysical velocity magnitudes), then MESHFREE ignores this solution and goes on to the next time step.
The hope is, that in the next time step, the problems will be gone due to the movement/change of the point cloud.
HOWEVER, if the velocity-solution fails for "COMP_SkipHighVelocities" consecutive times, MESHFREE will stop execution

and provide an error message, accordingly.

3.2.70. COMP_SortBEintoBoxes_Version

version how to organize/prepare boundary elements for efficient computation

Version of the point tree algorithm, that is used to efficiently search for neighbors of a given MESHFREE point.
The default is

```
COMP_SortBEintoBoxes_Version = 2
```

```
  COMP_SortBEintoBoxes_Version  = 1    :: original, box-based search algorithm. The boundary triangles/elements (BE) are sorte
d into a regular box grid.
                                 If the triangles in a local region around a given point are requested, those triangles are chosen
                                 which intersect with the box the point is placed in.
  COMP_SortBEintoBoxes_Version  = 2    :: bintree-based search algorithm. The ordered hierarchically by cutting the set of BE b
y a plane into two equal
                                 half blocks. The equal half blocks are again cut into equal half blocks. In this manner, an adaptive box c
onfiguration
                                 evolved. If the triangles in a local region around a given point are requested, those triangles are chose
                                 which intersect with the adaptive box the point is placed in.
  COMP_SortBEintoBoxes_Version  = 21   :: same as COMP_SortBEintoBoxes_Version=2. The bintree is not re-established in e
very time cycle. Modalities of search
                                 treat organization are then given by  BND_SearchTreeAdministration_NbTimeStepsUntilFirstSkip  and
 BND_SearchTreeAdministration_RefreshTreeAfterHowManyCycles  .
```

3.2.72. COMP_TimeCheck

switch on time measurements for the main tasks of MESHFREE

Switch to measure the performance for different tasks of MESHFREE (see TIMECHECK).

```
COMP_TimeCheck = 1
```

Default: COMP_TimeCheck = 0 (no measurement)

COMP_TimeCheck = 1 :: run the time measurements (using the internal clock-function) and print the result in
the program's standard output. Possibly filter the output in order to see it.
COMP_TimeCheck = 2 :: run the time measurements, BUT do not print the result anywhere. Instead, the results of the
time measurement can be retrieved by the command time_check in the framework of ComputationalSteering.
The results of the measurement is appended to the .signallog-file.
COMP_TimeCheck = 3 :: run the time measurements, BUT do not print the result in the MESHFREE standard output.
Instead, the results of the time measurement are written to the file TIMECHECK.dat in the SAVE_path.

By putting a MINUS (-) in front of the number, the output is produced in hierarchy-structures, separated with commas,
such that the TIMECHECK-writeout can be copied directly into MS_Excel or LibreOffice.

TIMECHECK_Level defines the hierarchy level up to which the time measurements are performed.

MESHFREE · InputFiles · common_variables · COMP_TypeSmooth_Eta

3.2.73. COMP_TypeSmooth_Eta

type for smoothing of viscosity (CV)

See COMP_TypeSmooth_Eta . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · COMP_TypeSmooth_Rho

3.2.74. COMP_TypeSmooth_Rho

type for smoothing of density (CV)

See COMP_TypeSmooth_Rho . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · COMP_WettingAngleVariante

3.2.75. COMP_WettingAngleVariante

How to incorporate the contact angle between free surface and wall

```
COMP_WettingAngleVariante = 1
```

Default: COMP_WettingAngleVariante = 2

- COMP_WettingAngleVariante==1 ::
  1.) apprximate the contact angle by least-squares using the close free surface neighbors,
  2.) the difference between approximated and requested contact angle will be incorporated as additional curvature.

- COMP_WettingAngleVariante==2 ::
  1.) local Delaunay triangulation of the free surface,
  2.) at the edges of the triangle we apply the force of the surface tension,
  3.) if triangle edge contacts wall, the surface tension acts in the direction of the contact angle,
  4.) otherwise, it acts in the direction of the free surface.

MESHFREE · InputFiles · common_variables · COMP_dt_indep

3.2.77. COMP_dt_indep

parameter to switch on independent time stepping for two-phase LIQUID simulations with v-- and vp- (CV)

See COMP_dt_indep . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · COMP_facSmooth_Eta

3.2.79. COMP_facSmooth_Eta

parameter for weight kernel definition for smoothing of viscosity (CV)

See COMP_facSmooth_Eta . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · COMP_nbSmooth_Eta

3.2.81. COMP_nbSmooth_Eta

number of smoothing cycles for effective and total viscosity (CV)

See COMP_nbSmooth_Eta . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · COMP_nbSmooth_pCorr

3.2.82. COMP_nbSmooth_pCorr

smooth heat conductivity

This function is currently experimental, the variable COMP_nbSmooth_pCorr was previously used for smoothing the dynamic pressure, however now "mis"used for smoothing the dynamic pressure %ind_p_dyn% .

```
COMP_nbSmooth_pCorr = 2
```

Default: COMP_nbSmooth_pCorr = 0

Define the number of loops to smooth the heat conductivity, see lambda .
The k-th smoothing loop produces

$$\lambda_i^k = \sum_{j=1}^{N_i} \exp\left(-6 \cdot r_{ij}^2\right) \cdot \lambda_j^{k-1}$$

MESHFREE · InputFiles · common_variables · CONTROL_DirectTesting

3.2.83. CONTROL_DirectTesting

Instead of launching the computation, MESHFREE goes into a separate testing branch for different tasks

The currently implemented options:

CONTROL_DirectTesting = 1: PerformanceDistancePointToTriangle
CONTROL_DirectTesting = 2: PerformanceIntersectionTriangleBox
CONTROL_DirectTesting = 3: PerformanceVoronoiTesselation
CONTROL_DirectTesting = 4: quicksort
CONTROL_DirectTesting = 5: Shared Memory on Cray
CONTROL_DirectTesting = 6: Hybrid on Cray

MESHFREE steps into a different branch and executes only the implemented testing routines.
Thus, the key routines of MESHFREE can be checked within the currently compiled MESHFREE -version,
i.e. the functionality of theses modules can be verified within the framework of
dedicated MESHFREE -deliverables.

```
CONTROL_DirectTesting = 1
```

Default: CONTROL_DirectTesting = 0 (switched off)

Convenience extension:

CONTROL_DirectTesting = (numberOfTest, CONTROL_DirectTesting_Param1 , CONTROL_DirectTesting_Param2 , CONTROL_DirectTesting_Param3 , ..., additional parameters if needed, ...)

List of members:

DIFFOPconstants	performance for the construction of the local differential operators (gradient/laplace)
PerformanceDistancePointToTriangle	performance for distance point-to-triangle computation
PerformanceIntersectionTriangleBox	performance for intersection check between triangle and rectilinear box
PerformanceVoronoiTesselation	performance for the voronoi tesselation
SharedMemory	memory test for shared pointers
quicksort	check functionality of the quicksort routine for integer lists

DIFFOPconstants

performance for the construction of the local differential operators (gradient/laplace)

Here we check the performance and correctness of the operator stencil of Laplace/Neumann operators.
Create a number of random stencils and perform the constrution of the Laplace/Neumann stencil operators.

Run a performance test (flops per call, flops per second) for M points. For each point, create N random neighbors.

deprecated:

```
CONTROL_DirectTesting_Param1 = M # number of points, deafault=1000000
CONTROL_DirectTesting_Param2 = N # number of neighbors, default=40
```

```
CONTROL_DirectTesting (1) = 7 # index for this testing procedure
CONTROL_DirectTesting (2) = M # number of points, default=1000000
CONTROL_DirectTesting (3) = N # number of (average) neighbors, default = 40
CONTROL_DirectTesting (4) = Q # possible variation of N, default=0
CONTROL_DirectTesting (5) = 0 or 1 # 0: interior stencil, 1: boundary stencil, default=0
CONTROL_DirectTesting (6) = GL # gradient or laplace: either 0 (gradient) or 1 (Laplacian), default=0
CONTROL_DirectTesting (7) = 0 or 1 # 0: do not write stencils 1: write stencils to file
# additional parameters that have the same meaning as usual
DIFFOP_Version = ...
DIFFOP_kernel_Gradient = ...
DIFFOP_kernel_Laplace = ...
```

If the parameters are not defined, the default values are
M = 100000
N = 40

PerformanceDistancePointToTriangle

performance for distance point-to-triangle computation

Here we check the performance of the originally implemented algorithm, i.e. the one used for COMP_SortBEintoBoxes_Version == 21 or COMP_SortBEintoBoxes_Version == 2.

Run a performance test (flops per call, flops per second) for M points, for each point check the diatance to N triangles.
The points and triangles are established using random number generator.

M = CONTROL_DirectTesting_Param1 (number of points)
N = CONTROL_DirectTesting_Param2 (number of triangles)

If the parameters are not defined, the default values are
M = 100000
N = 1000

DOWNLOAD COMPREHENSIVE EXAMPLE

PerformanceIntersectionTriangleBox

performance for intersection check between triangle and rectilinear box

Here we check the performance of the intersection check between a rectilinear box and a triangle.

Run a performance test (flops per call, flops per second) for M intersection checks.
A box is given by the upper right and the lower left corner point.
A triangle is given by its corner points A, B, and C.

- the components of the upper right corner of the box are random numbers in (0,1)
- the components of the lower left corner of the box are random numbers in (-1,0)
- the components of A, B, C are random numbers in (-1,1)

Box and triangle coordinates are re-established after N intersection checks.

M = CONTROL_DirectTesting_Param1 (number of intersection checks to be performed)
N = CONTROL_DirectTesting_Param2 (reestalish box and triangle coordinates after N intersection checks)

If the parameters are not defined, the default values are
M = 10000000
N = 10000

PerformanceVoronoiTesselation

performance for the voronoi tesselation

Here we check the performance of the Voronoi tesselation in order to find out whether it can be used for tasks like activation of boundary points,
hole serach, check free surface point, volume computation of points etc.

Run a performance test (flops per call, flops per second) for M points. For each point, create N random neighbors.

M = CONTROL_DirectTesting_Param1 (number of points)
N = CONTROL_DirectTesting_Param2 (number of triangles)

If the parameters are not defined, the default values are
M = 100000

N = 1000

MESHFREE · InputFiles · common_variables · CONTROL_DirectTesting · SharedMemory

SharedMemory

memory test for shared pointers

Here we check the time of MPI-communication.

M = CONTROL_DirectTesting_Param1 (size of array for MPI-communication)
N = CONTROL_DirectTesting_Param2 (size of array for OpenMP loop)

If the parameters are not defined, the default values are
M = 1e+9
N = 1e+9

Here we check the memory usage for COMP_SharedMemoryForBE = true and COMP_SharedMemoryForGT2 = true.

M = CONTROL_DirectTesting_Param1 (size of array)
N = CONTROL_DirectTesting_Param2 (number of iterations)

If the parameters are not defined, the default values are
M = 100000
N = 100

MESHFREE · InputFiles · common_variables · CONTROL_DirectTesting · quicksort

quicksort

check functionality of the quicksort routine for integer lists

Check the correctness of integer list sorting by the MESHFREE -original quicksort algorithm.

- establish a random list of integers
- sort the list by quicksort
- check if sorted list is in ascending order
- error message if lists were not properly sorted
- repeat this test 1000000 times with random neighbor lists (random stencils)

N = CONTROL_DirectTesting_Param1 (length of list)
M = CONTROL_DirectTesting_Param2 (maximum size of list entries)
Q = CONTROL_DirectTesting_Param3 (refresh random stencil every Q test cycles)

If the parameters are not defined, the default values are
M = 1000
N = 1000
Q = 1

The test is repeated for

- quicksort_list_int_int
- quicksort_list_re_int
- FPMSTENCIL_OrderNeighborList (very quick but incomplete sorting of neighbors by distance)
- FPMSTENCIL_SelectClosestNeighbors (choose the 40 closest neighbors in the list)

MESHFREE · InputFiles · common_variables · CONTROL_DirectTesting_Param1

3.2.84. CONTROL_DirectTesting_Param1

additional parameter for the testing environment

see CONTROL_DirectTesting .

MESHFREE · InputFiles · common_variables · CONTROL_DirectTesting_Param2

3.2.85. CONTROL_DirectTesting_Param2

additional parameter for the testing environment

see CONTROL_DirectTesting .

MESHFREE · InputFiles · common_variables · CONTROL_DirectTesting_Param3

3.2.86. CONTROL_DirectTesting_Param3

additional parameter for the testing environment

see CONTROL_DirectTesting .

MESHFREE · InputFiles · common_variables · CONTROL_StopAfterReadingGeometry

3.2.87. CONTROL_StopAfterReadingGeometry

stops the MESHFREE program after geometry is read (CV)

See CONTROL_StopAfterReadingGeometry . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · CONTROL_writeUcvLines

3.2.88. CONTROL_writeUcvLines

write out the Ucv-lines read during startup (debugging feature)

```
CONTROL_writeUcvLines = 1
```

Default: CONTROL_writeUcvLines = 0 (no writeout)

CONTROL_writeUcvLines = 1 # write all lines read
CONTROL_writeUcvLines = 2 # write only those lines which are active (there might be lines dropped due to Selection -environment
CONTROL_writeUcvLines = 3 # write both all lines and selected lines
CONTROL_writeUcvLines = 4 # write both all lines and selected lines and pause after each CommonVar (WILL NOT WORK IN MPI-MODUS!!!!!!)

MESHFREE · InputFiles · common_variables · CompDistToBoundary_Acc

3.2.90. CompDistToBoundary_Acc

threshold of distance until which the distance to different BE is treated as equal

For a given point x, the distance of the point to two different boundary elements (BE1, BE2) is treated as EQUAL

if abs(dist(x,BE1)-dist(x,BE2)) < CompDistToBoundary_Acc *h(x)
where h(x) is the smoothing length at the location x.

3.2.92. DEBUG_Check_CCOR

generate control writeout for correction pressure

DEBUG_Check_CCOR=1

In the project directory, a file is created containing the maximum/minimum values of the dynamic pressure at several instants during one MESHFREE -time cycle.

3.2.93. DEBUG_Check_PDYN

generate control writeout for dynamic pressure

DEBUG_Check_PDYN=1
In the project directory, a file is created containing the maximum/minimum values of the dynamic pressure at several instants during one MESHFREE -time cycle.

3.2.94. DEBUG_Check_PHYD

generate control writeout for hydrostatic pressure

DEBUG_Check_PHYD=1

In the project directory, a file is created containing the maximum/minimum values of the hydrostatic pressure at several instants during one MESHFREE -time cycle.

3.2.95. DEBUG_Check_VELO

generate control writeout for velocity

DEBUG_Check_VELO=1

In the project directory, a file is created containing the maximum/minimum values of the magnitude of the velocity at several instants during one MESHFREE -time cycle.

3.2.97. DEBUG_GeneralParameter

General list of debug parameters at the developpers disposal

for development only

DEBUG_GeneralParameter = (1.0, 2.0, 3.0, ...)

Default: DEBUG_GeneralParameter = 0

Currently involved:

- DEBUG_GeneralParameter (1)...DEBUG_GeneralParameter(4) :: testing for DIFFOP_Version=10
- DEBUG_GeneralParameter (5) :: testing COMP_GradtEtaGrad_Version=%GradtEtaGrad_Identity%
- DEBUG_GeneralParameter (7) = -1 :: switch off special velocity correction at free surfaces
- DEBUG_GeneralParameter (8) = -1 :: switch off special re-interpolation of newly created free surface points
- DEBUG_GeneralParameter (9) = 1.5 (default=1000) :: DIFFOP_Version=9: for point "i", step back from order=3 to order=2 if $\max_{j=1...N(i)} \left(c_j^x(x_j - x_i) + c_j^y(y_j - y_i) + c_j^z(z_j - z_i)\right) > 1.5$
- DEBUG_GeneralParameter (10) = 0.5 (default=1000) :: DIFFOP_Version=9: for point "i", step back from order=2 to order=1 if $\max_{j=1...N(i)} \left(c_j^x(x_j - x_i) + c_j^y(y_j - y_i) + c_j^z(z_j - z_i)\right) > 0.5$
- DEBUG_GeneralParameter (11) = 0.1 (deault=0.0) :: Smagorinsky-Lilly-ansatz for viscosity (SLA) in degenerated phases/films: $\eta_{film} = \max\left(\eta^*, \rho(C_{SLA}D)^2 \frac{\|\mathbf{v}\|}{D}\right)$, with $D =$ film thickness, if $C_{SLA} < 0$ then $\eta^* = \eta_{turblent}$, else $\eta^* = \eta_{laminar}$
- DEBUG_GeneralParameter (12) = 1.5 (default=1000) :: DIFFOP_Version=9: safety threshold for interior points (TODO Tobias)
- DEBUG_GeneralParameter (13) = 0.05 (default=0.1) :: DIFFOP_Version=9: numerical differentiation step size D (relative to h): c_x = (c_0(x+D)-c_0(x-D))/(2*D), step size D=0.1 (default), can be adapted by this parameter
- DEBUG_GeneralParameter (14) = 8 (default=3) :: DIFFOP_Version=9: interior points: drop from order 3 to order 2, if norm(cx,cy,cz) > DEBUG_GeneralParameter (14), default: 3.0
- DEBUG_GeneralParameter (15) = 25 (default=16) :: DIFFOP_Version=9: boundary points: drop from order 3 to order 2, if norm(cx,cy,cz) > DEBUG_GeneralParameter (15), default: 16.0
- DEBUG_GeneralParameter (16) = 0.1 (default=1.0) :: free surface boundary conditions on velocity: weight for the div(v)=rhs condition, default: 1.0
- DEBUG_GeneralParameter (17) = 0.0 (default=1.0) :: free surface boundary conditions on velocity: weight for the div(v)=alpha*rhs condition, where DEBUG_GeneralParameter (17) describes the value of alpha, default: 1.0

3.2.98. DEBUG_SHM_MPIwindow

GASDYN parameter for FPM2

DEBUG_SHM_MPIwindow = 2

Default: DEBUG_SHM_MPIwindow = 1

Debugging shred memory, especially the creation of an MPI-window.
DEBUG_SHM_MPIwindow == 1 :: classically create the MPI-window such that the main shared process created all memory, and the slave processes create 0 memory.
DEBUG_SHM_MPIwindow == 2 :: each shared process creates an equal partition of memory inside of the window.

3.2.100. DIFFOP_ConsistentGradient

*consistent gradient in the sense d/dn = n*grad (CV)*

See DIFFOP_ConsistentGradient . Definitions in USER_common_variables are dominant.

3.2.101. DIFFOP_Neumann_ExcludeBND

(chamberwise) parameter to exclude boundary points from the neighborhood for the computation of the Neumann operators (CV)

See DIFFOP_Neumann_ExcludeBND . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · DIFFOP_Version

3.2.105. DIFFOP_Version

version of least squares operators

MESHFREE provides different versions for the least squares operators. That is due to the fact, that experience in MESHFREE -applications steadily improves also the mathematical and numerical algorithms.

```
DIFFOP_Version = 9
```

Default: DIFFOP_Version = 5

The differential operators are completely described in DOCUMATH_DifferentialOperators.pdf .

option	description
3	see section 2 -> original operator idea
5	see section 3 -> most commonly used version 5
6 and 7	sections 4 and 5 -> version 6 and 7 used for airbags (PAMCRASH FPM2) in order to handle difficult geometrical settings (folded membranes etc.)
9	section 6 -> version 9 as an attempt to come up with conservative gradient operators; numerical differentiation step size D (relative to h): c_x = (c_0(x+D)-c_0(x-D))/(2*D); step size D=0.1, can be adapted by DEBUG_GeneralParameter (16)
90 to 99	same as version 9, additional randomization of the weights in the least squares formulation between 10% and 100%
-9	**experimental** -> version 9 for interior points and version 5 for boundary points

MESHFREE · InputFiles · common_variables · DIFFOP_WeightReductionInCaseOfDeactivation

3.2.106. DIFFOP_WeightReductionInCaseOfDeactivation

(chamberwise) parameter to reduce the weight of a neighbor point in case of deactivation (CV)

See DIFFOP_WeightReductionInCaseOfDeactivation . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · DIFFOP_gradient

3.2.107. DIFFOP_gradient

type of least squares approximation stencils for gradients

Default: DIFFOP_gradient = DIFFOP_gradient_MLS

MESHFREE · InputFiles · common_variables · DIFFOP_kernel_Gradient

3.2.108. DIFFOP_kernel_Gradient

(chamberwise) factor for the weight kernel for the least squares approximation stencils for gradients (CV)

See DIFFOP_kernel_Gradient . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · DIFFOP_kernel_Laplace

3.2.109. DIFFOP_kernel_Laplace

(chamberwise) factor for the weight kernel for the least squares approximation stencils for the Laplacian (CV)

See DIFFOP_kernel_Laplace . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · DIFFOP_kernel_Neumann

3.2.110. DIFFOP_kernel_Neumann

(chamberwise) factor for the weight kernel for the least squares approximation stencils for Neumann operators (CV)

See DIFFOP_kernel_Neumann . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · DIFFOP_kernel_Transport

3.2.111. DIFFOP_kernel_Transport

(chamberwise) factor for the weight kernel for the least squares approximation stencils for the transport operators (CV)

See DIFFOP_kernel_Transport . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · DIFFOP_laplace

3.2.112. DIFFOP_laplace

type of least squares approximation stencils for the Laplacian (CV)

See DIFFOP_laplace . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · DP_UseOnlyRepulsiveContactForce

3.2.113. DP_UseOnlyRepulsiveContactForce

switch regarding attractive forces in spring-damper model (CV)

See DP_UseOnlyRepulsiveContactForce . Definitions in USER_common_variables are dominant.

3.2.118. FLIQUID_AssignPenalties_EpsilonP

vp- coupled linear system: lower bound for ratio between pressure and velocity entries, PRESSURE EQUATION

For example,

```
FLIQUID_AssignPenalties_EpsilonP = 0.3
```

In the coupled linear system "vp-", for the pressure equation, the ratio of the matrix entries referring to pressure and velocity
can be limited, such that the off-diagonal submatrix does not become too dominant. I.e.

$$\frac{IIp}{IIv} > \epsilon_P$$

'IIp' is the order of magnitude of the pressure relevant parts
'IIv' is the order of magnitude of the velocity relevant parts
in the PRESSURE equation.
More details can be found in the document DOCUMATH_ScalingOfLinearSystem_MxV.pdf .
section 4.1. "Conditions on matrix normalization",

3.2.119. FLIQUID_AssignPenalties_EpsilonV

vp- coupled linear system: upper bound for ratio between velocity and pressure entries, VELOCITY EQUATION

For example,

```
FLIQUID_AssignPenalties_EpsilonV = 0.3
```

In the coupled linear system "vp-", for the velocity equation(s), the ratio of the matrix entries referring to velocity and pressure
can be limited, such that the off-diagonal submatrix does not become too dominant. I.e.

$$\frac{Iv}{Ip} > \epsilon_V$$

'Iv' is the order of magnitude of the velocity relevant parts
'Ip' is the order of magnitude of the pressure relevant parts
in the VELOCUTY equation.
More details can be found in the document DOCUMATH_ScalingOfLinearSystem_MxV.pdf ,
section 4.1. "Conditions on matrix normalization",

3.2.121. FLIQUID_ConsistentPressure_CoeffMM

TEMPORARY: factor to study consistent pressure version 2

```
FLIQUID_ConsistentPressure_CoeffMM = 0.01
```

Default: FLIQUID_ConsistentPressure_CoeffMM = 0.0
In the RegularizeDPA -algorithm, it provides a possibility to derfine the vector $\boldsymbol{q}_i$

3.2.122. FLIQUID_ConsistentPressure_CoeffNN

TEMPORARY: factor to study consistent pressure version 2

FLIQUID_ConsistentPressure_CoeffNN = 0.01

Default: FLIQUID_ConsistentPressure_CoeffNN = 0.0
In the RegularizeDPA -algorithm, it provides a possibility to derfine the vector $\boldsymbol{q}_i$

MESHFREE · InputFiles · common_variables · FLIQUID_ConsistentPressure_CoeffTT

3.2.123. FLIQUID_ConsistentPressure_CoeffTT

TEMPORARY: factor to study consistent pressure version 2

FLIQUID_ConsistentPressure_CoeffTT = 0.01

Default: FLIQUID_ConsistentPressure_CoeffTT = 0.0
In the RegularizeDPA -algorithm, it provides a possibility to derfine the vector $\boldsymbol{q}_i$

MESHFREE · InputFiles · common_variables · FLIQUID_ConsistentPressure_CoeffWEIGHT

3.2.124. FLIQUID_ConsistentPressure_CoeffWEIGHT

TEMPORARY: factor to study consistent pressure version 2

FLIQUID_ConsistentPressure_CoeffWEIGHT = 0.9

Default: FLIQUID_ConsistentPressure_CoeffWEIGHT = 1.0
In the RegularizeDPA -algorithm, it provides a possibility to less/more emphasize the weight-approach W_{ij} .
currently experimental, better do not touch

MESHFREE · InputFiles · common_variables · FLIQUID_ConsistentPressure_UseDivV

3.2.125. FLIQUID_ConsistentPressure_UseDivV

(chamberwise) parameter to use numerical approximations of div(v) in direct computation of dynamic pressure (i.e. consistent pressure)

FLIQUID_ConsistentPressure_UseDivV = 0

Default: FLIQUID_ConsistentPressure_UseDivV = 1

Use or DO NOT use the term containing the divergence of velocity in the Poisson equation for the dynamic pressure, see equations (3.43) and (3.44) in DOCUMATH_NumericalSchemeIncompressible.pdf .

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

MESHFREE · InputFiles · common_variables · FLIQUID_ConsistentPressure_Version

3.2.126. FLIQUID_ConsistentPressure_Version

version how to compute the consistent pressure (CV)

See FLIQUID_ConsistentPressure_Version . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · FOFTLIQUID_AdditionalCorrectionLoops

3.2.127. FOFTLIQUID_AdditionalCorrectionLoops

additional velocity correction loops (CV)

See FOFTLIQUID_AdditionalCorrectionLoops . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · FPM_LICENSE_FILE

3.2.128. FPM_LICENSE_FILE

overwrite the environment variable

```
FPM_LICENSE_FILE = 'FPM.lcs'
```

default: FPM_LICENSE_FILE = 'none'

BE AWARE that this overrides the definition of the environment variable FPM_LICENSE_FILE, see EnvironmentVariables .

MESHFREE · InputFiles · common_variables · GASDYN_CorrectEnergy

3.2.129. GASDYN_CorrectEnergy

correct total energy in GASDYN application

```
GASDYN_CorrectEnergy = 0.01 # allow 1% of the defect energy to be corrected during a time step
```

Default: GASDYN_CorrectEnergy = 0.0

The correction preocedure is

$$_{i}^{corrected} =_{i} +\gamma W_{i}$$

determine γ such that

$$\int_{\Omega} \gamma W \, dV \approx \sum_{\Omega} \gamma W_i V_i = \mathrm{GasdynCorrectTotalEnergy} \cdot \mathrm{DefectTotalEnergy}$$

MESHFREE · InputFiles · common_variables · GASDYN_CorrectMass

3.2.130. GASDYN_CorrectMass

correct mass in GASDYN application

```
GASDYN_CorrectMass = 0.01 # allow 1% of the defect mass to be corrected during a time step
```

Default: GASDYN_CorrectMass = 0.0

The correction preocedure is

$$\rho_i^{corrected} = \rho_i + \gamma W_i^\rho$$

determine γ such that

$$\int_\Omega \gamma W^\rho dV \approx \sum_\Omega \gamma W_i^\rho V_i = \mathrm{GasdynCorrectMass} \cdot \mathrm{DefectMass}$$

3.2.131. GASDYN_FPM2_alpha

GASDYN parameter for FPM2

```
GASDYN_FPM2_alpha = 10
```

Default: GASDYN_FPM2_alpha = 13

This is the parameter α in the FPM2-description DOCUMATH_Gasdyn_O2.pdf , equation (5.4) ff.

3.2.132. GASDYN_FPM2_beta

GASDYN parameter for FPM2

```
GASDYN_FPM2_beta = 1.0
```

Default: GASDYN_FPM2_beta = 0.5

This is the parameter β in the FPM2-description DOCUMATH_Gasdyn_O2.pdf , equation (5.4) ff.

3.2.133. GASDYN_T_gain

limit the temperature gain in GASDYN-applications

```
GASDYN_T_gain = 0.1 # allow temperature to grow by not more than 10% per time step
```

Default: GASDYN_T_gain = 0.2

3.2.134. GASDYN_T_loss

limit the temperature drop in GASDYN-applications

```
GASDYN_T_loss = 0.1 # allow temperature to drop by not more than 10% per time step
```

Default: GASDYN_T_loss = 0.2

3.2.135. GASDYN_Upwind2ndOrder

DEPRECATED!!! (GASDYN parameter for FPM1)

This is a deprecated parameter. Do not use anymore. Instead, use the parameters GASDYN_Upwind_Lbeta and GASDYN_Upwind_Lgamma .

GASDYN_Upwind2ndOrder = 0.0

Default: GASDYN_Upwind2ndOrder = 0.5

Represents gamma in the improved (practically second order} upwind velocity $\mathbf{v}_{uw}^{*} = \mathbf{v}_{uw} - \gamma \frac{\Delta t}{\rho} \nabla p$
Second order is reached, if GASDYN_Upwind2ndOrder = 0.5 .

Please remember that the classical upwind velocity is given by $\mathbf{v}_{uw} = \mathbf{v} - \frac{1}{2\rho c}\left(p^{+} - p^{-}\right)$
The distance between the upwind locations at the plus(+) and minus(-)-points is ruled by the parameter GASDYN_UpwindOffset .

This second order idea comes from the following consideration: First order (for example for the density) is given by

$$\frac{\rho^{n+1} - \rho^{n}}{\Delta t} = -\bar{\rho} \cdot \nabla^{T}\left(\mathbf{v}\right)$$

Higher order (second order) improvement is given by

$$\frac{\rho^{n+1} - \rho^{n}}{\Delta t} = -\bar{\rho} \cdot \nabla^{T}\left(\mathbf{v} + \gamma \Delta t \frac{d\mathbf{v}}{dt}\right) = -\bar{\rho} \cdot \nabla^{T}\left(\mathbf{v} - \gamma \frac{\Delta t}{\rho} \nabla p\right)$$

OPTION:
Choose this parameter negative, i.e.

GASDYN_Upwind2ndOrder = -0.2

This will lead to the improved upwind velocity

$$\mathbf{v}_{uw}^{*} = \mathbf{v} - |\gamma| \frac{h}{c} \frac{1}{\rho} \nabla p$$

This improved upwind idea comes from the consideration that

$$\mathbf{v}_{uw} = \mathbf{v} - \frac{1}{2\rho c}\left(p^{+} - p^{-}\right) \approx \mathbf{v} - \gamma \frac{h}{c} \frac{1}{\rho} \nabla p$$

So, the classical upwind velocity can be approximated in this way. The nice side effect is, that the divergence of the upwind velocity leads to laplace-lke term (damping!!!) in the numerical scheme, i.e.

$$\frac{\rho^{n+1} - \rho^{n}}{\Delta t} = -\bar{\rho} \cdot \left[\nabla^{T}\left(\mathbf{v}\right) + \nabla^{T}\left(|\gamma| \frac{h}{c} \frac{1}{\rho} \nabla p\right)\right]$$

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

3.2.136. GASDYN_UpwindOffset

(chamberwise) GASDYN parameter for FPM1

GASDYN_UpwindOffset = 0.2

Default: GASDYN_UpwindOffset = 0.15

It represents the parameter α in GeneralizedUpwind .

Additional information:
The spatial shift in order to compute the upwind quantities is GASDYN_UpwindOffset times smoothing length. i.e.

$$\mathbf{v}_{uw} = \mathbf{v} - \frac{1}{2\rho c}\left(p^{+} - p^{-}\right) = \mathbf{v} - \frac{1}{2\rho c}\left(p(\mathbf{x} + \gamma h \mathbf{n}_{uw}) - p(\mathbf{x} - \gamma h \mathbf{n}_{uw})\right)$$

where γ represents the present parameter GASDYN_UpwindOffset and $\mathbf{n}_{uw}$ is the upwind direction.

GASDYN_UpwindOffset is equal to the parameter α_{uw} in equation (13) in the FPM1-paper paper_SIA_2005_kuhnert.pdf .
Refer also to equations (6.12) and (6.13) in thesis_kuhnert.pdf . Here, GASDYN_UpwindOffset represents the value $\Delta\tau\frac{c}{h}$.

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

MESHFREE · InputFiles · common_variables · GASDYN_Upwind_Lbeta

3.2.137. GASDYN_Upwind_Lbeta

(chamberwise) GASDYN parameter for FPM1 and FPM3

GASDYN_Upwind_Lbeta = 0.5 # second order time integration

Default: GASDYN_Upwind_Lbeta = 0.0 (first order time integration)

GASDYN_Upwind_Lbeta represents the parameter β in GeneralizedUpwind .

Additional feature:
By putting a minus-sign in front, i.e. GASDYN_Upwind_Lbeta = -A, we have

$$\beta = \begin{cases} abs(A) \ if \ \nabla^T \mathbf{v} > 0 \ (\text{rarefaction}) \\ 0 \ \text{elsewise} \end{cases}$$

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

MESHFREE · InputFiles · common_variables · GASDYN_Upwind_Lgamma

3.2.138. GASDYN_Upwind_Lgamma

(chamberwise) GASDYN parameter for FPM1 and FPM3

GASDYN_Upwind_Lgamma = 0.2 # 0.2*H as upwind step size

Default: GASDYN_Upwind_Lgamma = 0.0 (no upwind)

GASDYN_Upwind_Lgamma represents the parameter γ in GeneralizedUpwind .

Additional feature:
By putting a minus-sign in front, i.e. GASDYN_Upwind_Lgamma = -A, then we have

$$\gamma = \begin{cases} 0 \; if \; \nabla^T \mathbf{v} > 0 \; (\text{rarefaction}) \\ abs(A) \; \text{elsewise} \end{cases}$$

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

3.2.139. GASDYN_Version

(chamberwise) GASDYN parameter to choose FPM1 or FPM2

```
GASDYN_Version = 'FPM1'
```

Default: GASDYN_Version = 'FPM2'

FPM1: see paper_SIA_2005_kuhnert.pdf and chapter 6 in thesis_kuhnert.pdf .
A condensed summary of the FPM upwind is given in DOCUMATH_GeneralizationOfUpwindFPM.pdf
FPM2: see DOCUMATH_Gasdyn_O2.pdf .
FPM3: same as FPM1. The upwind step size is put to zero in case of expansion (i.e. if $\nabla^T(\mathbf{v}) > 0$). This models rarefaction waves more precisely.
Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

3.2.140. GASDYN_p_gain

limit the pressure gain in GASDYN-applications

```
GASDYN_p_gain = 0.1 # allow pressure to grow by no more than 10% per time step
```

Default: GASDYN_p_gain = 0.2

3.2.141. GASDYN_p_loss

limit the pressure drop in GASDYN-applications

```
GASDYN_p_loss = 0.1 # allow pressure to drop by no more than 10% per time step
```

Default: GASDYN_p_loss = 0.2

3.2.142. GASDYN_r_gain

limit the density gain in GASDYN-applications

```
GASDYN_r_gain = 0.1 # allow density to grow by no more than 10% per time step
```

Default: GASDYN_r_gain = 0.2

MESHFREE · InputFiles · common_variables · GASDYN_r_loss

3.2.143. GASDYN_r_loss

limit the density drop in GASDYN-applications

```
GASDYN_r_loss = 0.1 # allow density to drop by no more than 10% per time step
```

Default: GASDYN_r_loss = 0.2

MESHFREE · InputFiles · common_variables · GEOTREE2_BND_FinalBoxDimension

3.2.144. GEOTREE2_BND_FinalBoxDimension

relative size extent of GEOTREE2 leaves

The size extent is given relative to the local smoothing length.

```
GEOTREE2_BND_FinalBoxDimension = 1.0
```

Default: 0.5
Only taken into account if COMP_SortBEintoBoxes_Version is set to 21.
See also GEOTREE2_BND_FinalBoxSize .

MESHFREE · InputFiles · common_variables · GEOTREE2_BND_FinalBoxSize

3.2.145. GEOTREE2_BND_FinalBoxSize

number of triangles in a GEOTREE2 leave

```
GEOTREE2_BND_FinalBoxSize = 100
```

Default: 10
Only taken into account if COMP_SortBEintoBoxes_Version is set to 21.
See also GEOTREE2_BND_FinalBoxDimension .

MESHFREE · InputFiles · common_variables · GEOTREE2_EstablishCON_Version

3.2.146. GEOTREE2_EstablishCON_Version

parameter for the bintree-search of the neighborhood of MESHFREE points

Rules the version how to execute the loop to establish all neighborhood stencils.
Version 1: loop over all leafs -> neighborhood of leaf -> each point in the leaf will obtain the neighbor list of the leaf
Version 2: loop over all leafs -> try to vectorizes all points contained by a leaf -> attention: seems to deliver not always the same as version 3
Verison 3: loop over all points -> neighborhood directly from the search tree

```
GEOTREE2_EstablishCON_Version = 2 # use the old version
```

Default: GEOTREE2_EstablishCON_Version = 3 (new version)

OPTIONAL VALUE: sort-by-distance version

```
GEOTREE2_EstablishCON_Version = (3, 4 )
```

Default: GEOTREE2_EstablishCON_Version = (3, 2)
The second parameter invokes the way how the neighbor lists are sorted by their distance to the central point. (This enables to select the closest N neigbors, N given by the parameter max_N_stencil)
sort-by-distance version 1: do not use -> automatically fallling back to default
sort-by-distance version 2: classical quicksort (default)
sort-by-distance version 3: fast sorting, which allows for permutations of points whose distance to the central point is almost the same
sort-by-distance version 4: subdivide the stencil in the N closest points (coming first in the neighbor list) and the rest (thus also allowing us to select the closest N neighbors).

List of members:

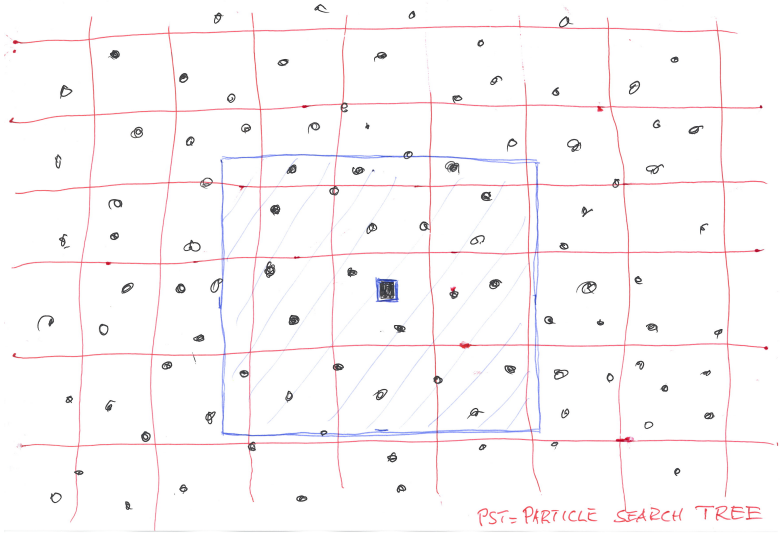
Version=3	special remarks on version 3
Version=2	special remarks on version 2

Version=2

special remarks on version 2

- compute potential neighbors around some single point (central points, black square)
- store potential neighbors in a local list (blue marked area)
- with local list, produce the final neighbor list of the point

Disadvantage: for every point, the program will perform one access to the point search tree.

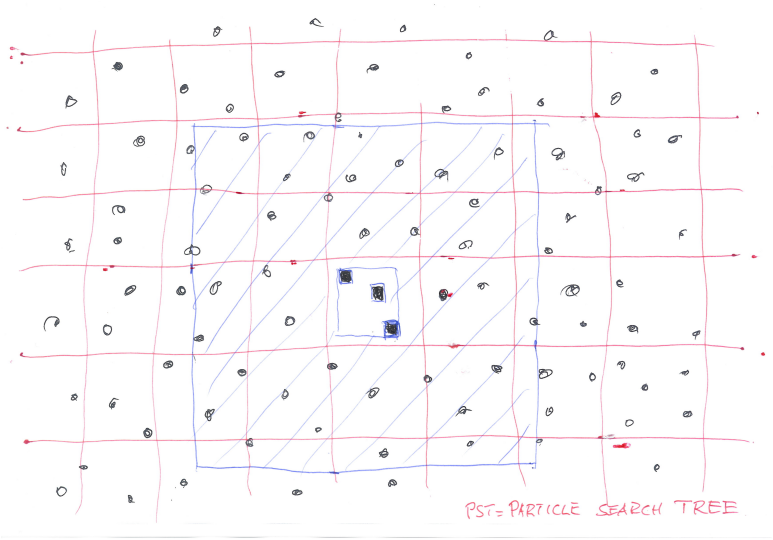


Version=3

special remarks on version 3

- compute potential neighbors around ALL points (central points, black squares) of some cell of the point search tree (blue marked area)
- store potetial neighbors in a local list (most possibly fitting into in cache memory)
- with local list, produce the final neighbor lists of all central points (most possibly all operation are out of the cache memory)

Advantage: reduce the number of search tree access operations.



3.2.147. GEOTREE2_FinalBoxSize

parameter for the bintree-search of the neighborhood of MESHFREE points

For the point search tree, assign the number of MESHFREE points that should be in the tree leaf.

```
GEOTREE2_FinalBoxSize = 8
```

Default: GEOTREE2_FinalBoxSize = 16

3.2.148. GEOTREE2_IntListMargin

parameter for the bintree-search of the neighborhood of MESHFREE points

For the point searcch tree, assign the margin for the list of point indices in the leaf, i.e. if the list is reallocated, how many empty places are in the list. That avoids reallocation at every time a new point is created.

```
GEOTREE2_IntListMargin = 10
```

Default: GEOTREE2_IntListMargin = 4

3.2.149. GEOTREE2_MaximumBoxSize

parameter for the bintree-search of the neighborhood of MESHFREE points

For the point searcch tree, assign the MAXIMUM number of MESHFREE point that should
be in the tree leaf.

GEOTREE2_MaximumBoxSize = 12

Default: GEOTREE2_MaximumBoxSize = 20

MESHFREE · InputFiles · common_variables · GEOTREE2_SizeOfSearchBox

3.2.150. GEOTREE2_SizeOfSearchBox

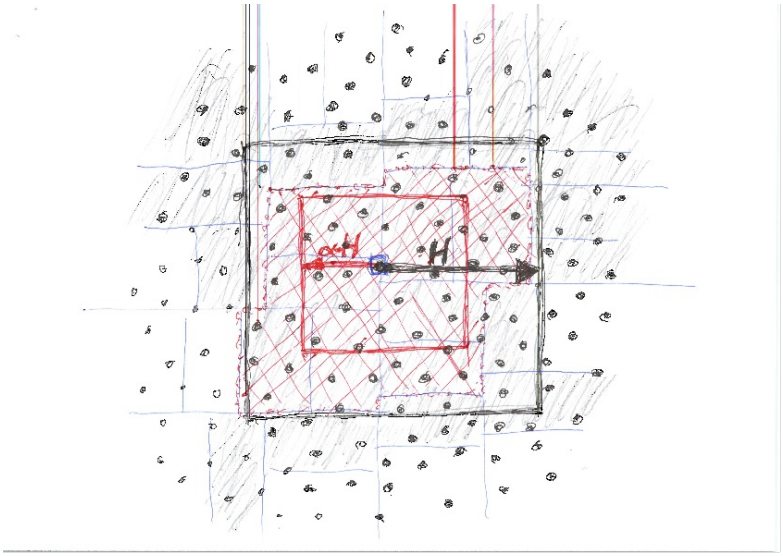
parameter for the bintree-search of the neighborhood of MESHFREE points

In order to find the CANDIDATES of neighbors to a point, MESHFREE collects all points from the search-tree cells (blue cells in the picture)
that intersect with a box of size H. The grey marked cells are the ones containing all condidates. One can imagine, that especially in 3D
this is still a big number of points.

To reduce the effort, one can reduce the size of the search box by the factor α =GEOTREE2_SizeOfSearchBox
still coming out with sufficiently many candidates, but saving some computation time.

GEOTREE2_SizeOfSearchBox = 0.5

Default: GEOTREE2_SizeOfSearchBox = 1.0



MESHFREE · InputFiles · common_variables · IGES_Accuracy

3.2.155. IGES_Accuracy

relative accuracy for consistency checks of IGES-faces (CV)

See IGES_Accuracy . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · IGES_HealCorruptFaces

3.2.156. IGES_HealCorruptFaces

allow a certain depth of healing triangulation of IGES faces by refinement (CV)

See IGES_HealCorruptFaces . Definitions in USER_common_variables are dominant.

3.2.157. INTEGRATION_ReopenTimestpFilesAfterHowManyCycles

**.timestep-Files close and reopen again after how many cycles (debug reasons)*

On the cluster, the permanently opened file units of the *.timestep-files get sometimes in conflict with the frequent reopening of these files if FPM_CurveMonitor is used. Try to check, if re-opening can avoid this trouble.

On the other hand, keeping the file units open in MESHFREE leads to better performance, especially on slow file systems (like the ITWM-one)

3.2.158. ISOLATEDPOINTS_ClusterOnResultingVolume

threshold to cluster two isolated points into one

```
ISOLATEDPOINTS_ClusterOnResultingVolume = 1.0
```

Default: ISOLATEDPOINTS_ClusterOnResultingVolume = 0.0

Isolated points do not have neighbors, they are marked by the value %ORGANIZE_IsIsolated% in the variable Y %ind_Organize% . If two isolated points (index i and j) come close to each other (distance is less than dist_rip * Y %ind_h%), then they may be clustered to one point, if the resulting volume is small enough. That is

- the resulting volume fulfills $(V_i + V_j)^{\frac{1}{3}} \leq \alpha \cdot \frac{1}{2}(h_i + h_j)$, where α represents the value of ISOLATEDPOINTS_ClusterOnResultingVolume
- their normals agree in the sense $\mathbf{n}_i \cdot \mathbf{n}_j \geq 0.5$

This feature only works for single phase liquid simulations.

3.2.159. ISOLATEDPOINTS_ProduceVolumePackage

threshold to turn isolated points into volume packages

```
ISOLATEDPOINTS_ProduceVolumePackage = 0.5
```

Default: ISOLATEDPOINTS_ProduceVolumePackage = 100000

If an isolated point

- is close enough to the boundary, i.e. if Y %ind_dtb% < ISOLATEDPOINTS_ProduceVolumePackage *Y %ind_h%
- has a representative volume that fulfills Y %ind_Vi% > ISOLATEDPOINTS_ProduceVolumePackage *Y %ind_h% *((0.3*Y %ind_h%)^2)

then it is turned into a cubicle of MESHFREE points, representing that volume.

3.2.163. ITWMESI_PressureMapping_Filter

coupling ITWMESI filter for mapping the pressure solution to the boundary elements

ITWMESI_PressureMapping_Filter = 0.0

This is the default.
The updated pressure values are computed by

$$p_{map}^{n+1} = p_{map}^{n} \cdot \mathrm{Filter} + p_{current} \cdot (1 - \mathrm{Filter})$$

3.2.164. ITWMESI_PressureMapping_WeightPdyn

coupling ITWMESI weight for mapping dynamic pressure

ITWMESI_PressureMapping_WeightPdyn = 1.0

This is the default. For testing, this value can be changed. 0 would mean: ignore the dynamic pressure.
The pressure communicated to VPS is

$$p_{current} = \mathrm{WeightPdyn} \cdot p_{dyn}^{n+1} + \mathrm{WeightPhyd} \cdot p_{hyd}^{n+1}$$

See also ITWMESI_PressureMapping_WeightPhyd .

3.2.165. ITWMESI_PressureMapping_WeightPhyd

coupling ITWMESI weight for mapping hydrostatic pressure

ITWMESI_PressureMapping_WeightPhyd = 1.0

This is the default. For testing, this value can be changed. 0 would mean: ignore the hydrostatic pressure.
The pressure communicated to VPS is

$$p_{current} = \mathrm{WeightPdyn} \cdot p_{dyn}^{n+1} + \mathrm{WeightPhyd} \cdot p_{hyd}^{n+1}$$

See also ITWMESI_PressureMapping_WeightPdyn .

3.2.166. ITWMESI_ShearForceMapping_BasedOnStresses

coupling ITWMESI: decide whether the shear forces be projected as stress values (N/m^2) or as forces (N)

ITWMESI_ShearForceMapping_BasedOnStresses = 0

This is the default and projects shear forces (unit: N) per VPS element.
In the case of ITWMESI_ShearForceMapping_BasedOnStresses = 1, average shear stresses (unit: N/m^2) are mapped to

the VPS shell elements.

MESHFREE · InputFiles · common_variables · ITWMESI_ShearForceMapping_Filter

3.2.167. ITWMESI_ShearForceMapping_Filter

coupling ITWMESI filter for mapping the shear force solution to the VPS boundary elements

ITWMESI_ShearForceMapping_Filter = 0.0

This is the default.
The updated shear forces are computed by

$$Sn_{map}^{n+1} = Sn_{map}^{n} \cdot \mathrm{Filter} + Sn_{current} \cdot (1 - \mathrm{Filter})$$

MESHFREE · InputFiles · common_variables · ITWMESI_ShearForceMapping_Weight

3.2.168. ITWMESI_ShearForceMapping_Weight

coupling ITWMESI weight for mapping the shear forces

ITWMESI_ShearForceMapping_Weight = 1.0

This is the default. For testing, this value can be changed. 0 would mean: ignore the dynamic shear forces.

MESHFREE · InputFiles · common_variables · ITWMMpCCI_PressureMapping_WeightPdyn

3.2.169. ITWMMpCCI_PressureMapping_WeightPdyn

coupling ITWMESI weight for mapping dynamic pressure

ITWMMpCCI_PressureMapping_WeightPdyn = 1.0

This is the default. For testing, this value can be changed. 0 would mean: ignore the dynamic pressure.
The pressure communicated to VPS is

$$p_{current} = \mathrm{WeightPdyn} \cdot p_{dyn}^{n+1} + \mathrm{WeightPhyd} \cdot p_{hyd}^{n+1}$$

See also ITWMMpCCI_PressureMapping_WeightPhyd .

MESHFREE · InputFiles · common_variables · ITWMMpCCI_PressureMapping_WeightPhyd

3.2.170. ITWMMpCCI_PressureMapping_WeightPhyd

coupling ITWMESI weight for mapping hydrostatic pressure

ITWMMpCCI_PressureMapping_WeightPhyd = 1.0

This is the default. For testing, this value can be changed. 0 would mean: ignore the hydrostatic pressure.
The pressure communicated to VPS is

$$p_{current} = \mathrm{WeightPdyn} \cdot p_{dyn}^{n+1} + \mathrm{WeightPhyd} \cdot p_{hyd}^{n+1}$$

See also ITWMMpCCI_PressureMapping_WeightPdyn .

3.2.171. LIMITER

slope limiter for controlling numerical diffusion in MUSCL-reconstruction scheme in EULERIMPL and EULEREXPL setting

```
LIMITER = 1
```

Default: LIMITER = 5

The following limiters are implemented and can be used in the EULERIMPL and EULEREXPL setting:

- LIMITER = 1 -> van Leer:
  $$\phi(r) = \frac{r + |r|}{1 + |r|}$$
- LIMITER = 2 -> Minmod:
  $$\phi(r) = \max\left(0, \min(r, 1)\right)$$
- LIMITER = 3 -> Superbee:
  $$\phi(r) = \max\left(0, \min(2r, 1), \min(r, 2)\right)$$
- LIMITER = 4 -> Koren:
  $$\phi(r) = \max\left(0, \min\left(2r, \frac{1}{3}(1 + 2r), 2\right)\right)$$
- LIMITER = 5 -> Sweby:
  $$\phi(r) = \max\left(0, \min(\beta\, r, 1), \min(r, \beta)\right), \qquad 1 \leq \beta \leq 2$$
  β can be controlled by BETA_FOR_LIMITER (default: $\beta = 1.9$)
- LIMITER = 6 -> monotoniced central (MC):
  $$\phi(r) = \max\left(0, \min\left(2r, \frac{1}{2}(1 + r), 2\right)\right)$$
- LIMITER = 7 -> van Albada 2:
  $$\phi(r) = \frac{2r}{1 + r^2}$$
- LIMITER = 8 -> Barth & Jespersen:
  $$\phi(r) = \frac{1}{2}(r + 1)\, \min\left(\min\left(1, \frac{4r}{r+1}\right), \min\left(1, \frac{4}{r+1}\right)\right)$$
- LIMITER = 9 -> 1st order Upwind:
  $$\phi = 0$$

3.2.172. LINEQN_scaling

choose the way how to scale/normalize the linear systems (CV)

See LINEQN_scaling . Definitions in USER_common_variables are dominant.

3.2.173. LINEQN_solver_ScalarSystems

linear solver to be used for the scalar systems like pressure, temperature, etc. (CV)

See LINEQN_solver_ScalarSystems . Definitions in USER_common_variables are dominant.

3.2.174. LINEQN_solver

linear solver to be used for the coupled vp- or v-- system (CV)

See LINEQN_solver . Definitions in USER_common_variables are dominant.

3.2.175. MASS_correction_DivergenceVelocity

Mass Correction for weakly compressible flow problems

MASS_correction_DivergenceVelocity = 'YES'

Default: MASS_correction_DivergenceVelocity = 'NON'

It only works for the LIQUID solver and for problems with pressure dependent densities!
Furthermore at the moment an inflow and/or outflow boundary condition is required to determine the target mass!
The idea is to add a source term $\tilde{q}$ to the continuity equation

$$\frac{D\varrho}{Dt} + \varrho \nabla \cdot \mathbf{v} = \tilde{q}$$

in order to compensate the mass loss resp. the gain in mass. Hence the desired divergence of velocity for the CorrectionPressureAlgorithm
(see DesiredAndNominalDivergenceOfVelocity) is computed by

$$\nabla \cdot \mathbf{v} = -\frac{1}{\varrho}\left(\frac{D\varrho}{Dt} - \tilde{q}\right)$$

This is also used for the computation of the dynamic pressure (see FLIQUID_ConsistentPressure_Version, ClassicalDPA , RegularizeDPA , AlternativeDPA).
Therefore it can be interpreted as a correction method of the dynamic pressure as well.

For the computation of the source term $\tilde{q}$ the relative error err of target mass

$$M_t = \int_{t^0}^{t^n} \int_{\partial\Omega_{\mathrm{in/out}}} \varrho(t)\big(\boldsymbol{v}(t)\cdot\boldsymbol{n}(t)\big)\,dA\,dt \approx \sum_{k=0}^{n}\sum_{i\in P} \varrho_i(t^k)\big(\boldsymbol{v}_i(t^k)\cdot\boldsymbol{n}_i(t^k)\big)A_i$$

and current mass

$$M_c = \int_{\Omega} \varrho\,dV - \int_{\Omega} \varrho^{(\mathrm{start})}\,dV \approx \sum_{i\in P} \varrho_i \cdot V_i - \sum_{i\in P} \varrho_i^{(\mathrm{start})} \cdot V_i$$

is computed by

$$\mathrm{err} = \frac{M_t - M_c}{M_t}.$$

Moreover the relative error err is weighted with a coefficient $d \in [1, 1200]$, which depends on the absolute error, the smaller the mass difference the higher d .
But overall the product $\mathrm{coeff} = d \cdot \mathrm{err}$ is limited by

$$-12 \leq \mathrm{coeff} \leq 12.$$

This results in the source term

$$\tilde{q} = \mathrm{coeff} \cdot \tilde{\varrho} = d\,\frac{M_t - M_c}{M_t}\,\tilde{\varrho},$$

where

$$\tilde{\varrho} = \frac{1}{N}\sum_{i=1}^{N} \varrho_i$$

is the average density.

MESHFREE · InputFiles · common_variables · MEMORIZE_ResetReadFlag

3.2.176. MEMORIZE_ResetReadFlag

reset frequency for MEMORIZE_Read flag (CV)

See MEMORIZE_ResetReadFlag . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · MESHFREE_LICENSE_FILE

3.2.177. MESHFREE_LICENSE_FILE

overwrite the environment variable

```
MESHFREE_LICENSE_FILE = 'MESHFREE.lcs'
```

default: MESHFREE_LICENSE_FILE = 'none'

BE AWARE that this overrides the definition of the environment variable MESHFREE_LICENSE_FILE, see EnvironmentVariables .

MESHFREE · InputFiles · common_variables · MPI_WeightingMethodForBisection

3.2.180. MPI_WeightingMethodForBisection

how to give weights to points for the MPI-bisection process

MPI-bisection performed such that the sum of the point-weights is equal among the single MPI-processes.
Default weights:

- active points: weight=1
- inactive points: weight=0
- STANDBY points: weight=0

MPI_WeightingMethodForBisection = (1, 0.07)

Default: MPI_WeightingMethodForBisection = (0, 0.02)

- **first entry:**
  MPI_WeightingMethodForBisection (1) == 0: dry/inactive points have weight 0, active points have weight 1
  MPI_WeightingMethodForBisection (1) == 1: dry/inactive points and active points get a weight according to the last per-point computation times, given by %CLOCK_STATISTICS_FLIQUID% and %CLOCK_STATISTICS_ORGANIZE%
  MPI_WeightingMethodForBisection (1) < 1: the weight applied for inactive points

- **second entry**
  MPI_WeightingMethodForBisection (2) represents the weight for the points in the STANDBY -pointcloud.
  Remark: STANDBY pointsclouds usually do not cost simulation time (only at the startup and the first 5...10 time cycles. However, if having no weight, in the worst case it can happen, that ALL STANDBY points fall into a single MPI-process. Thus, if the STANDBY pointcloud contains 100Mio points (which is not unrealistic), this would lead to the clash of the program. By giving weights to the STANDBY points, one can destress the situation.

Note: As MPI_WeightingMethodForBisection=1 incorporates the measured performance into finding the MPI bisection, this yields **non-deterministic** results and should be used for performance tuning only.

3.2.183. NB_OF_ACCEPTED_REPETITIONS

number of permitted repetitions of substep in EULERIMPL setting

NB_OF_ACCEPTED_REPETITIONS = 3

Default: NB_OF_ACCEPTED_REPETITIONS = 1

If the automatic time step size control method based on local errors (see TOL_T , TOL_v , TOL_keps) rejects a result of an EULERIMPL substep,
the substep will be recomputed with a smaller time step size

$$\Delta t_{k+1} = \Delta t_k \left(\frac{\theta \, \mathrm{TOL}}{\|\hat{\mathbf{e}}_{k+1}\|} \right)^{1/(\hat{p}+1)} .$$

NB_OF_ACCEPTED_REPETITIONS controls how often the substep may be repeated. If this number is reached, MESHFREE will continue with the current (inaccurate) result.

3.2.184. NB_POINTS_BC_HEAT_EQUATION_1D

number of points for 1D heat equation for temperature boundary condition

NB_POINTS_BC_HEAT_EQUATION_1D = 20

Default: NB_POINTS_BC_HEAT_EQUATION_1D = 10

This is the discretization measure of the 1D heat equation (see HeatEquation1D). The maximum number of points is limited to 40.

3.2.185. NEIGHBOR_FilterMethod

choose how to exclude neighbors from MESHFREE points at critical geometry parts

NEIGHBOR_FilterMethod = 1

Default: NEIGHBOR_FilterMethod = 0

Algorithms chosen:

NEIGHBOR_FilterMethod == 0 : NormalBased
PositionBased
NEIGHBOR_FilterMethod == 1 : GeometryBased
NormalBased
ReplugNeighbors
NEIGHBOR_FilterMethod == 2 : GeometryBased
ReplugNeighbors
NEIGHBOR_FilterMethod == 3 : GeometryBased
ReplugNeighbors
this version is a pure re-implementation of NEIGHBOR_FilterMethod==2 with optimal memory caching.
However, it does not seem to be significantly faster than version 2
NEIGHBOR_FilterMethod == 4 : same as version 3, ADDITIONALLY: include the free surface points, also representing a boundary disc

List of members:

Version=3+4	special option for verion 3 and 4

Version=3+4

special option for verion 3 and 4

NEIGHBOR_FilterMethod = (3, **NormalShift** , **DiskSize** , **nValidNeighbors** , **geodeticDistance**)

default: NEIGHBOR_FilterMethod = (3, **5** , **35** , **1** , **90**)

- **NormalShift** (in percent!!!): regular boundary point to be shifted by (NormalShift/100)*H towards the interior for ray analysis
- **DiskSize** (in percent!!!): disk size (representing the boundary around a regular boundary point) is (DiskSize/100)*H
- **nValidNeighbors** : a neighbor is plugged back (see ReplugNeighbors) if both have this number of common valid points
- **geodeticDistance** (in percent!!!): a neighbor is plugged back (see ReplugNeighbors) only if the geodetic distance to the central point is less than (geodeticDistance/100)*H

3.2.187. N_addvar

definition of the number of %ind_addvar% to be used (legacy code)

For example,

```
N_addvar = 3
```

defines the number for the current MESHFREE simulation to be three. If, in this case, %ind_addvar(4)% is used, this will lead to serious problems.

Currently, the maximum number is 9. So, N_addvar = 10 or higher is illegal and will lead to errors.

MESHFREE · InputFiles · common_variables · Nb_InflowLayers

3.2.188. Nb_InflowLayers

For example,

```
Nb_InflowLayers = 5
```

will produce 5 layers of inflow-fluid, As default, this variable is set to 3

MESHFREE · InputFiles · common_variables · OBJ_ConvertQuadToTria

3.2.189. OBJ_ConvertQuadToTria

convert quads into triangles upon read-in

```
OBJ_ConvertQuadToTria = 1
```
Default: OBJ_ConvertQuadToTria = 0

For OBJ files convert all quads into triangles while reading in the geometry files.

MESHFREE · InputFiles · common_variables · ORGANIZE_ActivateBNDpoints_Version

3.2.191. ORGANIZE_ActivateBNDpoints_Version

define version number for the boundary point activation

First of all, for better understanding of the activation algorithm, see Illustration .

```
ORGANIZE_ActivateBNDpoints_Version = 2 # invoke the old version
```

The default value is ORGANIZE_ActivateBNDpoints_Version = 3 (the new version, additional options see FurtherOptions)

Option selection for version 3:
This version, in general, places ghost points at a small distance from free surface points in their normal direction, i.e. $\mathbf{x}_{\text{ghost}} = \mathbf{x}_{\text{FS}} + \epsilon h \cdot \mathbf{n}_{\text{FS}}$
These ghost points are considered as interior points.

- ORGANIZE_ActivateBNDpoints_Version = 30:
  measure the volume angle spanned by the local tetrahedrization;
  ignore tetras touching only free surface or boundary points (including the ghost points);
  deactivate BND-point if volume angle too small.
- ORGANIZE_ActivateBNDpoints_Version = 31: **experimental, do not use**
  same as version 30;
  additionally: boundary point is inactive if the local tetrahedrization forms open faces touching free surface points or their ghosts.

- ORGANIZE_ActivateBNDpoints_Version = 32 (is evenly the default ORGANIZE_ActivateBNDpoints_Version = 3):
  same as version 30;
  additionally: ignore any tetra touching some free surface point or its sidewise ghost.
- ORGANIZE_ActivateBNDpoints_Version = 33:
  activate BND-point, if its local tetrahedrization touches at least one interior point.
- ORGANIZE_ActivateBNDpoints_Version = 34:
  activate BND-point, if there is a path to some interior point, that does not intersect
  with a disc spanned bei either free surface of regular wall points;
  radius of the disc: $r_{\text{disc}} = 0.4 * h$

List of members:

FurtherOptions	define further options for boundary point activation versoin 3
Illustration	illustrate the idea of boundary point activation

FurtherOptions

define further options for boundary point activation versoin 3

ORGANIZE_ActivateBNDpoints_Version = (3, **N_ghostFree** , **N_ghostCentral** , **N_minFreeOrInterior** , **r_radiusCutoff** , **SolidAngle_threshold** , **dist_InteriorGhost** , **dist_BoundaryGhost** , **max_R** , **UmbrellaCheck** , **InteriorGhostForFreeSurfaceGhost**)

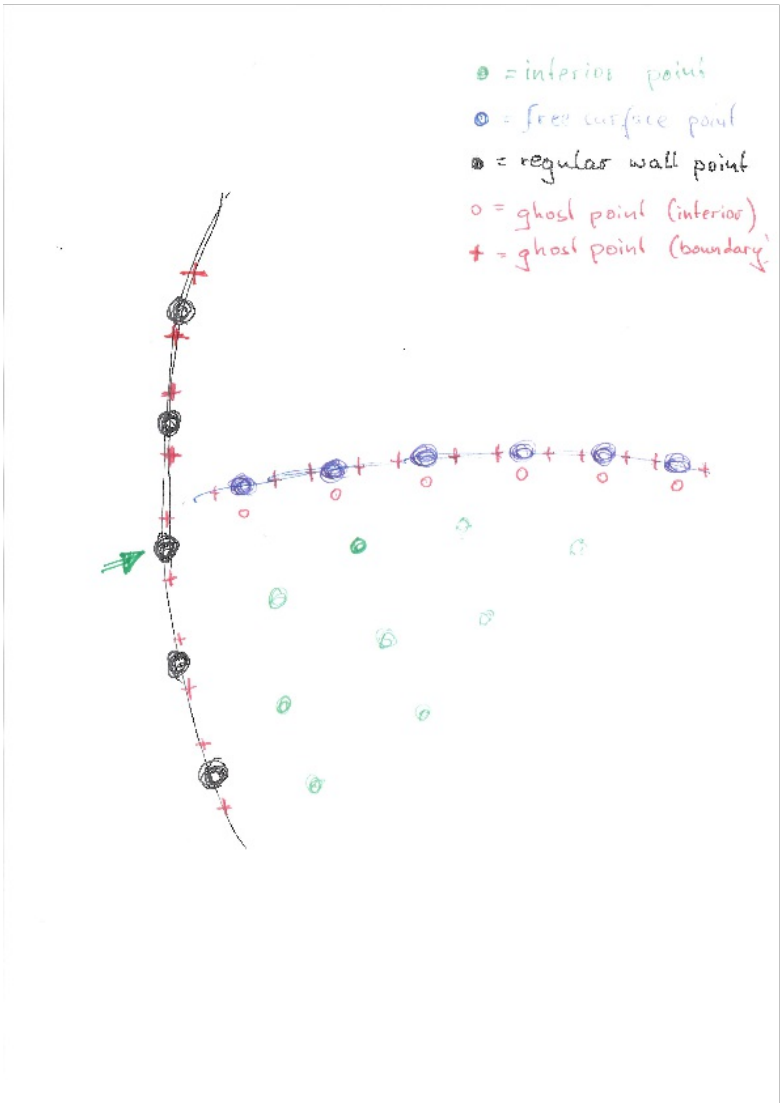
- (2) **N_ghostFree** = number of (sidewise) ghost points around each free surface point.
- (3) **N_ghostCentral** = number of ghost points around central wall point.
- (4) **N_minFreeOrInterior** = minimum number of free or interior points in order to invoke the activation procedure.
- (5) **r_radiusCutoff** = (in percent!!!) do not consider neighbors for activations whose relative radius is bigger than r_radiusCutoff/100 .
- (6) **SolidAngle_threshold** = (in percent!!!) if the relative solid angle, given by the local Delaunay triangulation, falls below **SolidAngle_threshold** /100, then the boundary point is **inactive** .
- (7) **dist_InteriorGhost** = (in percent!!!) relative distance of the interior ghost points (normal direction of any free surface point, see Illustration) is assumed to be **dist_InteriorGhost** /100 .
- (8) **dist_BoundaryGhost** = (in percent!!!) relative distance of the ghost points in normal direction of a surface point (see Illustration) is assumed to be **dist_BoundaryGhost** /100 .
- (9) **max_R** = (in percent!!!) maximum radius (relative to the SMOOTH_LENGTH) of the tetras in the tetrahedrization. If the radius of circumcircle/circumsphere exceeds **max_R** , the tetra is rejected
- (10) **UmbrellaCheck** = 1 (on) or 0 (off)
- (11) **InteriorGhostForFreeSurfaceGhost** = 1 (on) or 0 (off)

default:
ORGANIZE_ActivateBNDpoints_Version = (3, **4, 0, 3, 70, -70, 1, 10, 60, 1, 1**)
These values represent the hard coded parameters as used in ORGANIZE_ActivateBNDpoints_Version=2 .

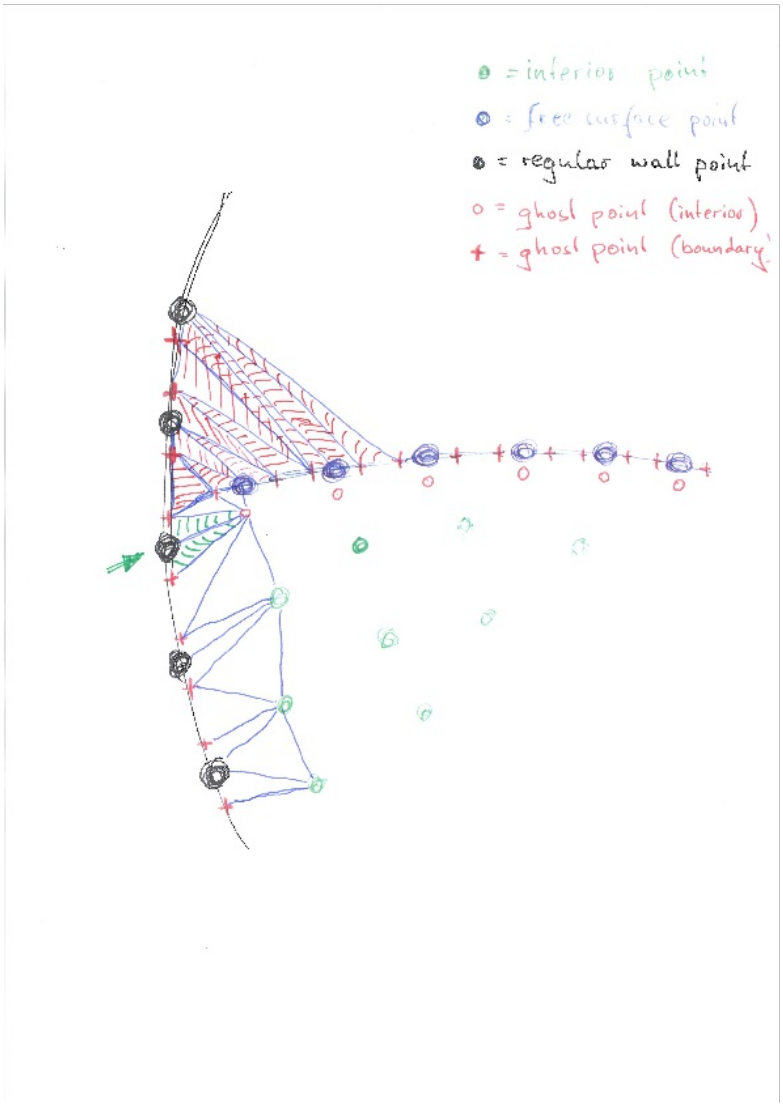
Illustration

illustrate the idea of boundary point activation

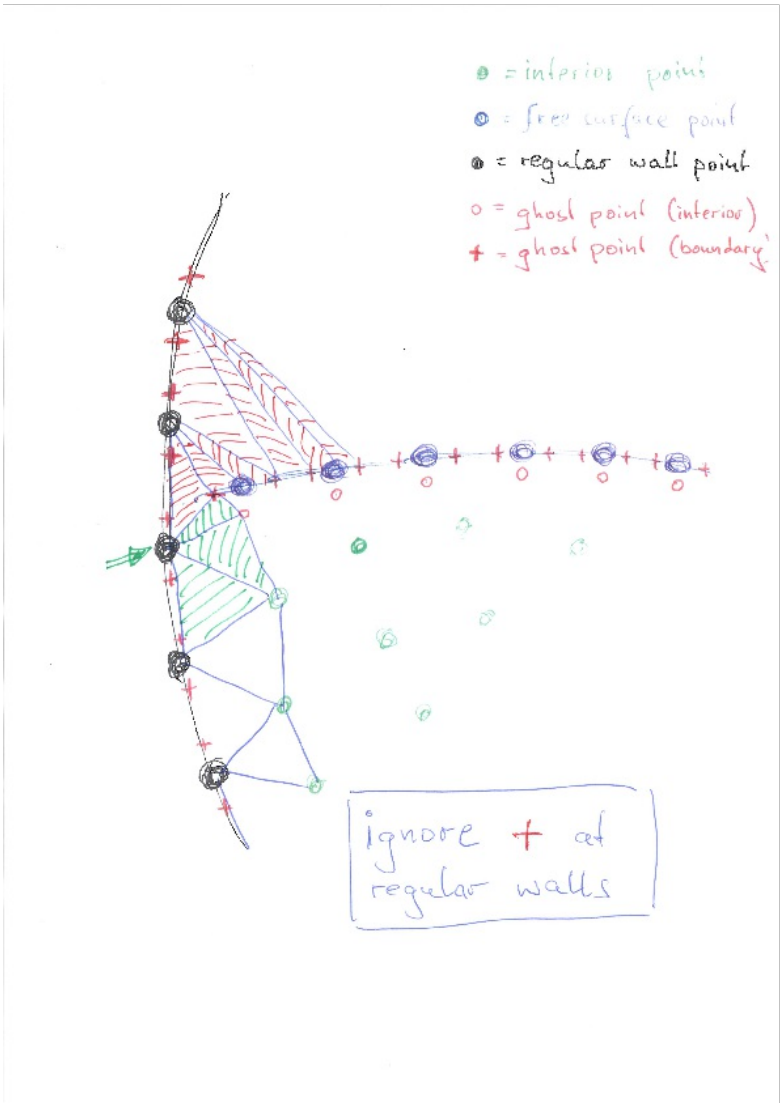
Suppose there is a local point configuration of boundary, free surface, and interior points.
We establish sidewise ghost points for free surface and boundary points. The ghost points mimic the same type of point as their origin.
For free surface points, we establish one additional ghost point in normal direction. The mimic regular interior points.



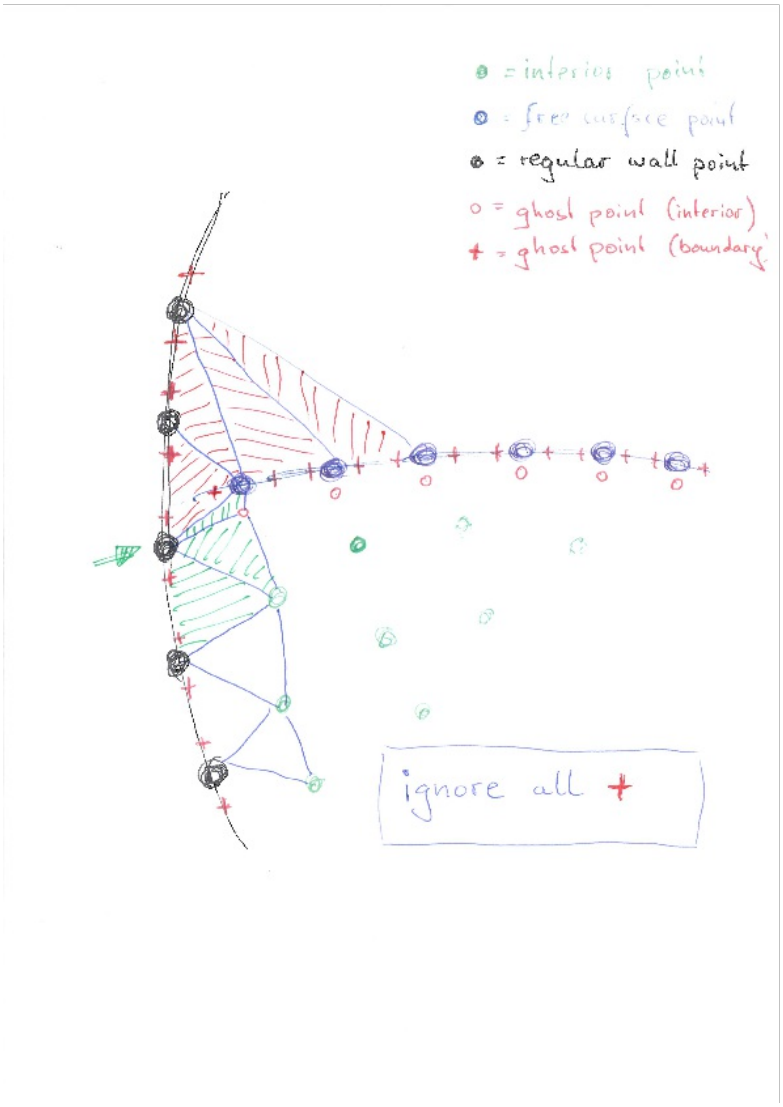
In order to judge activation of boundary points, we establish a local tetrahedrization around potentially activazed boundary points.
We neglect tetras/triangles whose corners only touch boundary or free surface points (marked in red).
We measure the opening angle of the remaining regular tetras/triangles (marked in green) and give a nondimensionalized functional spanning: 1=full half sphere, -1=zero opening angle.



The user can give the number of ghost points to be used for regular walls (see FurtherOptions).
If 0 is given, we speed up the computation, then the terehedrization looks like this:



The user can also give the number of ghost points to be used for free surface points (see FurtherOptions).
If 0 is given also here, we speed up the computation even more. In this case, the tetrahedrization looks like this:



3.2.192. ORGANIZE_BE_ClusterNodesPoints_Version

define version number for clusterig of geometry node points after geometry is read in from file (such as stl-files)

```
ORGANIZE_BE_ClusterNodesPoints_Version = 3 # current new version to be tested
```

The default value is ORGANIZE_BE_ClusterNodesPoints_Version = 2 (classical version)

The clustering combines node points of the geometry and thus established topological connectivity between BE-triangles. That is essential upon read-in of stl-files, as here triangles are originally decoupled.

Version 3 uses a smaller local search radius for neighboring/adjacent node points, saving a lot of computation time, but missing maybe some nodes to be clustered.

3.2.193. ORGANIZE_BringNewPointToFreeSurface

define maximum distance a newly created point at the free surface can be moved in order to perfectly fit the free surface

Free surface points are filled by surface triangulation, and by placing a new point at the center of a Delaunay triangle big

enough.
If the surface is curved (sphere,cylinder or the like), placing it in the plane of the triangle introduces a geometrical error and locally flattens the surface.
MESHFREE tries to correct the position of the new point towards the surface curvature of the corner points of the triangle.

The value ORGANIZE_BringNewPointToFreeSurface limits the distance by which the new location can be corrected:
maximum distance = ORGANIZE_BringNewPointToFreeSurface * SMOOTH_LENGTH

```
ORGANIZE_BringNewPointToFreeSurface = 0.3
```

The default value is ORGANIZE_BringNewPointToFreeSurface = 0.2

If setting the value of ORGANIZE_BringNewPointToFreeSurface negative, we use the least-squares-representation of the free surface given by the neighbor points.

OPTIONAL VALUE

```
ORGANIZE_BringNewPointToFreeSurface = ( 0.3, 0.8 )
```

if the cosine of the angle between free surface and boundary is bigger than the given (second) value, then switch off the bring to surface algorithm.
Current default: ORGANIZE_BringNewPointToFreeSurface = (0.2, 1.1) (bring-to-surface-algorithm always switched on)

3.2.194. ORGANIZE_CheckAllPointsForFreeSurfaceUntilTimeStep

consider all points as candidates for free surface until a given time step

```
ORGANIZE_CheckAllPointsForFreeSurfaceUntilTimeStep = 50
```

Default: ORGANIZE_CheckAllPointsForFreeSurfaceUntilTimeStep = -1

MESHFREE checks for free surface points only in the neighborhood of already existing free surface points (in order to save computation time).
With this option, we can force MESHFREE to consider all point as candidates for free surface. That would be favourable if the geometry is, for example, an overpresure valve.
The opening of the valve would generate a free surface, where there was NO fre surface before.

An alternative is ORGANIZE_CheckPointsAtFS_PerformPreCheck .

3.2.195. ORGANIZE_CheckFreeSurface_Version

define version number for the free-surface-check

```
ORGANIZE_CheckFreeSurface_Version = 2 # classical version
```

The default value is ORGANIZE_CheckFreeSurface_Version = 3 (new version with the options below)

Other values than 3 will invoke the classical version (ORGANIZE_CheckFreeSurface_Version=2).

compared to version 2, **ORGANIZE_CheckFreeSurface_Version = 3 will:**
- improve the computational performance of the free surface check,
- not make use of AdvancedFreeSurfaceAtTimeStep ,
- not make use of ORGANIZE_CheckAllPointsForFreeSurfaceUntilTimeStep ,

- indeed make use of ORGANIZE_CheckPointsAtFS_PerformPreCheck .

In general:

- The MESHFREE point $\mathbf{x}$ is a free surface point, if its local Delaunay tetrahedrization contains open faces, that means if the ring of tetras around the point is not closed.
- Let us call the corner points of the i-th local tetra by $\{\mathbf{P}_i^0, \mathbf{P}_i^1, \mathbf{P}_i^2, \mathbf{P}_i^3\}, i = 1...N$, where $\mathbf{P}_i^0$ is the central point around which the Delaunay cells are formed.
- the boundary normal is computed from the normals of the open faces, that is (in 3D):
  $\mathbf{n} = \sum_{i=1}^{N} \sum_{j=1}^{3} (\mathbf{P}_i^j - \mathbf{P}_i^0) \times (\mathbf{P}_i^{j+1} - \mathbf{P}_i^0)$, with the definition $\mathbf{P}_i^4 = \mathbf{P}_i^1$
  Of course, $\mathbf{n}$ will have to be normalized.
- Curvature computation: (to follow)
- The different options below concern different ways of computing and admitting the Delaunay tetrahedrization.

OPTIONs :

ORGANIZE_CheckFreeSurface_Version = 30

- simple local Delaunay tetrahedrization,
- the tetras is not admissible if their circumference is bigger than dist_FS_from_BND * smoothingLength.

ORGANIZE_CheckFreeSurface_Version = 31

- add a ghost point in normal direction if the point was previously a free surface point, the ghost point status is like an inner point,
- run local Delaunay tetrahedrization together with the ghost points,
- tetras are not admissible, if their circumference is bigger than dist_FS_from_BND * smoothingLength,
- tetras are not admissible, if all corner points were free surface points at the previous time step.

ORGANIZE_CheckFreeSurface_Version = 32 #(this is basically the original ORGANIZE_CheckFreeSurface_Version = 2)

- tetras are not admissible if their circumference is bigger than dist_FS_from_BND * smoothingLength,
- tetra always admissible if one of its corner points is a (previous) inner point,
- otherwise the i -th tetra is not admissible if for some of their corner points j we have
  $\mathbf{n}_i^j \cdot (\mathbf{x}_i^0 - \mathbf{P}_i^j) < \text{maxCos}$ (the normal looks "away" from the center of the circumcircle)
  where
  $\mathbf{n}_i^j$ is the computed normal of the previous time step (the point being a former free surface point) or the wall normal (the point being a regular wall point),
  $\mathbf{x}_i^0$ is the center of the circumcircle of the tetra.

ORGANIZE_CheckFreeSurface_Version = 33 #(same as ORGANIZE_CheckFreeSurface_Version = 3)

- tetras are not admissible if their circumference is bigger than dist_FS_from_BND * smoothingLength,
- a tetra is always admissible if one of its corner points is a (previous) inner point,
- otherwise the i -th tetra is not admissible if for some of their corner points j we have
  $\max_{k=1...4, k\neq j} \left(\mathbf{n}_i^j \cdot (\mathbf{P}_i^k - \mathbf{P}_i^j) \right) < \text{maxCos}$ (all corner points look "away" from the normal)
  where $\mathbf{n}_i^j$ is the computed normal of the previous time step (the point being a former free surface point) or the wall normal (the point being a regular wall point).

ORGANIZE_CheckFreeSurface_Version = 34

- add a ghost point in normal direction if the point was previously any non-interior point, the ghost point status is like an inner point,
- run local Delaunay tetrahedrization together with the ghost points,
- tetras are not admissible, if their circumference is bigger than dist_FS_from_BND * smoothingLength,
- tetras are not admissible, if all corner points were free surface points at the previous time step.

for version 2 and 3:
ORGANIZE_CheckFreeSurface_Version = 32x
ORGANIZE_CheckFreeSurface_Version = 33x
the last digits define the maximum allowed cosine-values (maxCos) for admissibility as described above: maxCos=-0.x

3.2.196. ORGANIZE_CheckPointsAtFS_PerformPreCheck

invoke additional algorithm in order to find candidates for free surface detection

In order to activate, set

```
ORGANIZE_CheckPointsAtFS_PerformPreCheck = 1
```

By default, the algorithm is switched off (ORGANIZE_CheckPointsAtFS_PerformPreCheck = 0).

- If ORGANIZE_CheckPointsAtFS_PerformPreCheck > 0, the prechecking is performed every n-th time cycle, where the number n is the given number .
- If ORGANIZE_CheckPointsAtFS_PerformPreCheck < 0, the prechecking is performed every n-th time cycle, where the number n is abs(given number) .
  In this case (number is negative), new candidates for free surface points are only searched in the neighborhood of already existing free surfaces .

In order not to miss candidates for free surface computation, the prechecking is a way to find candidates by a simple hole-search-algorithm:

- place equally distributed discrete checkpoints around a given MESHFREE point
- if some discrete check point is the center of a hole (empty ball) with radius dist_FS_from_BND * smoothingLength, the MESHFREE point is a candidate for free surface

3.2.197. ORGANIZE_DeveloppperCheck_Version

version of the debugging routine ORGANIZE_DeveloppperCheck

```
ORGANIZE_DeveloppperCheck_Version = 1
```

The default value is ORGANIZE_DeveloppperCheck_Version = 0

ORGANIZE_DeveloppperCheck_Version = 1 : just writeout the ident number for each call to ORGANIZE_DeveloppperCheck

ORGANIZE_DeveloppperCheck_Version = 2 : just writeout the ident number for each call to ORGANIZE_DeveloppperCheck and execute a call to MPI_Barrier() afterwards

3.2.198. ORGANIZE_DistanceToBoundary_Version

define version number for distance-to-boundary computations

```
ORGANIZE_DistanceToBoundary_Version = 2 # perform the classical implementation (distance-to-boundary computation for every point
# seeing a regular boundary points AND for all free surface points)
```

The default value is ORGANIZE_DistanceToBoundary_Version = 3 (current version)

Additional options for version 3

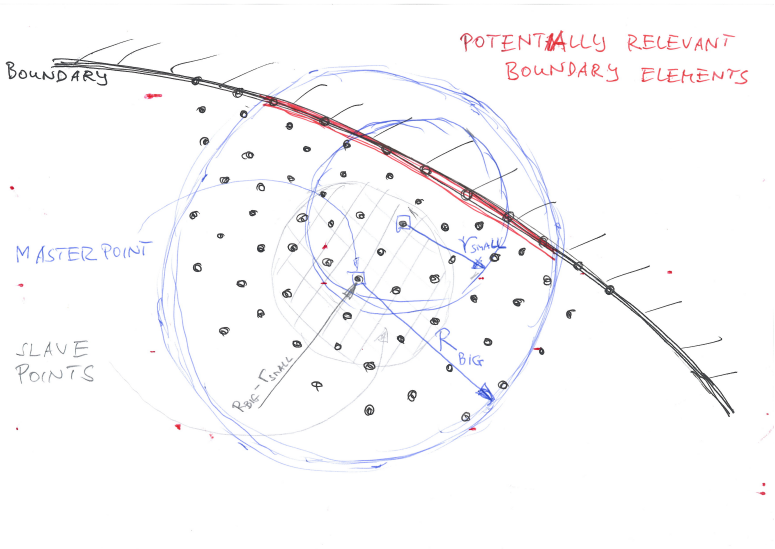
ORGANIZE_DistanceToBoundary_Version = (3, **radiusBig** , **radiusSmall** , NN)

Default: ORGANIZE_DistanceToBoundary_Version = (3, **100** , **50** , 0)

- **radiusBig** : (integer value to be GIVEN IN PERCENT!)
  - A point "i" looks within a ball of radius = (**radiusBig** /100)*SMOOTH_LENGTH for boundary elements (BE) like trias, quads, etc., and solves the boundary distance based on the found BE-list.
  - Neighbor points of "i" closer than (**radiusBig** /100 - **radiusSmall** /100)*SMOOTH_LENGTH use the same neighbor list (thus saving a lot of compution time as retrieving the BE-list from the search tree might be costly if the geometry is of much finer resolution than local SMOOTH_LENGTH). The calculated distance is valid only if it is smaller than (**radiusSmall** /100)*SMOOTH_LENGTH .

- **radiusSmall** : (integer value to be GIVEN IN PERCENT!)
  effective radius of BE-search and distance-to-boundary computation.
- NN: not used

ORGANIZE_DistanceToBoundary_Version = (3, 100, 100, 0) -> original idea of version 2. Slighly better in performance,
ORGANIZE_DistanceToBoundary_Version = (3, 50, 50, 0) -> keeps the neighborlists smaller than 100,100, so much less computational effort, but BE-list to be computed for every point,
ORGANIZE_DistanceToBoundary_Version = (3, 100, 50, 0) -> BE-list to be computed for only a few points, exclusion of points that do not need explicit distance computation,
ORGANIZE_DistanceToBoundary_Version = (3, 60, 30, 0) -> can be even more efficient, make sure that MESHFREE points do not travel more than 30 percent of the SMOOTH_LENGTH per time step.



3.2.199. ORGANIZE_ForceInsideCheckForAllParticles

inside-check for all MESHFREE points

ORGANIZE_ForceInsideCheckForAllParticles = 1

Default: ORGANIZE_ForceInsideCheckForAllParticles = 0

If set to 1, it forces an explicit inside-check for all MESHFREE points one time per time step.

Otherwise, MESHFREE points are inside-checked only if boundary points are in the neighborhood.

For all other points, they are assumed to be inside.

3.2.200. ORGANIZE_ForceInsideCheckForNewParticles

inside-check for new MESHFREE points

```
ORGANIZE_ForceInsideCheckForNewParticles = 0
```

Default: ORGANIZE_ForceInsideCheckForNewParticles = 1

If set to 1, it forces an explicit inside-check for all newly created MESHFREE points.

Otherwise, new MESHFREE points are inside-checked only if boundary points are in the neighborhood.
For all other new points it is assumed that they are inside.

3.2.201. ORGANIZE_ForceTouchCheckAtWalls

touch-check for MESHFREE points at walls

```
ORGANIZE_ForceTouchCheckAtWalls = 1
```

Default: ORGANIZE_ForceTouchCheckAtWalls = 0

If set to 1, it forces an explicit activation-check for all MESHFREE points at a boundary, whose TOUCH -flag is %TOUCH_liquid% or %TOUCH_solid%
Otherwise, MESHFREE points are checked for activation only if free surface points are in the neighborhood.
For all other points, they are assumed to be active, if at least one interior point is in the neighborhood.

3.2.202. ORGANIZE_FuzzyMPIFilling

(chamberwise) parameter to allow MPI processes to fill points outside their own domain

```
ORGANIZE_FuzzyMPIFilling = 1
```

Default: ORGANIZE_FuzzyMPIFilling = 0 (off)

By default, each MPI process can only fill new points within its assigned domain.
This can sometimes lead to some MPI domains not being filled sufficiently.
With this setting on, each MPI process can fill points in a thin layer outside
its assigned domain as well (fuzzy). In a future filling step, these points are
redistributed to the neighboring MPI domain such that the neighboring MPI process
can continue filling its domain.

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

3.2.203. ORGANIZE_OppositePoints_Version

define version number for detecting points of the other phase to be coupled (opposite points)

```
ORGANIZE_OppositePoints_Version = 2 # is required for the boundary condition BCON_CNTCT (0,%ind_v(1)%) = (%BND_slip_InContact%, 0.0, 0.3) to work
```

The default value is ORGANIZE_OppositePoints_Version = 3

In most cases version 3 is working, but for %BND_slip_InContact% we still need version 2!

Options:

```
ORGANIZE_OppositePoints_Version = (3,1) #default
```

- (3,1) is the default version.
- It checks the type (%ind_kob%) of the opposite point and allows only opposite points from the same type. This means:
- Free surface points are allowed to have only free surface opposite partners!
- Regular boundary points are allowed to have only regular boundary opposite partners!

```
ORGANIZE_OppositePoints_Version = (3,2)
```

- It does not use the above mentioned checks so that free surface points can also interact with regular boundary points.
- Be very careful with this option! Only use it if you know exactly how you want to couple your phases!

3.2.204. ORGANIZE_PSTOneReductionStep_Version

version how to reduce MESHFREE points if they come to close to each other

```
ORGANIZE_PSTOneReductionStep_Version = 2
```

Default: ORGANIZE_PSTOneReductionStep_Version = 1

ORGANIZE_PSTOneReductionStep_Version==1 : points are remove immediately if they are detected to be too close
ORGANIZE_PSTOneReductionStep_Version>=2 : if detected to be too close, points are marked by Y%ind_vol%=0, Y%ind_act%=-1, and Y%ind_OrganizePC(2)%=-1

3.2.205. ORGANIZE_PSTOneRefillStep3_UseFromWhichTime

use the new implementation of PST_OneRefillStep_3 from which time

Example:

```
ORGANIZE_PSTOneRefillStep3_UseFromWhichTime = 0.1
```

The default value is ORGANIZE_PSTOneRefillStep3_UseFromWhichTime = -1.0.

If negative, the old version PST_OneRefillStep_2 is used throughout the simulation.
If positive, the old version PST_OneRefillStep_2 is used until the given time, then the new version PST_OneRefillStep_3 is employed.

This is temporary until PST_OneRefillStep_3 has become standard.

3.2.206. ORGANIZE_PSTOneRefillStep3_UseFromWhichTimeStep

use the new implementation of PST_OneRefillStep_3 from which time step

Example:

```
ORGANIZE_PSTOneRefillStep3_UseFromWhichTimeStep = 2
```

The default value is ORGANIZE_PSTOneRefillStep3_UseFromWhichTimeStep = -1.

If negative, the old version PST_OneRefillStep_2 is used throughout the simulation.
If positive, the old version PST_OneRefillStep_2 is used until the given time step-1, then the new version PST_OneRefillStep_3 is employed.

This is temporary until PST_OneRefillStep_3 has become standard.

3.2.207. ORGANIZE_PreAllocationSize

define version number for distance-to-boundary computations

```
ORGANIZE_PreAllocationSize = 300000 # preallocation of Y, AAA, and CON- arrays for 300000
```

The default value is ORGANIZE_PreAllocationSize = -1 (classical version, no preallocation)

3.2.208. ORGANIZE_QualityCheck_ListNbOfNeighbors

number of neighbors per point for which the quality check has to be performed

Check the quality of the point cloud for each MESHFREE point for a different number of neighbors (maximum 3 different values).
With this variables we can define the number of neighbors we wish to check.

```
ORGANIZE_QualityCheck_ListNbOfNeighbors = (30, 40, 50) # check the quality for the closest 30, 40, and 50 neighbors to each MESHFREE point
```

Default: ORGANIZE_QualityCheck_ListNbOfNeighbors = (25, 40, 60)

3.2.209. ORGANIZE_ReducedFillingOfWalls

(chamberwise) parameter for reduced filling of boundaries marked as walls

```
ORGANIZE_ReducedFillingOfWalls = 0
```

Default: ORGANIZE_ReducedFillingOfWalls = 1 (on)

Reduced filling of those boundaries marked with IDENT%BND_wall%, IDENT%BND_slip%, or IDENT%BND_wall_nosl%: Boundary points are removed from the walls if no interior point is found in the neighborhood.

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

3.2.211. ORGANIZE_RefillOnlyForActiveBoundaryParticles

(chamberwise) parameter to trigger the point refilling procedure along the boundary only for active boundary points

```
ORGANIZE_RefillOnlyForActiveBoundaryParticles = 1
```

Default: ORGANIZE_RefillOnlyForActiveBoundaryParticles = 0

In regular intervals, the point cloud along the boundary is check for quality and points are filled or removed.
If ORGANIZE_RefillOnlyForActiveBoundaryParticles = 1, this action is executed ONLY for active boundary points.
Inactive boundary points are kept as they are.

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

3.2.213. ORGANIZE_USER_update_boundary_particles_Version

version of USER_update_boundary_particles.f90 to be used

```
ORGANIZE_USER_update_boundary_particles_Version = 2
```

The default value is ORGANIZE_USER_update_boundary_particles_Version = 3
If COMP_SharedMemoryForBE = true, then ORGANIZE_USER_update_boundary_particles_Version=2 will not work.

- Version 1 is original.
- Version 2 tries to distribute the computations of the geometry nodes to several MPI-processes, and then broadcast the data by MPI_bcast.
- Version 3 implements MPI-shared-memory movement of the boundary. Aditionally, it does not touch boundary elements which are flagged by MOVE -1 (see ALIAS attributes)

Special feature for version 2:

```
ORGANIZE_USER_update_boundary_particles_Version = 264
```

This invokes version 2 (first integer digit), but also tells MESHFREE to use no more than 64 broadcasting processes.
In this logic,

```
ORGANIZE_USER_update_boundary_particles_Version = 20
ORGANIZE_USER_update_boundary_particles_Version = 2
```

is the same, and 0 broadcasting processes means no broadcasting at all, rather each process computes all necessary

movement data on its own.

MESHFREE · InputFiles · common_variables · PHASE_distinction

3.2.217. PHASE_distinction

invoke detection of interface connections (CV)

See PHASE_distinction . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · PointDsplMethod

3.2.218. PointDsplMethod

(experimental) Choice among different ways to move points in Lagrangian framework (CV)

See PointDsplMethod . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · RESTART_useSTREAMfile

3.2.223. RESTART_useSTREAMfile

use the STREAM inp/output for restart files

```
RESTART_useSTREAMfile = true
```

Default: RESTART_useSTREAMfile = false

MESHFREE · InputFiles · common_variables · RIGIDBODY_TimeIntegrationDamping

3.2.224. RIGIDBODY_TimeIntegrationDamping

Numerically damping of the time integration

instead of putting 1 in the diagonal of the linear system, add RIGIDBODY_TimeIntegrationDamping to the diagonal of the linear system for the rigid bodies.

```
RIGIDBODY_TimeIntegrationDamping = 1.0e-4
```

Default: RIGIDBODY_TimeIntegrationDamping = 0.0

This term provoks additional damping by enhancing the differential equation of the velocity of a rigid body by

$$m\frac{d\mathbf{v}}{dt} = F - \beta\mathbf{v}$$

which numerically is solved by

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \frac{\Delta t}{m}F - \frac{\Delta t}{m}\beta\mathbf{v}^{n+1}$$

RIGIDBODY_TimeIntegrationDamping = $\frac{\Delta t}{m}\beta$
There is, so far, no physical motivation behind this, only numerical stabilization in some critical applications.
See also RIGIDBODY .

3.2.225. RIGIDBODY_TimeIntegrationPPI

Tichonov-regularization parameter for rigid bodies with links or intersections

regularize the linear system of equations concerning rigid bodies, that are connceted by links or intersections.

RIGIDBODY_TimeIntegrationPPI = 1.0e-4

Default: RIGIDBODY_TimeIntegrationPPI = 1.0e-10
See also RIGIDBODY .

3.2.226. RIGIDBODY_TimeIntegrationVersion

choose time integration version (still experimental)

If the MOVE statement of a geometrical unit / body is %MOVE_rigid% (see there) then the time integration of the equations of motion are solved explicitly.
Version 2 mainly allows also body-body and body-boundary collisions. For this, the time integration of the rotation has to be reduced from quasi-analytical
to second order in time.

RIGIDBODY_TimeIntegrationVersion = 2 , OPTIONAL: N_sub , OPTIONAL: dt_fix

Default: RIGIDBODY_TimeIntegrationVersion = 1
The collision model (see RIGIDBODY_UseCollisionModel) can only be chosen with version 2.

Optional arguments:

- N_sub: define the number of sub-iterations for the rigid body structure per MESHFREE time cycle, so the numerical time step size for the rigid body structure (RB) would be $\Delta t_{\mathrm{sub}} = \frac{\Delta t_{\mathrm{MESHFREE}}}{N_{\mathrm{sub}}}$
- dt_fix: define numerical time step size for the rigid body structure

Taking into account the optional arguments, the numerical time step size for the rigid body structure is $\Delta t_{\mathrm{RB}} = \min\left(\Delta t_{\mathrm{sub}}, \Delta t_{\mathrm{fix}}\right)$

- Version 1: second order for the velocity, but quasianalytical for the rotation (exact integration of the Euler equation for rotation)
- Version 2: second order for velocity and rotation (in this way, implicit collision and joint/link forces can be taken into account)

See also RIGIDBODY .

3.2.227. RIGIDBODY_UseCollisionModel

switch on the collision model for rigs bodies (rigid-wall and rigid-rigid)

RIGIDBODY_UseCollisionModel = true

Default: RIGIDBODY_UseCollisionModel = false
See DOCUMATH_RigidBodyCollisions.pdf for a detailed discussion of the way it is implemented.

See also RIGIDBODY .

MESHFREE · InputFiles · common_variables · RepairGeometry

3.2.228. RepairGeometry

enforce clustering of geometry nodes upon read-in (CV)

See RepairGeometry . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · RepresentativeMass_iData

3.2.229. RepresentativeMass_iData

(chamberwise) parameter for the RepresentativeMass algorithm (CV)

See RepresentativeMass_iData . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · SAMG_Setupreuse

3.2.230. SAMG_Setupreuse

accelerates SAMG solver for quasi-stationary point clouds (CV)

See SAMG_Setupreuse . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · SAVE_ASCII_split

3.2.231. SAVE_ASCII_split

splits ASCII output files if larger than 2GB

```
SAVE_ASCII_split = 1
```

Default: SAVE_ASCII_split = 0 (do not split ASCII files)

SAVE_ASCII_split = 1: split ASCII file into multiple files per timestep if larger than 2GB
SAVE_ASCII_split = 2: after splitting automatically merge if supported by the command CommonVar
SAVE_ASCII_split = 3: try closing and then reopening the file after 2GB were written

MESHFREE · InputFiles · common_variables · SAVE_PrecisionTimestepFile

3.2.232. SAVE_PrecisionTimestepFile

choose the precision (number of digits) for values in the timestep file (CV)

See SAVE_PrecisionTimestepFile . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · SAVE_atEndOfTimestep

3.2.234. SAVE_atEndOfTimestep

choose to save data for visualization at the end of time steps instead of at the start (CV)

See SAVE_atEndOfTimestep . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · SCAN_ClustersOfConnectivity

3.2.235. SCAN_ClustersOfConnectivity

(chamberwise) switch on cluster checking of MESHFREE point cloud by neighborhood connectivity (CV)

See SCAN_ClustersOfConnectivity . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · SIGNAL_LaunchComputationalSteering

3.2.236. SIGNAL_LaunchComputationalSteering

Switch between the two options of computational steering

See ComputationalSteering .

MESHFREE · InputFiles · common_variables · SPM_Regularization_Epsilon

3.2.247. SPM_Regularization_Epsilon

adjust numerical parameter epsilon for the matrix regularizations

Especially see SPM_Regularization_Type .
Adjust the parameter epsilon, stated there, in the common_variables.dat file.

```
SPM_Regularization_Epsilon = 1.0e-2
```

Default: SPM_Regularization_Epsilon = 1.0e-6

MESHFREE · InputFiles · common_variables · SPM_Regularization_Type

3.2.248. SPM_Regularization_Type

regularization type if all boundaries are Neumann-type

The PDEs solved in MESHFREE for the pressure are of the form $\Delta p = f$ with the boundary conditions $p = A$ (Dirichlet) or $\frac{\partial p}{\partial n} = B$ (Neumann).
If, for the whole domain, only Neumann-conditions are given, the arising linear sparse system is singular and has to be regularized.

```
SPM_Regularization_Type = 1
```

Default: SPM_Regularization_Type = 2

Type 1: Instead of solving $\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}$, we solve the perturbed system
$(\boldsymbol{A} + \epsilon \boldsymbol{I}) \cdot \boldsymbol{x} = \boldsymbol{b}$

where $\boldsymbol{I}$ is the identity matrix

Type 2: Instead of solving $\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}$, we solve the perturbed system
$(\boldsymbol{A} + \epsilon \boldsymbol{J}) \cdot \boldsymbol{x} = \boldsymbol{b}$

where $\boldsymbol{J}$ is a matrix that contains 1 in all entries. This amounts to weakly requiring that the sum of the result vector entries is zero, i.e. $\sum \boldsymbol{x}_i = 0$
Esilon can be adjusted in SPM_Regularization_Epsilon .

MESHFREE · InputFiles · common_variables · SPM_matrix_times_vector_Version

3.2.249. SPM_matrix_times_vector_Version

version for the matrix-times-verctor operations for sparse linear systems

version 1: -> automatically switch to version 2
version 2: classical loop over all matrix lines: do i = 1,nMatrixLines)
version 3: vectorize the loop over all matrix lines: do i = 1,nMatrixLines,i4 :: MxV(i:i+i4-1) = sum{ (MMM(i:i+i4-1,.) * X(.) } assuming that the intrinsic sum() function of INTEL-fortran is also perfectly vectorized.

```
SPM_matrix_times_vector_Version = 3 # supposed to be faster as vectorizing some loops
```

Default: SPM_matrix_times_vector_Version = 1

MESHFREE · InputFiles · common_variables · STRESSTENSOR_Variante_Factor

3.2.251. STRESSTENSOR_Variante_Factor

factor in stress tensor time integration wrt the shear modulus (CV)

See STRESSTENSOR_Variante_Factor . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · STRESSTENSOR_Variante

3.2.252. STRESSTENSOR_Variante

version of stress tensor time integration (CV)

See STRESSTENSOR_Variante . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · SUBSTEPS_EXPL

3.2.253. SUBSTEPS_EXPL

number of explicit substeps for solving TRANSPORT part in EULEREXPL setting

```
SUBSTEPS_EXPL = 5.0
```

Default: SUBSTEPS_EXPL = 8.0

Similar to SUBSTEPS_IMPL , however SUBSTEPS_EXPL controls only the number of the explicit time integration substeps for solving the transport terms.

MESHFREE · InputFiles · common_variables · SUBSTEPS_IMPL

3.2.254. SUBSTEPS_IMPL

number of implicit substeps with constant time step size in EULERIMPL setting

SUBSTEPS_IMPL = 5.0

Default: SUBSTEPS_IMPL = 1.0

In order to save more computation time within the EULERIMPL scheme the MESHFREE time step size ΔT_k can be increased by a factor of SUBSTEPS_IMPL .
If the automatic time step size control method based on local errors (see TOL_T , TOL_v , TOL_keps) determines a time step size Δt_k ,
then the size of the next MESHFREE time step will be

$$\Delta T_k = \text{SUBSTEPS\_IMPL} \cdot \Delta t_k \ .$$

In order to conserve the given error tolerance the physical entities must be computed with Δt_k , so that SUBSTEPS_IMPL steps are needed
to compute one MESHFREE time step. Furthermore during one MESHFREE step a constant Δt_k is used in order to reuse the same matrix, what saves
a lot of computation time.

Remark: Take care that SUBSTEPS_IMPL is not bigger than 10! Otherwise the automatic time step size control method based on local errors will reject the result
and repeat the substep with a smaller Δt_k . Repetitions of substeps can be suppressed by setting

NB_OF_ACCEPTED_REPETITIONS = 0,

but then a big SUBSTEPS_IMPL will lead to very inaccurate results. Therefore we recommend SUBSTEPS_IMPL $\in [2, 6]$ to avoid repetitions.

3.2.258. SimCut

(chamberwise) parameter to stop filling of geometry by MESHFREE points after a certain number of filling cycles

SimCut = 10

Default: SimCut = 0

If a positve number is given, MESHFREE will perform the given number of filling iterations
for interior points. Then it generates an output as defined in Saving and Results, after that MESHFREE stops.

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

Special feature:

SimCut = -10

With a negative number for SimCut , MESHFREE will perform abs(SimCut) filling cycles for interior points.
After this, the simulation is started regularly.

3.2.259. SimCutBoundary

(chamberwise) parameter to stop filling of boundary by MESHFREE points after a certain number of filling cycles

SimCutBoundary = 10

Default: SimCutBoundary = 1000

If a positve number is given, then MESHFREE will perform the given number of filling iterations
for boundary points.

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

MESHFREE · InputFiles · common_variables · SkipMarkingPointsLayer2

3.2.260. SkipMarkingPointsLayer2

(experimental) switch for marking the second layer near the boundary in EULERIMPL setting

SkipMarkingPointsLayer2 = 'YES'

Default: SkipMarkingPointsLayer2 = 'NON'

If EULERIMPL is used, then all points are marked in an environment of $2 \times h$ near the boundary. This information
is needed for the special boundary treatment in the MUSCL reconstruction scheme, see SpecialBNDtreatmentEULERIMPL
,
pure_TRANSPORT=1, additionalPoint_approximation . In many cases it is sufficient regarding the accuracy only to mark the points
in an environment of h . For that the user can skip the second layer by setting SkipMarkingPointsLayer2 = 'YES'.
If the results are still satisfying, this option can save a lot of computation time.

MESHFREE · InputFiles · common_variables · SpecialBNDtreatmentEULERIMPL

3.2.261. SpecialBNDtreatmentEULERIMPL

(experimental) switch for special boundary treatment for MUSCL reconstruction in EULERIMPL scheme

SpecialBNDtreatmentEULERIMPL = (1,0,1)

Default: SpecialBNDtreatmentEULERIMPL = (1,1,1)

If EULERIMPL is used, then by default for all points close to the boundary it is checked if the auxiliary points for the MUSCL
reconstruction (see pure_TRANSPORT=1) are outside the domain. If an auxiliary point is outside, then the value of the nearest
boundary point is projected to the auxiliary point. Otherwise the auxiliary point is computed based on interior neighbor points.

In cases with long thin geometries, this treatment could lead to incorrect results because of the dominating effect of boundary
values. For example in a long thin tube, the auxiliary points are almost always outside the domain for each interior point. If then the boundary values of the closest boundary points are always used, the boundary conditions overlay the interior domain.
This leads to non-physically fast in/decrease of temperature or/and wrong velocity profiles. Therefore in such cases it can

be
helpful to switch off this treatment for single or even all quantities. It can be controlled by using the integer array

```
SpecialBNDtreatmentEULERIMPL = ( 1 , 0 , 1 )
```

where the first entry controls the temperature , the second one the velocity and
the third one the k-epsilon model. The value 0 means that the boundary treatment is switched off and 1
means it is active. Thus (1,0,1) means that for temperature and k-epsilon the boundary treatment is active, whereas for the

velocity it is switched off. Furthermore it can help to reduce the number of points, which are marked near the boundary for this treatment by skipping the second layer, see SkipMarkingPointsLayer2 . Moreover it is highly recommended to check the
influence of the parameter StencilOrderReductionNearBND_forEULERIMPL . For long thin geometries it can be necessary in terms off
accuracy to switch off the stencil order reduction.

In order to switch off the entire boundary treatment for EULERIMPL one must set:

```
SpecialBNDtreatmentEULERIMPL = (0,0,0,0)
```

Remark: But keep in mind that switching off this boundary treatment means that all auxiliary points for MUSCL are approximated
based on interior points, even if they are outside the domain!

Under additionalPoint_approximation you will find the explanation for the approximation of the auxiliary points based on interior points.

3.2.262. StencilOrderReductionNearBND_forEULERIMPL

(experimental) switch for order reduction of x,y,z-derivative stencils in EULERIMPL setting

```
StencilOrderReductionNearBND_forEULERIMPL = 'NON'
```

Default: StencilOrderReductionNearBND_forEULERIMPL = 'YES'

I f EULERIMPL is used, then due to stabilization issues by default the order of the x,y,z-derivative stencils (see ord_gradient)
near the boundary (in an environment of h) is reduced by one. In some cases it can be helpful in terms of accuracy to switch it
off by StencilOrderReductionNearBND_forEULERIMPL = 'NON'. For example in cases with long thin geometries (see SpecialBNDtreatmentEULERIMPL) it can greatly improve the results, if it is still stable.

3.2.263. SurfaceTesselationActiveBoundary_cRadius

radius of the basic disc for the surface tesselation cells on active boundary, including free surface, excluding inactive points

```
SurfaceTesselationActiveBoundary_cRadius = 0.35
```

Default: SurfaceTesselationActiveBoundary_cRadius = -1.0 (i.e. no surface tesselation is executed)
The surface tesselation is performed in the follwoing way. First, each boundary points obtains a circular disc
with the radius SurfaceTesselationActiveBoundary_cRadius *SmoothingLength. Then the discs cut each other, such that, in the oprimal case,

we obtain a set of non-overlapping tesselation cells.

MESHFREE · InputFiles · common_variables · SurfaceTesselationRegularBoundary_cRadius

3.2.264. SurfaceTesselationRegularBoundary_cRadius

radius of the basic disc for the surface tesselation cells on regular boundary

```
SurfaceTesselationRegularBoundary_cRadius = 0.35
```

Default: SurfaceTesselationRegularBoundary_cRadius = -1.0 (i.e. no surface tesselation is executed)
The surface tesselation is performed in the follwoing way. First, each boundary points obtains a circular disc with the radius SurfaceTesselationRegularBoundary_cRadius *SmoothingLength. Then the discs cut each other, such that, in the oprimal case,
we obtain a set of non-overlapping tesselation cells.

MESHFREE · InputFiles · common_variables · TIMECHECK_Level

3.2.265. TIMECHECK_Level

time check only up to a given level

This parameter defines the hierarchy level up to which the time measurements are performed (see TIMECHECK).
COMP_TimeCheck controls the type of writeout.

```
TIMECHECK_Level = 1
```

Default: TIMECHECK_Level = 3

The level is given by the point-separators in the name of the TIMECHECK -item (see NamesOfStopWatches)
The stop watch ADMIN_TIME_INTEG.ORGANIZE.NEIGHBORLISTREDUCTION.CC2 (for example) is level 4.

MESHFREE · InputFiles · common_variables · TOL_T

3.2.266. TOL_T

(control of time step size) error tolerance for computing the temperature using SDIRK2 method in EULERIMPL setting

```
TOL_T = 1.0e-4
```

Default: TOL_T = 3.0e-4

The EULERIMPL scheme is a fully implicit method, which does not need to fulfill the CFL condition. Therefore the time step size is computed dependent on a given tolerance

$$\mathrm{TOL} := \mathrm{RTOL}\,\|\mathbf{U}^k\| + \mathrm{ATOL},$$

whereby in this case $\mathrm{RTOL} = \mathrm{ATOL} =$ TOL_T is determined by the user.
With the help of a proportional-integral (PI) controller the time step size is computed by

$$\Delta t_{k+1} = \begin{cases} \Delta t_k \left(\dfrac{\theta\,\mathrm{TOL}}{\|\hat{\mathbf{e}}_{k+1}\|} \right)^{\beta_I/(\hat{p}+1)} \left(\dfrac{\|\hat{\mathbf{e}}_k\|}{\|\hat{\mathbf{e}}_{k+1}\|} \right)^{\beta_P/(\hat{p}+1)}, & \dfrac{\|\hat{\mathbf{e}}_{k+1}\|}{\mathrm{TOL}} \le 1.2 \\ \Delta t_k \left(\dfrac{\theta\,\mathrm{TOL}}{\|\hat{\mathbf{e}}_{k+1}\|} \right)^{1/(\hat{p}+1)}, & \dfrac{\|\hat{\mathbf{e}}_{k+1}\|}{\mathrm{TOL}} > 1.2 \end{cases}.$$

The local error estimator $\|\hat{\mathbf{e}}_{k+1}\| = \|\mathbf{U}^{k+1} - \hat{\mathbf{U}}^{k+1}\|$ is computed by using an embedded Runge-Kutta method where two results of different order
are compared. Due to the use of the SDIRK2 method (2nd order) the result $\hat{\mathbf{U}}^{k+1}$ is based on a method of order $\hat{p} = 1$. The other parameters are

$$\theta = 0.8, \qquad \beta_I = 0.3, \qquad \beta_P = 0.4.$$

Remark: For solving the ODE $\dot{\mathbf{U}} = \mathbf{F}(t, \mathbf{U})$, the Singly Diagonally Implicit Runge Kutta (SDIRK2) method

$$\boldsymbol{\eta}_1 = \mathbf{U}^k + \Delta t\, \alpha\, \mathbf{F}\big(t^k + \alpha\, \Delta t,\, \boldsymbol{\eta}_1\big),$$

$$\mathbf{U}^{k+1} = \mathbf{U}^k + \Delta t \Big((1-\alpha)\, \mathbf{F}\big(t^k + \alpha\, \Delta t,\, \boldsymbol{\eta}_1\big) + \alpha\, \mathbf{F}\big(t^k + \Delta t,\, \mathbf{U}^{k+1}\big) \Big),$$

$$\alpha = 1 - \frac{\sqrt{2}}{2}$$

is used, which is of second order accuracy. That is why it is abbreviated as SDIRK2 .
See time_integration_impl , TOL_v and TOL_keps .

3.2.267. TOL_keps

(control of time step size) error tolerance for computing the k-epsilon model using SDIRK2 method in EULERIMPL setting

TOL_keps = 1.0e-2

Default: TOL_keps = 5.0e-2

The EULERIMPL scheme is a fully implicit method, which does not need to fulfill the CFL condition. Therefore the time step size is computed dependent on a given tolerance

$$\mathrm{TOL} := \mathrm{RTOL}\, \|\mathbf{U}^k\| + \mathrm{ATOL},$$

whereby in this case $\mathrm{RTOL} = \mathrm{ATOL} =$ TOL_keps is determined by the user.
With the help of a proportional-integral (PI) controller the time step size is computed by

$$\Delta t_{k+1} = \begin{cases} \Delta t_k \left(\dfrac{\theta\,\mathrm{TOL}}{\|\hat{\mathbf{e}}_{k+1}\|} \right)^{\beta_I/(\hat{p}+1)} \left(\dfrac{\|\hat{\mathbf{e}}_k\|}{\|\hat{\mathbf{e}}_{k+1}\|} \right)^{\beta_P/(\hat{p}+1)}, & \dfrac{\|\hat{\mathbf{e}}_{k+1}\|}{\mathrm{TOL}} \le 1.2 \\ \Delta t_k \left(\dfrac{\theta\,\mathrm{TOL}}{\|\hat{\mathbf{e}}_{k+1}\|} \right)^{1/(\hat{p}+1)}, & \dfrac{\|\hat{\mathbf{e}}_{k+1}\|}{\mathrm{TOL}} > 1.2 \end{cases}.$$

The local error estimator $\|\hat{\mathbf{e}}_{k+1}\| = \|\mathbf{U}^{k+1} - \hat{\mathbf{U}}^{k+1}\|$ is computed by using an embedded Runge-Kutta method where two results of different order
are compared. Due to the use of the SDIRK2 method (2nd order) the result $\hat{\mathbf{U}}^{k+1}$ is based on a method of order $\hat{p} = 1$. The other parameters are

$$\theta = 0.8, \qquad \beta_I = 0.3, \qquad \beta_P = 0.4.$$

Remark: For solving the ODE $\dot{\mathbf{U}} = \mathbf{F}(t, \mathbf{U})$, the Singly Diagonally Implicit Runge Kutta (SDIRK2) method

$$\boldsymbol{\eta}_1 = \mathbf{U}^k + \Delta t\, \alpha\, \mathbf{F}\big(t^k + \alpha\, \Delta t,\, \boldsymbol{\eta}_1\big),$$

$$\mathbf{U}^{k+1} = \mathbf{U}^k + \Delta t \Big((1-\alpha)\, \mathbf{F}\big(t^k + \alpha\, \Delta t,\, \boldsymbol{\eta}_1\big) + \alpha\, \mathbf{F}\big(t^k + \Delta t,\, \mathbf{U}^{k+1}\big) \Big),$$

$$\alpha = 1 - \frac{\sqrt{2}}{2}$$

is used, which is of second order accuracy. That is why it is abbreviated as SDIRK2 .
See time_integration_impl , TOL_T and TOL_v .

3.2.268. TOL_v

(control of time step size) error tolerance for computing the velocity using SDIRK2 method in EULERIMPL setting

TOL_v = 1.0e-3

Default: TOL_v = 2.0e-3

The EULERIMPL scheme is a fully implicit method, which does not need to fulfill the CFL condition. Therefore the time step size is computed dependent on a given tolerance

$$\mathrm{TOL} := \mathrm{RTOL}\, \|\mathbf{U}^k\| + \mathrm{ATOL},$$

whereby in this case $\mathrm{RTOL} = \mathrm{ATOL} =$ TOL_v is determined by the user.
With the help of a proportional-integral (PI) controller the time step size is computed by

$$\Delta t_{k+1} = \begin{cases} \Delta t_k \left(\dfrac{\theta\, \mathrm{TOL}}{\|\hat{\mathbf{e}}_{k+1}\|} \right)^{\beta_I/(\hat{p}+1)} \left(\dfrac{\|\hat{\mathbf{e}}_k\|}{\|\hat{\mathbf{e}}_{k+1}\|} \right)^{\beta_P/(\hat{p}+1)}, & \dfrac{\|\hat{\mathbf{e}}_{k+1}\|}{\mathrm{TOL}} \le 1.2 \\ \Delta t_k \left(\dfrac{\theta\, \mathrm{TOL}}{\|\hat{\mathbf{e}}_{k+1}\|} \right)^{1/(\hat{p}+1)}, & \dfrac{\|\hat{\mathbf{e}}_{k+1}\|}{\mathrm{TOL}} > 1.2 \end{cases}.$$

The local error estimator $\|\hat{\mathbf{e}}_{k+1}\| = \|\mathbf{U}^{k+1} - \hat{\mathbf{U}}^{k+1}\|$ is computed by using an embedded Runge-Kutta method where two results of different order
are compared. Due to the use of the SDIRK2 method (2nd order) the result $\hat{\mathbf{U}}^{k+1}$ is based on a method of order $\hat{p} = 1$.
The other parameters are

$$\theta = 0.8, \qquad \beta_I = 0.3, \qquad \beta_P = 0.4.$$

Remark: For solving the ODE $\dot{\mathbf{U}} = \mathbf{F}(t, \mathbf{U})$, the Singly Diagonally Implicit Runge Kutta (SDIRK2) method

$$\boldsymbol{\eta}_1 = \mathbf{U}^k + \Delta t\, \alpha\, \mathbf{F}\big(t^k + \alpha\, \Delta t,\, \boldsymbol{\eta}_1\big),$$

$$\mathbf{U}^{k+1} = \mathbf{U}^k + \Delta t \Big((1-\alpha)\, \mathbf{F}\big(t^k + \alpha\, \Delta t,\, \boldsymbol{\eta}_1\big) + \alpha\, \mathbf{F}\big(t^k + \Delta t,\, \mathbf{U}^{k+1}\big) \Big),$$

$$\alpha = 1 - \frac{\sqrt{2}}{2}$$

is used, which is of second order accuracy. That is why it is abbreviated as SDIRK2 .
See time_integration_impl and time_integration_impl_solve_v .
See TOL_T and TOL_keps .

3.2.269. TRANSPORT_ODE_fct_evaluation

(experimental) switch for additional function evaluation within the implicit time integration scheme in EULERIMPL setting

TRANSPORT_ODE_fct_evaluation = 'YES'

Default: TRANSPORT_ODE_fct_evaluation = 'NON'

If DIRK(Diagonally Implicit Runge-Kutta) methods of the form

$$\boldsymbol{\eta}_j = \mathbf{U}^k + \Delta t \sum_{\nu=1}^{j} a_{j\nu}\, \mathbf{L}_\nu, \qquad j = 1, \ldots, s,$$

$$\mathbf{U}^{k+1} = \mathbf{U}^k + \Delta t \sum_{j=1}^{s} b_j\, \mathbf{L}_j$$

$$\mathbf{L}_j = \mathbf{F}\left(t^k + c_j\, \Delta t,\, \boldsymbol{\eta}_j\right)$$

are used for solving the ODE $\dot{\mathbf{U}} = \mathbf{F}(t, \mathbf{U})$, each stage $\boldsymbol{\eta}_j$ can be calculated one by one using the previous stages $\boldsymbol{\eta}_\nu, \nu = 1, \ldots j-1$.
This requires the function values $\mathbf{L}_j$ that are not yet directly available after the solution of the equation system for $\boldsymbol{\eta}_j$.
Either one can evaluate the discretization function $\mathbf{F}$ at $\boldsymbol{\eta}_j$ (TRANSPORT_ODE_fct_evaluation = 'YES'), what could be quite expensive, or after solving the
equation system one can use the relation (TRANSPORT_ODE_fct_evaluation = 'NON')

$$\mathbf{L}_j = \frac{1}{a_{jj}\,\Delta t}\left(\boldsymbol{\eta}_j - \mathbf{U}^k - \Delta t \sum_{\nu=1}^{j-1} a_{j\nu}\, \mathbf{L}_\nu\right), \qquad j = 1, \ldots, s.$$

Due to the linearization of the discretization function both approaches are not equivalent. In some cases the additional evaluation
(TRANSPORT_ODE_fct_evaluation = 'YES') can lead to more accurate results, but especially when using larger time steps it can become very unstable.
It has been observed that using TRANSPORT_ODE_fct_evaluation = 'YES' brings no advantage in solving the velocity and k-epsilon model.
Therefore it is only implemented for the temperature. Thus TRANSPORT_ODE_fct_evaluation = 'YES' only influences the temperature,
but for stability reasons it is recommended to use TRANSPORT_ODE_fct_evaluation = 'NON'.

Remark: For solving the ODE $\dot{\mathbf{U}} = \mathbf{F}(t, \mathbf{U})$, the Singly Diagonally Implicit Runge-Kutta(SDIRK) method

$$\boldsymbol{\eta}_1 = \mathbf{U}^k + \Delta t\, \alpha\, \mathbf{F}\left(t^k + \alpha\, \Delta t,\, \boldsymbol{\eta}_1\right),$$

$$\mathbf{U}^{k+1} = \boldsymbol{\eta}_2 = \mathbf{U}^k + \Delta t \left((1-\alpha)\, \mathbf{F}\left(t^k + \alpha\, \Delta t,\, \boldsymbol{\eta}_1\right) + \alpha\, \mathbf{F}\left(t^k + \Delta t,\, \boldsymbol{\eta}_2\right)\right),$$

$$\alpha = 1 - \frac{\sqrt{2}}{2}$$

is used, which is of second order accuracy. That's why it is abbreviated as SDIRK2.

3.2.270. USER_curve_interpol_cache

turn on caching in USER_curve_interpol_3

```
USER_curve_interpol_cache = 1
```

Default: USER_curve_interpol_cache = 0 (i.e. no caching used)
When applying MOVE statements often the same curve parameters are repeatedly evaluated.
Use this flag to turn on caching on the first level for this.
Note: This only works when MOVE statements are only time-dependent!

3.2.271. UseBoxSystemVersion

force MESHFREE to use a certain tree algorithm for the MESHFREE point neighbor search

```
UseBoxSystemVersion = 2
```

Default: 2
UseBoxSystemVersion = 0 : The original method was the box search, were the whole flow domain was covered with regular
boxes. Different box systems of different box sizes (edge length) were established in order
to pay attention to locally varying smoothing length.
UseBoxSystemVersion = 1 : Same as Version 0, but using a list-tree-algorithm in order to be more efficient.
UseBoxSystemVersion = 2 : bintree-decomposition of the pointcloud, that avoids different boxsystems. The bisectors of the
bintree form a natural sequence of cells adaptive to the smoothing length.

3.2.272. V00_SmoothDivV

Chorin projection: smooth the local values of div(v) before going into the correction pressure computation (CV)

See V00_SmoothDivV . Definitions in USER_common_variables are dominant.

3.2.273. VOLUME_correction_FreeSurface

(chamberwise) parameter to correct volume by tiny global lifting of the free surface (CV)

See VOLUME_correction_FreeSurface . Definitions in USER_common_variables are dominant.

3.2.274. VOLUME_correction_ResetOnRestart

(experimental) resets the volume correction quantities of each chamber to the current values

ONLY FOR TESTING AND DEBUGGING.

3.2.275. VOLUME_correction

(chamberwise) parameter to correct volume by GLOBALLY adjusting the divergence of velocity term (CV)

See VOLUME_correction . Definitions in USER_common_variables are dominant.

3.2.276. VOLUME_correction_local

(chamberwise) parameter to correct volume by LOCALLY adjusting the divergence of velocity term due to representative mass balance (CV)

See VOLUME_correction_local . Definitions in USER_common_variables are dominant.

3.2.277. VP0_VelocityCorrection

invoke velocity correction based on correction pressure (%ind_c%) for vp- solver (CV)

See VP0_VelocityCorrection . Definitions in USER_common_variables are dominant.

3.2.278. WARNINGS_BND_Integrate

flag controlling the warnings in BND_Integrate

Failed checks regarding the aliases given in a ConstructClause produce warnings for each boundary element. They can be disabled by

```
WARNINGS_BND_Integrate = 0
```

Possible options:
WARNINGS_BND_Integrate = 0 # no warnings are triggered
WARNINGS_BND_Integrate = 1 # all warnings are triggered (default)

3.2.279. WARNINGS_USER_parse_IsConditionStringFulfilledByBE

flag controlling the warnings in USER_parse_IsConditionStringFulfilledByBE

Failed checks regarding the alias parsing (complete string) produce warnings for each boundary element. They can be disabled by

```
WARNINGS_USER_parse_IsConditionStringFulfilledByBE = 0
```

Possible options:
WARNINGS_USER_parse_IsConditionStringFulfilledByBE = 0 # no warnings are triggered
WARNINGS_USER_parse_IsConditionStringFulfilledByBE = 1 # all warnings are triggered (default)

3.2.280. WARNINGS_USER_parse_IsConditionSubstringFulfilledByBE

flag controlling the warnings in USER_parse_IsConditionSubstringFulfilledByBE

Failed checks regarding the alias parsing (substring) produce warnings for each boundary element. They can be disabled by

```
WARNINGS_USER_parse_IsConditionSubstringFulfilledByBE = 0
```

Possible options:
WARNINGS_USER_parse_IsConditionSubstringFulfilledByBE = 0 # no warnings are triggered
WARNINGS_USER_parse_IsConditionSubstringFulfilledByBE = 1 # all warnings are triggered (default)

3.2.282. WallLayer

Turbulent wall layer thickness

Simulation wide preset for wall layer thickness in %BND_wall% for BC_k .
Default: 0.1

3.2.284. additionalPoint_approximation

(experimental) in EULERIMPL and EULEREXPL setting

```
additionalPoint_approximation = 1
```

Default: additionalPoint_approximation = 2

This is used in the EULERIMPL and EULEREXPL setting.
There are two options to approximate the unknown function values $\tilde{u}_i, \tilde{u}_j$ of the MUSCL reconstructions (see pure_TRANSPORT)

$$u_{ij}^{+} = u_i + \frac{1}{2}\,\phi\left(r_{ij}^{+}\right)(u_i - \tilde{u}_i),$$

$$u_{ij}^{-} = u_j - \frac{1}{2}\,\phi\left(r_{ij}^{-}\right)(\tilde{u}_j - u_j),$$

whereby ϕ is the slope LIMITER and

$$r_{ij}^{+} = \begin{cases} \dfrac{u_j - u_i}{u_i - \tilde{u}_i}, & u_i \neq \tilde{u}_i \\ 0\,, & \text{sonst} \end{cases}, \qquad r_{ij}^{-} = \begin{cases} \dfrac{u_j - u_i}{\tilde{u}_j - u_j}, & \tilde{u}_j \neq u_j \\ 0\,, & \text{sonst} \end{cases}.$$

- additionalPoint_approximation = 1 -> function values $\tilde{u}_i, \tilde{u}_j$ are approximated by FPM stencil, i.e. $\tilde{\mathbf{c}}_j^{(0)} = \left(\tilde{c}_{j1}^{(0)}, \ldots, \tilde{c}_{j\tilde{N}_j}^{(0)}\right)^T$:

$$u(\tilde{\mathbf{x}}_j) \approx \tilde{u}_j = \sum_{k=1}^{\tilde{N}_j} \tilde{c}_{jk}^{(0)} u(\mathbf{x}_k)$$

On the hand this approach is very accurate, but it is very expensive and unsuitable for MPI parallelization because of the neighborhood extension.
Therefore it is only for experimental purposes!

- additionalPoint_approximation = 2 -> function values $\tilde{u}_i, \tilde{u}_j$ are approximated by Taylor Expansions of second order:

$$u(\tilde{\mathbf{x}}_i) \approx \tilde{u}_i = u_j - 2\, d\mathbf{x}_j^T \cdot \nabla u_i, \qquad u(\tilde{\mathbf{x}}_j) \approx \tilde{u}_j = u_i + 2\, d\mathbf{x}_j^T \cdot \nabla u_j,$$

whereby the gradients are approximated by using the FPM stencils for approximating x,y,z-derivative $c_{ij}^x, c_{ij}^y, c_{ij}^z$. Thus

$$\nabla u_i = \sum_{\substack{j \in S(i) \\ j \neq i}} \begin{pmatrix} c_{ij}^x \\ c_{ij}^y \\ c_{ij}^z \end{pmatrix} (u_j - u_i), \qquad \nabla u_j = \sum_{\substack{k \in S(j) \\ k \neq j}} \begin{pmatrix} c_{jk}^x \\ c_{jk}^y \\ c_{jk}^z \end{pmatrix} (u_k - u_j).$$

This approach is very fast and only slightly less accurate than approach 1.
Thus this is the method of choice!

Remark:
The additionalPoint_approximation parameter is intended for experimental purposes only! Better do not touch!

3.2.286. compute_FS

(chamberwise) switch to compute free surfaces (CV)

See compute_FS . Definitions in USER_common_variables are dominant.

3.2.287. compute_LAYER

(experimental) influence to Neighbor Filtering over Layers

see LAYER .

3.2.288. compute_phase_boundary

(obsolete) invoke detection of interface connections (CV)

Obsolete, use PHASE_distinction instead.

3.2.291. damping_p_corr

(chamberwise) parameter to reduce the dynamic pressure as initial guess for the next time level (CV)

See damping_p_corr . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · delaunay_reduction

3.2.292. delaunay_reduction

switch for delaunay reduction procedure

```
delaunay_reduction = 0
```

Default: delaunay_reduction = 0 (off)
performs a reduction of the local delaunay tetrahedrization: If a boundary element triangle intersects with a local delaunay tetrahedrization, the corresponding tet is going to deleted. Advise: This option should only be activated when necessary because it is a very expensive operation

MESHFREE · InputFiles · common_variables · dist_FS_from_BND

3.2.295. dist_FS_from_BND

define hole size for the free surface detection

```
dist_FS_from_BND = 0.52
```

Default: dist_FS_from_BND = 0.65

See especially ORGANIZE_CheckFreeSurface_Version .

Sorry for the slightly missleading name

MESHFREE · InputFiles · common_variables · dist_LayerThickness

3.2.297. dist_LayerThickness

minimal thickness for degenerated 3D phase

```
dist_LayerThickness = (0.02, OPTIONAL: 1)
```

Default: dist_LayerThickness = (-1, -1) # not switched on

Wherever a 3D fluid phase degenerates to only one layer of free surface points above one layer of active or inactive boundary points, without any interior points in between, the free surface points are kept at a distance (smoothing length)*(dist_LayerThickness) in normal direction from the boundary.

Note:

- Currently, choosing this parameter > 0 is mandatory to model degenerated thin films in a full 3D approach.
- Since all free surface points are kept at the prescribed distance (independent of the activation status of the subjacent boundary points), artifacts in the form of apparently "hovering" points may occur.
  optional second value: if >0, it will switch on the adaptive dist_LayerThickness , based on the representative masses of the free surface points, i.e.
  $d_i = \frac{1}{h_i} \frac{\hat{m}}{A_i \cdot \rho_i}$

3.2.298. dist_aip

initial relative distance to boundary of a newly injected MESHFREE point (aip = add injected points)

Injection is actively performed at those boundaries who carry the identifier IDENT%BND_inflow%

```
dist_aip = 0.16 # newly injected points have a distance from the injecting boundary of abs( dist_aip )*SMOOTH_LENGTH
```

newly injected points have a distance from the injecting boundary of abs(dist_aip)*SMOOTH_LENGTH
ATTENTION: if dist_aip < dist_rab , then dist_aip is corrected to ind_aip=dist_rab+0.01

OPTION:

```
dist_aip = -0.02 # newly injected points have a distance from the injecting boundary of abs( dist_aip )*SMOOTH_LENGTH
```

By putting a minus in front, we force this injection distance regardless of dist_rab . In this case, removal of points in the %BND_inflow% regions is prevented.

Attention: in case of abs(dist_aip) <= 0.05, the newly injected point will not be interpolated for initialization but will obtain the values of the injecting MESHFREE point.

3.2.301. dist_rab

relative allowed minimum distance of MESHFREE points to boundary (rab = remove at boundary)

MESHFREE points are removed if they come too close to the regular boundary.
The results of the distance to boundary check are retrievable (in an a-posteriori sense) from the variables %ind_dtb% , %ind_OrganizeDTB% .

3.2.302. dist_rip

relative allowed minimum distance between MESHFREE points (rip = remove interior points)

If two points become closer to each other than (dist_rip * H), then they will be clustered.
The state of clustering is stored in %ind_OrganizePC(2)% for interior points and in %ind_OrganizePC(4)% for boundary points.

3.2.303. eps_T

precision in the breaking criterion for the linear systems of temperature (CV)

See eps_T . Definitions in USER_common_variables are dominant.

3.2.304. eps_p

precision in the breaking criterion for the linear systems of pressure (CV)

See eps_p . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · eps_phyd

3.2.305. eps_phyd

precision in the breaking criterion for the linear systems of hydrostatic pressure (CV)

See eps_phyd . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · eps_v

3.2.306. eps_v

precision in the breaking criterion for the linear systems of velocity (CV)

See eps_v . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · iFPM_process_ID

3.2.307. iFPM_process_ID

give a maximum 16-digit MESHFREE process ID

```
iFPM_process_ID = 0123456789123456
```

Default: if this variable is not given, then MESHFREE assigns the process ID as to be the computers clock time in seconds.
All signal and log-files carry the MESHFREE process ID inside of their name.
Hence, setting the iFPM_process_ID will avoid long lists of signal and log-files if not desired.

Changing the process ID during run-time (by ComputationalSteering) will have no effect

MESHFREE · InputFiles · common_variables · int_BND_part_add

3.2.309. int_BND_part_add

boundary point addition interval

Defines after how many time steps boundary point addition will be done
Default: int_BND_part_add=3

MESHFREE · InputFiles · common_variables · int_BND_part_remove

3.2.310. int_BND_part_remove

boundary point removal interval

Defines after how many time steps boundary point removal will be done
Default: int_BND_part_remove=3

MESHFREE · InputFiles · common_variables · int_part_add

3.2.311. int_part_add

interior point addition interval

Defines after how many time steps interior point addition will be done
Default: int_part_add=3

MESHFREE · InputFiles · common_variables · int_part_remove

3.2.313. int_part_remove

interior point removal interval

Defines after how many time steps interior point removal will be done
Default: int_part_remove=3

MESHFREE · InputFiles · common_variables · max_N_stencil_INTERIOR

3.2.316. max_N_stencil_INTERIOR

max. number of neighbors accepted for stencil computation and numerics only for interior pooints

```
max_N_stencil_INTERIOR = 25
```

Default: max_N_stencil_INTERIOR = 40
Defines the maximum number of accepted neighbor points for the pure numerics (stencil computation, differential operators).
Only interior points are concerned.
Out of the complete neighbor list, MESHFREE selects the max_N_stencil_INTERIOR closest ones.
Additionally, we have the constraint
max_N_stencil_INTERIOR = min(max_N_stencil, max_N_stencil_INTERIOR)

MESHFREE · InputFiles · common_variables · max_N_stencil

3.2.317. max_N_stencil

maximum number of neighbor points accepted for stencil computation and numerics (CV)

See max_N_stencil . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · ord_eval

3.2.319. ord_eval

define approximation order for refill points (CV)

See ord_eval . Definitions in USER_common_variables are dominant.

3.2.320. ord_gradient

(chamberwise) approximation order of the gradient operators (CV)

See ord_gradient . Definitions in USER_common_variables are dominant.

3.2.321. ord_laplace

define approximation order of the Laplace operators (CV)

See ord_laplace . Definitions in USER_common_variables are dominant.

3.2.322. pBubble_Offset

define offset pressure for bubble pressure-on-volume analysis

Deprecated version of BUBBLE_pOffset , see there.

3.2.323. prec_seek_holes

number of test points created for hole search

If prec_seek_holes = 0, the local Delauney tetrahedrization will be used to seek holes in the point cloud
If prec_seek_holes > 0, the hole search will be done independent of the local Delauney tetrahedrization and the value of prec_seek_holes defines the number of test points created

Default: prec_seek_holes = 0

Note: In 3D, if values larger than 10 are specified, they will be reduced to this upper bound.

3.2.324. pure_TRANSPORT

(experimental) choice of spatial discretization scheme for transport terms in EULERIMPL and EULEREXPL setting

```
pure_TRANSPORT = 2
```

Default: pure_TRANSPORT = 1

There are two different methods for discretizing transport terms within the EULERIMPL and EULEREXPL setting:

1.) Cutting method for efficient solving of transport terms $\mathbf{v} \cdot \nabla u$. Designed for incompressible solver LIQUID .

2.) Rotational method for approximating flux functions in hyperbolic equations $\partial_t u + \nabla \cdot \mathbf{Q}(u) = 0$. Designed for compressible solver.

- pure_TRANSPORT = 1 -> transport terms $\mathbf{v} \cdot \nabla u$ are discretized by cutting method:
  For the transport equation $\partial_t u + \mathbf{v} \cdot \nabla u = 0$ the discretization scheme for an interior point $\mathbf{x}_i$ is

$$\frac{du_i}{dt} = -2 \sum_{\substack{j \in S(i) \\ j \neq i}} c_{ij}^{\mathbf{v}\cdot\nabla} \left(u_{ij} - u_i\right),$$

$$u_{ij} = \frac{1}{2}\Big[\left(1 + \text{sign}((\mathbf{x}_j - \mathbf{x}_i)\cdot \mathbf{v}_i)\right) u_{ij}^{+} + \left(1 - \text{sign}((\mathbf{x}_j - \mathbf{x}_i)\cdot \mathbf{v}_i)\right) u_{ij}^{-}\Big],$$

$$c_{ij}^{\mathbf{v}\cdot\nabla} = \mathbf{c}_{ij} \cdot \mathbf{v}_i = c_{ij}^x \, v_i^x + c_{ij}^y \, v_i^y + c_{ij}^z \, v_i^z .$$

$c_{ij}^x, c_{ij}^y, c_{ij}^z$: FPM stencils for approximating x,y,z-derivative

$$\text{sign}(x) = \begin{cases} -1, & x \leq 0 \\ 1, & x > 0 \end{cases}$$

The MUSCL reconstructions are

$$u_{ij}^{+} = u_i + \frac{1}{2}\,\phi\left(r_{ij}^{+}\right)(u_i - \tilde{u}_i),$$

$$u_{ij}^{-} = u_j - \frac{1}{2}\,\phi\left(r_{ij}^{-}\right)(\tilde{u}_j - u_j),$$

whereby ϕ is the slope LIMITER and

$$r_{ij}^{+} = \begin{cases} \dfrac{u_j - u_i}{u_i - \tilde{u}_i}, & u_i \neq \tilde{u}_i \\ \\ 0\,, & \text{sonst} \end{cases}, \qquad r_{ij}^{-} = \begin{cases} \dfrac{u_j - u_i}{\tilde{u}_j - u_j}, & \tilde{u}_j \neq u_j \\ \\ 0\,, & \text{sonst} \end{cases}$$

The unknown function values $\tilde{u}_i, \tilde{u}_j$ are approximated by Taylor expansions (see additionalPoint_approximation).

- pure_TRANSPORT = 2 -> transport terms $\mathbf{v} \cdot \nabla u$ are discretized by rotational method. Only for experimental purposes!
- pure_TRANSPORT = 3 -> flux function $\mathbf{Q}(u)$ is approximated by cutting method. This works only for scalar hyperbolic equations!
  Only for experimental purposes!
- pure_TRANSPORT = 4 -> flux function $\mathbf{Q}(u)$ is approximated by rotational method. This method is only implemented for scalar hyperbolic equations in the EULEREXPL setting so far. But it is also being implemented for hyperbolic systems like shallow water or gas dynamic equations.
  For the scalar equation $\partial_t u + \nabla \cdot \mathbf{Q}(u) = 0$ the discretization scheme for an interior point $\mathbf{x}_i$ is

$$\frac{du_i}{dt} = -2 \sum_{\substack{j \in S(i) \\ j \neq i}} \tilde{c}_{ij}^x \left(F(u_{ij}^{+}, u_{ij}^{-}, \hat{\mathbf{n}}_{ij}) - \mathbf{Q}(u_i)\cdot \hat{\mathbf{n}}_{ij}\right) + \tilde{c}_{ij}^y \left(G(u_{ij}^{+}, u_{ij}^{-}, \hat{\mathbf{s}}_{ij}) - \mathbf{Q}(u_i)\cdot \hat{\mathbf{s}}_{ij}\right) + \tilde{c}_{ij}^z \left(H(u_{ij}^{+}, u_{ij}^{-}, \hat{\mathbf{t}}_{ij}) - \mathbf{Q}(u_i)\cdot \hat{\mathbf{t}}_{ij}\right),$$

$$F(u_{ij}^{+}, u_{ij}^{-}, \hat{\mathbf{n}}_{ij}) = \frac{1}{2}\big(\mathbf{Q}(u_{ij}^{+}) + \mathbf{Q}(u_{ij}^{-})\big)\cdot \hat{\mathbf{n}}_{ij} - \frac{1}{2}\big|\tilde{\mathbf{a}}_{ij}\cdot \hat{\mathbf{n}}_{ij}\big|\big(u_{ij}^{-} - u_{ij}^{+}\big),$$

$$G(u_{ij}^{+}, u_{ij}^{-}, \hat{\mathbf{s}}_{ij}) = \frac{1}{2}\big(\mathbf{Q}(u_{ij}^{+}) + \mathbf{Q}(u_{ij}^{-})\big)\cdot \hat{\mathbf{s}}_{ij},$$

$$H(u_{ij}^{+}, u_{ij}^{-}, \hat{\mathbf{t}}_{ij}) = \frac{1}{2}\big(\mathbf{Q}(u_{ij}^{+}) + \mathbf{Q}(u_{ij}^{-})\big)\cdot \hat{\mathbf{t}}_{ij}$$

$$\tilde{\mathbf{a}}_{ij} = \begin{cases} \dfrac{\mathbf{Q}(u_{ij}^-) - \mathbf{Q}(u_{ij}^+)}{u_{ij}^- - u_{ij}^+}, & u_{ij}^- \neq u_{ij}^+ \\ \dfrac{d}{du}\mathbf{Q}(u_{ij}^+), & u_{ij}^- = u_{ij}^+ \end{cases}$$

whereby

$$\tilde{c}_{ij}^x = c_{ij}^x \sin\theta_{ij}^z \cos\theta_{ij}^x + c_{ij}^y \sin\theta_{ij}^z \sin\theta_{ij}^x + c_{ij}^z \cos\theta_{ij}^z$$
$$\tilde{c}_{ij}^y = c_{ij}^x \cos\theta_{ij}^z \cos\theta_{ij}^x + c_{ij}^y \cos\theta_{ij}^z \sin\theta_{ij}^x - c_{ij}^z \sin\theta_{ij}^z$$
$$\tilde{c}_{ij}^z = -c_{ij}^x \sin\theta_{ij}^x + c_{ij}^y \cos\theta_{ij}^x$$

are the stencils in the rotated coordinate system

$$\hat{\mathbf{n}}_{ij} = \begin{pmatrix} \sin\theta_{ij}^z \cos\theta_{ij}^x \\ \sin\theta_{ij}^z \sin\theta_{ij}^x \\ \cos\theta_{ij}^z \end{pmatrix}, \qquad \hat{\mathbf{s}}_{ij} = \begin{pmatrix} \cos\theta_{ij}^z \cos\theta_{ij}^x \\ \cos\theta_{ij}^z \sin\theta_{ij}^x \\ -\sin\theta_{ij}^z \end{pmatrix}, \qquad \hat{\mathbf{t}}_{ij} = \begin{pmatrix} -\sin\theta_{ij}^x \\ \cos\theta_{ij}^x \\ 0 \end{pmatrix}.$$

$\theta_{ij}^z \in [0, \pi]$ is the angle between z-axis and vector $\mathbf{x}_j - \mathbf{x}_i$
$\theta_{ij}^x \in [0, 2\pi)$ is the angle between x-axis and vector $(x_j - x_i, y_j - y_i, 0)^T$

Remark:
The pure_TRANSPORT parameter is intended for experimental purposes only! Better do not touch!
At the moment only pure_TRANSPORT = 1 is used in the EULERIMPL setting because it is specially developed for the LIQUID solver.
pure_TRANSPORT = 2-4 only works in the EULEREXPL setting because these spatial discretization schemes are not linearized yet for implicit solving.

3.2.325. radius_hole

relative allowed hole size (CV)

See radius_hole . Definitions in USER_common_variables are dominant.

3.2.326. rel_dist_bound

relative distance of neighboring points at boundaries (CV)

See rel_dist_bound . Definitions in USER_common_variables are dominant.

3.2.327. rel_dist_edge

relative distance of neighboring points at edges of the geometry

```
rel_dist_edge = 0.24
```

Default: rel_dist_edge = 0.15

3.2.328. restartnewBE_filling

(chamberwise) parameter to control filling of new boundary elements upon restart (CV)

See restartnewBE_filling . Definitions in USER_common_variables are dominant.

MESHFREE · InputFiles · common_variables · time_integration_expl

3.2.334. time_integration_expl

order of explicit time integration scheme in EULEREXPL setting

```
time_integration_expl = 1
```

Default: time_integration_expl = 2

- time_integration_expl = 1 uses for time integration the explicit Euler method, which is of first order accuracy.
- time_integration_expl = 2 uses for time integration the explicit Heun method, which is of second order accuracy.

This influences only the approximation of the transport terms in EULEREXPL setting.

MESHFREE · InputFiles · common_variables · time_integration_impl

3.2.335. time_integration_impl

order of implicit time integration scheme in EULERIMPL setting

```
time_integration_impl = 1
```

Default: time_integration_impl = 2

- time_integration_impl = 1 uses for time integration the implicit Euler method, which is of first order accuracy.
- time_integration_impl = 2 uses for time integration the implicit SDIRK2 (Singly Diagonally Implicit Runge-Kutta) method, which is of second order accuracy.

SDIRK2 method is only used if at least one phase (see KindOfProblem) is calculated with EULERIMPL .
Using LAGRANGE it is not worth to use second order time integration because it does not improve the accuracy of the results. Due to the movement
of the points in the LAGRANGE setting only viscous parts are implicitly solved. Thus the time discretization error plays a minor part. Therefore
time_integration_impl is automatically set to 1 in LAGRANGE phase.

Remark: For solving the ODE $\dot{\mathbf{U}} = \mathbf{F}(t, \mathbf{U})$, the SDIRK2 method is

$$\boldsymbol{\eta}_1 = \mathbf{U}^k + \Delta t\, \alpha\, \mathbf{F}\big(t^k + \alpha\, \Delta t,\, \boldsymbol{\eta}_1\big),$$

$$\mathbf{U}^{k+1} = \mathbf{U}^k + \Delta t \left((1-\alpha)\, \mathbf{F}\big(t^k + \alpha\, \Delta t,\, \boldsymbol{\eta}_1\big) + \alpha\, \mathbf{F}\big(t^k + \Delta t,\, \mathbf{U}^{k+1}\big) \right).$$

$$\alpha = 1 - \frac{\sqrt{2}}{2}$$

The time step size is controlled by the error tolerances TOL_T (temperature), TOL_v (velocity) and TOL_keps (k-epsilon model).

3.2.336. time_integration_impl_solve_v

order of implicit time integration scheme for velocity only (EULERIMPL)

```
time_integration_impl_solve_v = 1
```

Default: time_integration_impl_solve_v = 2

- time_integration_impl_solve_v = 1 uses for time integration of velocity the implicit Euler method, which is of first order accuracy.
- time_integration_impl_solve_v = 2 uses for time integration of velocity the implicit SDIRK2 (Singly Diagonally Implicit Runge-Kutta) method,
  which is of second order accuracy.

SDIRK2 method is only used if at least one phase (see KindOfProblem) is calculated with EULERIMPL .
If in the case of EULERIMPL the BND_free condition is used for the velocity, then there may be problems with the second order time integration (SDIRK2).
In this case the parameter time_integration_impl_solve_v can be used to solve the velocity with the implicit Euler time integration (first order).

See also time_integration_impl .

The time step size is controlled by the error tolerance TOL_v for the velocity.

3.2.337. time_step_gain

relative amount by which new timestep size can increase at maximum compared to old timestep size

```
time_step_gain = 0.7
```

Default: time_step_gain = 1.0
Parameter to control the maximum increase of time step size when using DELT_dt_variable = 1. The next time step size can at maximum increase to (1 + time_step_gain) times the current time step size.
see TimeControl .

3.2.338. time_step_loss

relative amount by which new timestep size can decrease at maximum compared to old timestep size (adaptive timestep size)

```
time_step_loss = 0.7
```

Default: time_step_loss = 0.5
Parameter to control the maximum decrease of time step size when using DELT_dt_variable = 1. The next time step size can at decrease at maximum to (1-time_step_loss) times the current time step size.
see TimeControl .

3.2.340. turn_down_BND_order

(chamberwise) parameter to automatically reduce the approximation order of a boundary point

```
turn_down_BND_order = 0.5
```

Default: turn_down_BND_order = 0.25
The approximation order of a boundary point (including free surfaces!) is automatically reduced by one,
if the ratio between number of interior and boundary points in the stencil fulfills

$$\frac{N_{interior}}{N_{boundary}} <= TurnDownBNDorder$$

Note: This parameter can also be set chamberwise for multiphase simulations (see also KindOfProblem , CHAMBER).
If it is not set for specific chambers, it is automatically set according to the non-chamberwise definition for all chambers.

MESHFREE · InputFiles · common_variables · use_BubbleManagement

3.2.341. use_BubbleManagement

(chamberwise) switch regarding bubble analysis

Deprecated version of BUBBLE_DoTheManagement , see there.

MESHFREE · Indices

4. Indices

MESHFREE indices for simulation entities

Indices for MESHFREE -variables have the form %ind_NameOfVariable% and they are used to refer to internally stored quantities:

- physical quantities: hydrostatic pressure at point %ind_p%
- geometrical quantities: distance of point to boundary %ind_dtb%
- organizational quantities: type of boundary (wall, inner, free surface) %ind_kob%

The indices can be used in equations to directly refer to the quantities on the pointcloud by

```
[... Y%ind_NameOfVariable% ...]
```

There are General indices, that are available in all chambers, chamber specific indices that are only available for specific simulation chambers, e.g. LIQUID and
UserDefinedIndices giving the user freedom to define own indices.

See also __Constants__ .

List of members:	
General	MESHFREE indices for general simulation entities
LIQUID	indices for the implicit (incompressible/weakly compressible) solver
TRANSPORT	MESHFREE indices for TRANSPORT, i.e. solving hyperbolic problems
MANIFOLD	indices for the manifold phase
SHALLOWWATER	Indices for the shallow water solver
GASDYN	Indices for the explicit (gasdynamics) solver
POPBAL	Indices for the population balance solver
DROPLETPHASE	Indices for the droplet and particle phase solver
UserDefinedIndices	user defined indices

4.1. DROPLETPHASE

Indices for the droplet and particle phase solver

List of members:	
%ind_betaDarcy%	
%ind_d30%	diameter of droplet [m]
%ind_diss%	DROPLETPHASE: dissipated energy of particle by interaction with other particles or wall
%ind_diss_BE%	DROPLETPHASE: dissipated energy at the BE by collision with the particle
%ind_ETA%	viscosity of droplet material [Pa*s], in case of water film it is etaNormal
%ind_ETA_eff%	viscosity of droplet material [Pa*s], in case of water film it is etaTangential
%ind_g(1)%	x-component of gravity [m/s^2]
%ind_g(2)%	y-component of gravity [m/s^2]
%ind_g(3)%	z-component of gravity [m/s^2]
%ind_g_eff(1)%	if droplets collect at the boundary as a film: effective gravity due to centrifugal acceleration, x-component [m/s^2]
%ind_g_eff(2)%	if droplets collect at the boundary as a film: effective gravity due to centrifugal acceleration, y-component [m/s^2]
%ind_g_eff(3)%	if droplets collect at the boundary as a film: effective gravity due to centrifugal acceleration, z-component [m/s^2]
%ind_grad_hwf(1)%	if droplets collect on a boundary: gradient of height of accumulated water film, x-component [1]

%ind_grad_hwf(2)%	if droplets collect on a boundary: gradient of height of accumulated water film, y-component [1]
%ind_grad_hwf(3)%	if droplets collect on a boundary: gradient of height of accumulated water film, z-component [1]
%ind_gradP_uw(1)%	if droplets collect at the boundary as a film: gradient of numerical (approximated) pressure, x-component [Pa/m]
%ind_gradP_uw(2)%	if droplets collect at the boundary as a film: gradient of numerical (approximated) pressure, y-component [Pa/m]
%ind_gradP_uw(3)%	if droplets collect at the boundary as a film: gradient of numerical (approximated) pressure, z-component [Pa/m]
%ind_ground_hwf%	if droplets collect on a boundary: try to estimate in what distance (normalized by h) the liquid film touches ground [1]
%ind_h_factor%	reduce smoothing length (H) if too many droplets locally collect to a cluster [1], this value shall be less than 1
%ind_hwf%	if droplets collect on a boundary: height of accumulated water film [m]
%ind_hwf_3d%	if droplets collect on a boundary: max(height of accumulated water film , 0.5*Y%ind_d30%) [m]
%ind_lap_geometry%	if droplets collect on a boundary: curvature of the geometry [1/m]
%ind_lap_hwf%	if droplets collect on a boundary: Laplacian of the height in tangential direction (i.e. curvature of the accumulated water film) [1/m]
%ind_p%	if droplets collect at the boundary as a film: pressure due to height+gavity as well as due to surface tension [Pa]
%ind_p_corr%	This index is deprecated. Please use ind_p_dyn for the same functionality.
%ind_p_dyn%	if droplets collect at the boundary as a film: pressure due to centrifugal acceleration [Pa]
%ind_p_uw%	if droplets collect at the boundary as a film: numerical (approximated) pressure computed from %ind_p% [Pa]
%ind_r%	density of the droplet material [kg/m^3]
%ind_r_cont%	
%ind_SIG%	surface tension of droplet material [N/m]
%ind_v(1)%	x-component of velocity [m/s]
%ind_v(2)%	y-component of velocity [m/s]
%ind_v(3)%	z-component of velocity [m/s]
%ind_v0Darcy(1)%	
%ind_v0Darcy(2)%	
%ind_v0Darcy(3)%	

%ind_v_0(1)%	velocity of the previous time step, x-component
%ind_v_0(2)%	velocity of the previous time step, y-component
%ind_v_0(3)%	velocity of the previous time step, z-component
%ind_v_cont(1)%	
%ind_v_cont(2)%	
%ind_v_cont(3)%	
%ind_v_dot(1)%	if droplets collect at the boundary as a film: additional acceleration due to film dynamics (surface tension, layer thickness etc.), x-component [m/s^2]
%ind_v_dot(2)%	if droplets collect at the boundary as a film: additional acceleration due to film dynamics (surface tension, layer thickness etc.), y-component [m/s^2]
%ind_v_dot(3)%	if droplets collect at the boundary as a film: additional acceleration due to film dynamics (surface tension, layer thickness etc.), z-component [m/s^2]

MESHFREE · Indices · GASDYN

4.2. GASDYN

Indices for the explicit (gasdynamics) solver

List of members:

%ind_c%	sound speed [m/s]
%ind_CG(1)%	specific heat definition [kJ/(kg*K)]
%ind_CG(2)%	specific heat definition [kJ/(kg*K^2)]
%ind_CG(3)%	specific heat definition [kJ/(kg*K^3)]
%ind_CG(4)%	specific heat definition [kJ/(kg*K^4)]
%ind_CG_dot(1)%	time change rate of specific heat
%ind_CG_dot(2)%	time change rate of specific heat
%ind_CG_dot(3)%	time change rate of specific heat
%ind_CG_dot(4)%	time change rate of specific heat
%ind_corpnt%	classify points to be at corners
%ind_CV%	not used for GASDYN
%ind_div%	numerically computed divergence of velocity
%ind_div_bar%	
%ind_divV_uw%	numerical divergence of upwind velocity
%ind_ent%	entropy [kJ/(kg*K)]

%ind_eps%	TURBULENCE: epsilon
%ind_ETA%	physical laminar viscosity of the fluid [Pa*s]
%ind_ETA_p%	artificial viscosity, used for pressure [m^2/s], only FPM2
%ind_ETA_sm%	turbulent viscosity, if turbulence model is switched on
%ind_ETA_u%	artificial viscosity, used for velocity [m^2/s], only FPM2
%ind_gradP_uw(1)%	gradient of upwind pressure, x-component
%ind_gradP_uw(2)%	gradient of upwind pressure, y-component
%ind_gradP_uw(3)%	gradient of upwind pressure, z-component
%ind_k%	TURBULENCE: k
%ind_L_uw%	local shift in order to evaluate upwind quantities
%ind_LAM%	physical laminar heat conductivity of the fluid [W/(m*K)]
%ind_Mdot_virt%	
%ind_MUE%	not used for GASDYN
%ind_p%	pressure given due to the gas law
%ind_p_uw%	upwind pressure [Pa]
%ind_PHI%	terms for second order time integration in FPM2
%ind_PI(1)%	terms for second order time integration in FPM2
%ind_PI(2)%	terms for second order time integration in FPM2
%ind_PI(3)%	terms for second order time integration in FPM2
%ind_PSI%	terms for second order time integration in FPM2
%ind_r%	density of the gas
%ind_r_check%	postprocessing density, stemming from the exact integratio of the deformation of the delaunay triangles/tetras
%ind_r_dot%	numerical time change rate of density
%ind_r_dot1%	time change rate of density of previous time step n-1
%ind_r_sm%	smoothed density (not used anymore in GASDYN)
%ind_rE%	total energy of the gas
%ind_rE_dot%	numerical time change rate of total energy
%ind_rE_dot1%	time change rate of total energy of previous time step n-1
%ind_RG%	gas constant for equation of state (perfect gas law) [kJ/(kg*K)]
%ind_RG_dot%	time change rate of gas constant
%ind_rv(1)%	momentum of the gas, x-component

%ind_rv(2)%	momentum of the gas, y-component
%ind_rv(3)%	momentum of the gas, z-component
%ind_rv_dot(1)%	numerical time change rate of momentum, x-component
%ind_rv_dot(2)%	numerical time change rate of momentum, y-component
%ind_rv_dot(3)%	numerical time change rate of momentum, z-component
%ind_rv_dot1(1)%	time change rate of momentum of previous time step n-1, x-component
%ind_rv_dot1(2)%	time change rate of momentum of previous time step n-1, y-componen
%ind_rv_dot1(3)%	time change rate of momentum of previous time step n-1, z-componen
%ind_SIG%	not used for GASDYN
%ind_T%	temperature [K]
%ind_tauW%	TURBULENCE: wall tension
%ind_TurbulentWallLayer%	representative thickness for turbulent boundary layer
%ind_v(1)%	x-component of velocity [m/s]
%ind_v(2)%	y-component of velocity [m/s]
%ind_v(3)%	z-component of velocity [m/s]
%ind_v_uw(1)%	upwind velocity [m/s], x-component
%ind_v_uw(2)%	upwind velocity [m/s], y-component
%ind_v_uw(3)%	upwind velocity [m/s], z-component
%ind_x_dot(1)%	change of position of point (movement velocity), x-component
%ind_x_dot(2)%	change of position of point (movement velocity), y-component
%ind_x_dot(3)%	change of position of point (movement velocity), z-component
%ind_x_dot1(1)%	time change rate of position of previous time step n-1, x-component
%ind_x_dot1(2)%	time change rate of position of previous time step n-1, y-component
%ind_x_dot1(3)%	time change rate of position of previous time step n-1, z-component
%ind_XI%	terms for second order time integration in FPM2
%ind_XI0%	original basis of artificial viscosity in FPM2

4.2.11. %ind_ETA_p%

artificial viscosity, used for pressure [m^2/s], only FPM2

see equation (3.3) in DOCUMATH_Gasdyn_O2.pdf , this values is $\eta_p \cdot \rho c^2$

4.2.13. %ind_ETA_u%

artificial viscosity, used for velocity [m^2/s], only FPM2

see equation (3.3) in DOCUMATH_Gasdyn_O2.pdf , this values is $\frac{\eta_v}{\rho}$

4.2.18. %ind_PHI%

terms for second order time integration in FPM2

see equation (3.5) in DOCUMATH_Gasdyn_O2.pdf

4.2.19. %ind_PI(1)%

terms for second order time integration in FPM2

see equation (3.27) in DOCUMATH_Gasdyn_O2.pdf

4.2.20. %ind_PI(2)%

terms for second order time integration in FPM2

see equation (3.27) in DOCUMATH_Gasdyn_O2.pdf

4.2.21. %ind_PI(3)%

terms for second order time integration in FPM2

see equation (3.27) in DOCUMATH_Gasdyn_O2.pdf

4.2.22. %ind_PSI%

terms for second order time integration in FPM2

see equation (3.43) in DOCUMATH_Gasdyn_O2.pdf

4.2.28. %ind_XI%

terms for second order time integration in FPM2

see equation (3.12) in DOCUMATH_Gasdyn_O2.pdf

4.2.29. %ind_XI0%

original basis of artificial viscosity in FPM2

represents the non-dimensional value A_i^{β} , see equation (5.5) in DOCUMATH_Gasdyn_O2.pdf

4.2.31. %ind_corpnt%

classify points to be at corners

no corner: Y%ind_corpnt%=0; corner character: Y%ind_corpnt%>0

4.2.32. %ind_div%

numerically computed divergence of velocity

based on %ind_v(1)% ... %ind_v(3)%

4.2.41. %ind_p%

pressure given due to the gas law

for example $p = \rho \cdot R \cdot T$

4.2.42. %ind_p_uw%

upwind pressure [Pa]

see equation (3.2) in DOCUMATH_Gasdyn_O2.pdf , this values is $\bar{p}$

4.2.44. %ind_rE%

total energy of the gas

rE = r*u + 0.5*r*v^2 where u is the internal ernegy

4.2.48. %ind_r_dot%

numerical time change rate of density

used for time integration of the density

$$\frac{\rho^{n+1} - \rho^n}{\Delta t} = \mathrm{TimeChangeRateDensity}$$

4.2.51. %ind_rv(1)%

momentum of the gas, x-component

same as Y %ind_v(1)% *Y %ind_r%

4.2.52. %ind_rv(2)%

momentum of the gas, y-component

same as Y %ind_v(1)% *Y %ind_r%

4.2.53. %ind_rv(3)%

momentum of the gas, z-component

same as Y %ind_v(1)% *Y %ind_r%

4.2.61. %ind_v(1)%

x-component of velocity [m/s]

same as Y %ind_rv(1)% /Y %ind_r%

4.2.62. %ind_v(2)%

y-component of velocity [m/s]

same as Y %ind_rv(2)% /Y %ind_r%

4.2.63. %ind_v(3)%

z-component of velocity [m/s]

same as Y %ind_rv(3)% /Y %ind_r%

4.2.64. %ind_v_uw(1)%

upwind velocity [m/s], x-component

see equation (3.2) in DOCUMATH_Gasdyn_O2.pdf , this values is $\bar{\mathbf{v}}$

MESHFREE · Indices · GASDYN · %ind_v_uw(2)%

4.2.65. %ind_v_uw(2)%

upwind velocity [m/s], y-component

see equation (3.2) in DOCUMATH_Gasdyn_O2.pdf , this values is $\bar{\mathbf{v}}$

MESHFREE · Indices · GASDYN · %ind_v_uw(3)%

4.2.66. %ind_v_uw(3)%

upwind velocity [m/s], z-component

see equation (3.2) in DOCUMATH_Gasdyn_O2.pdf , this values is $\bar{\mathbf{v}}$

MESHFREE · Indices · General

4.3. General

MESHFREE indices for general simulation entities

To be used by all classes of MESHFREE solver (LIQUID , GASDYN , POPBAL, PARTICLEPHASE, ...)

List of members:

%ind_act%	activation status of a boundary point
%ind_addvar%	additional variables that can be used for additional tasks (legacy code)
%ind_BC%	index of boundary condition
%ind_BE1%	BE=BoundaryElement, i.e. index of boundary element which a boundary point is placed on
%ind_bndBubble%	index of macroscopic bubbles
%ind_BNDfree_defect%	defect displacement of free surface with regards to the representative mass, clusterwise
%ind_BVA(1)%	BVA=Boundary VAlue, temporary array used for defining boundary conditions
%ind_BVA(2)%	BVA=Boundary VAlue, temporary array used for defining boundary conditions
%ind_BVA(3)%	BVA=Boundary VAlue, temporary array used for defining boundary conditions
%ind_BVA_NUS(1)%	BVA_NUS=Bounbdary VAlue for NUSselt type
%ind_BVA_NUS(2)%	BVA_NUS=Boundary VAlue for NUSselt type
%ind_BVA_NUS(3)%	BVA_NUS=Boundary VAlue for NUSselt type
%ind_cell%	cell number of the tree leaf for the tree-based neigbor search (UseBoxSystemVersion=2)
%ind_cell_Deflation%	cell number of the deflation cells (experimental status)

%ind_cham%	chamber index of point
%ind_cluster%	unique cluster index of pointcloud
%ind_ClusterSurface%	clustering of free surface or of initial regular boundary
%ind_connectBcBubble%	if bubble connected tro outflow, this item holds to BC_PASSON of the outflow boundary
%ind_create%	number/index of time step of creation of this point
%ind_dA%	area covered by a boundary point, including free surface points, unit=m^2
%ind_dbp%	dbp=distance between phases, unit=meters
%ind_debug(1)%	free variables in order to debug MESHFREE and be able to visualize
%ind_debug(2)%	free variables in order to debug MESHFREE and be able to visualize
%ind_debug(3)%	free variables in order to debug MESHFREE and be able to visualize
%ind_debug(4)%	free variables in order to debug MESHFREE and be able to visualize
%ind_debug(5)%	free variables in order to debug MESHFREE and be able to visualize
%ind_div_bar_c%	PURE POSTPROCESSING: the (divergence of velocity)^bar at the point in the numerical scheme where the correction pressure is computed
%ind_div_bar_pDyn%	PURE POSTPROCESSING: the (divergence of velocity)^bar at the point in the numerical scheme where the correction pressure is computed
%ind_dt%	global time step size used for this point, unit=seconds
%ind_dt_0%	size of the previous time step, unit=seconds
%ind_dt_local%	locally feasible time step size
%ind_dtb%	dtb=distance to boundary, unit=meters
%ind_dtb_nbTria%	how many BE-triangles found in neighborhood for computation of %ind_dtb%
%ind_dtb_status%	status of the new distance-to-boundary-computation
%ind_EdgeValue%	pointcloud configuration without interior points: this item marks points at the edge of such configurations
%ind_event_AbortFPM%	current event status for event stopping MESHFREE
%ind_event_DeletePoint%	current event status for event deleting points
%ind_event_FunctionManipulation%	current event status for function manipulation event
%ind_event_GeometricalFunctionManipulation%	current event status for geometrical function manipulation event
%ind_event_Msg%	current event status for event print message
%ind_event_SaveResults%	current event status for event saving computational results
%ind_event_StopFPM%	current event status for event stopping MESHFREE
%ind_event_WriteRestart%	current event status for event writing of a restart file

%ind_event_WriteResume%	current event status for event writing of a resume file
%ind_ForceApproximation%	marks points which are scheduled for re-approximation
%ind_h%	local smoothing length, unit=meters
%ind_h_adaptive%	local smoothing length proposed for adaptive treatment of H, unit=meters
%ind_ID%	global identifier of point (keeps the ID over simulation time, experimental)
%ind_IN%	current local index of point in the MPI domain
%ind_IN_glob%	current GLOBAL index of point in the MPI domain
%ind_IN_glob_reduced%	current GLOBAl index of point, only active points
%ind_iopp%	iopp=index of opposite point.
%ind_IsolationFlag%	local high frequent part of local curvature, not considering contact angle effects
%ind_k_Un(1)%	Stagevalue inside a higher order Runge-Kutta time integration method like SDIRK2
%ind_k_Un(2)%	Stagevalue inside a higher order Runge-Kutta time integration method like SDIRK2
%ind_k_Un(3)%	Stagevalue inside a higher order Runge-Kutta time integration method like SDIRK2
%ind_k_Un(4)%	Stagevalue inside a higher order Runge-Kutta time integration method like SDIRK2
%ind_kappa%	curvature of the free surface boundary, smooth part
%ind_kappa_prime%	curvature of the free surface boundary, noisy part
%ind_kinEnergy%	variable for saving the kinetic energy
%ind_kob%	kob=kind of boundary
%ind_lastDTB_t%	last time of distance-to-boundary-computation
%ind_lastDTB_x(1)%	position of the last distance-to-boundary computation, x-component
%ind_lastDTB_x(2)%	position of the last distance-to-boundary computation, y-component
%ind_lastDTB_x(3)%	position of the last distance-to-boundary computation, z-component
%ind_layer%	layer information of boundary point
%ind_log_rho%	Temporary logarithm of density rho
%ind_MARKER%	unique marker (integer number) is point is flagged as irreducible
%ind_MCT(1,1)%	transformation matrix for coordinate transformation
%ind_MCT(1,2)%	transformation matrix for coordinate transformation
%ind_MCT(1,3)%	transformation matrix for coordinate transformation
%ind_MCT(2,1)%	transformation matrix for coordinate transformation
%ind_MCT(2,2)%	transformation matrix for coordinate transformation
%ind_MCT(2,3)%	transformation matrix for coordinate transformation
%ind_MCT(3,1)%	transformation matrix for coordinate transformation

%ind_MCT(3,2)%	transformation matrix for coordinate transformation
%ind_MCT(3,3)%	transformation matrix for coordinate transformation
%ind_med%	material index of point
%ind_medopp%	material index of opposite point
%ind_memorize_DeletePoint%	current MEMORIZE_Write status for MEMORIZE_Write deleting points
%ind_memorize_KeepPoint%	current MEMORIZE_Write status for MEMORIZE_Write keeping points
%ind_memorize_ReadPoint%	current MEMORIZE_Read status
%ind_mi_rep%	representative mass of the point
%ind_MOVE%	index of boundary move condition
%ind_MPIcommunicate%	number of MPI-processes to which this point has to be communicated
%ind_n(1)%	x-component of boundary normal
%ind_n(2)%	y-component of boundary normal
%ind_n(3)%	z-component of boundary normal
%ind_nbInteriorNeighbors%	number of regular INTERIOR neighbors found in the ball of radius h
%ind_nbRegularNeighbors%	number of regular neighbors found in the ball of radius h
%ind_next%	if points lined up, here the index of the point next in line is stored
%ind_nML(1)%	direction of the midline, nML=NormalizeddirectionMidLine
%ind_nML(2)%	direction of the midline
%ind_nML(3)%	direction of the midline
%ind_np(1)%	particular direction or some vector, used as dummy variable
%ind_np(2)%	particular direction or some vector, used as dummy variable
%ind_np(3)%	particular direction or some vector, used as dummy variable
%ind_nR(1)%	x-component of boundary normal in RealWorld (if coordinate transformation is activated)
%ind_nR(2)%	y-component of boundary normal in RealWorld (if coordinate transformation is activated)
%ind_nR(3)%	z-component of boundary normal in RealWorld (if coordinate transformation is activated)
%ind_ohh%	ohh=OneByHH, i.e. one divided by local smoothing length squared
%ind_ooh%	ooh=OrderOfH, order of smooting length
%ind_OrdApprox(1)%	Approximation order used for gradient computation
%ind_OrdApprox(2)%	Approximation order used for Laplace operator
%ind_Organize%	Current status of point with respect to point cloud organization
%ind_OrganizeDTB%	status for the distance to boundary computation

%ind_OrganizeDTMP%	status for the distance/projection to metaplanes
%ind_OrganizePC(1)%	state of filling interior MESHFREE points
%ind_OrganizePC(2)%	state of removing interior MESHFREE points
%ind_OrganizePC(3)%	state of filling MESHFREE points at boundaries
%ind_OrganizePC(4)%	state of removing/clustering of MESHFREE points at boundaries
%ind_OrganizePC(5)%	currently unused
%ind_pBubble%	internal pressure of macroscopic bubble
%ind_prev%	if points lined up, here the index of the point previous in line is stored
%ind_proc%	index of MPI-process the point currently belongs to
%ind_qualityOfGrad(1)%	quality of the gradient operator, item 1
%ind_qualityOfGrad(2)%	quality of the gradient operator, item 2
%ind_qualityOfGrad(3)%	quality of the gradient operator, item 3
%ind_r_rep%	representative density from the RepresentativeMassAlgorithm
%ind_r_residual%	residual of density
%ind_rhs(1)%	rhs = righ hand side, used for rhs/source terms in the differential equations to be solved
%ind_rhs(2)%	rhs = righ hand side, used for rhs/source terms in the differential equations to be solved
%ind_rhs(3)%	rhs = righ hand side, used for rhs/source terms in the differential equations to be solved
%ind_rhs(4)%	rhs = righ hand side, used for rhs/source terms in the differential equations to be solved
%ind_sha(1)%	shape function for boundary points
%ind_sha(2)%	shape function for boundary points
%ind_sha(3)%	shape function for boundary points
%ind_sha(4)%	shape function for boundary points
%ind_SlipState%	status of the slip status, i.e. sliding boundary points along slip walls
%ind_SlipState_cntnbts%	
%ind_st%	st=start time of point, point generation time in seconds of simulation time
%ind_SubDivision%	CURRENTLY INACTIVE: index for Kim`s postprocessing filter for MESHFREE points
%ind_T1D(1)%	Temperature of the 1D heat equation, result for MESHFREE boundary condition
%ind_T1D(i)%	Temperature of the 1D heat equation, i = 2:NB_POINTS_BC_HEAT_EQUATION_1D+1
%ind_t_Ux(1)%	Temporary x-derivative of physical entity U
%ind_t_Ux(2)%	Temporary x-derivative of physical entity U
%ind_t_Ux(3)%	Temporary x-derivative of physical entity U
%ind_t_Uy(1)%	Temporary y-derivative of physical entity U

%ind_t_Uy(2)%	Temporary y-derivative of physical entity U
%ind_t_Uy(3)%	Temporary y-derivative of physical entity U
%ind_t_Uz(1)%	Temporary z-derivative of physical entity U
%ind_t_Uz(2)%	Temporary z-derivative of physical entity U
%ind_t_Uz(3)%	Temporary z-derivative of physical entity U
%ind_TearOff%	marks a point in direct neighborhood of a FreeSurface-SolidWall junction
%ind_time%	present time, unit=seconds
%ind_v_Euler(1)%	transport velocity in EULERIAN framework Y(ind_v(1),i) - Y(ind_v_trans(1),i), x-component, unit=m/s
%ind_v_Euler(2)%	transport velocity in EULERIAN framework Y(ind_v(2),i) - Y(ind_v_trans(2),i), y-component, unit=m/s
%ind_v_Euler(3)%	transport velocity in EULERIAN framework Y(ind_v(3),i) - Y(ind_v_trans(3),i), z-component, unit=m/s
%ind_v_p(1)%	x-component of the velocity of boundary movement
%ind_v_p(2)%	x-component of the velocity of boundary movement
%ind_v_p(3)%	x-component of the velocity of boundary movement
%ind_v_residual(1)%	residual of velocity (x-component)
%ind_v_residual(2)%	residual of velocity (y-component)
%ind_v_residual(3)%	residual of velocity (z-component)
%ind_v_trans(1)%	velocity a point is actually moving with, x-component, unit=m/s
%ind_v_trans(2)%	velocity a point is actually moving with, y-component, unit=m/s
%ind_v_trans(3)%	velocity a point is actually moving with, z-component, unit=m/s
%ind_Vi%	volume represented by a point, unit=m^3
%ind_vol%	numerical weight of point
%ind_volBubble%	volume of macroscopic bubble
%ind_WettingCurvature%	additional curvature due to discrepancy between given and current contact angle between free surface and wall
%ind_WettingParticle%	contact of free surface points to regular wall boundary points
%ind_x(1)%	x-component of point position, unit=meters
%ind_x(2)%	y-component of point position, unit=meters
%ind_x(3)%	z-component of point position, unit=meters
%ind_x0(1)%	x-component initial point position, unit=meters
%ind_x0(2)%	y-component initial point position, unit=meters

%ind_x0(3)%	z-component initial point position, unit=meters
%ind_x_displaced(1)%	x-component of point position before distance to boundary computation, unit = meters
%ind_x_displaced(2)%	y-component of point position before distance to boundary computation, unit = meters
%ind_x_displaced(3)%	z-component of point position before distance to boundary computation, unit = meters
%ind_xR(1)%	x-component of point position in real coordinates, unit=meters
%ind_xR(2)%	y-component of point position in real coordinates, unit=meters
%ind_xR(3)%	z-component of point position in real coordinates, unit=meters

4.3.1. %ind_BC%

index of boundary condition

This entity carries the index of the BC -flag, that is usually provided in the
alias section by BC $BCflag$, see also AliasForGeometryItems .
DO NOT mismatch with %ind_kob% .

4.3.3. %ind_BNDfree_defect%

defect displacement of free surface with regards to the representative mass, clusterwise

only computed (different from zero) if RepresentativeMass_iData(8) = 2 .
It holds the value $\frac{D_{\text{pot}}^{k\,\text{cluster}}}{H_i}$, that is the potential correction displacement of the free surface relative to the local SMOOTH_LENGTH .
See RepresentativeMassAlgorithm

4.3.7. %ind_BVA_NUS(1)%

BVA_NUS=Bounbdary VAlue for NUSselt type

A Nusselt type B C is given by dPHI/dn = A+B*PHI, the value A is stored in %ind_BVA(1)% , the value of B ind %ind_BVA_NUS(1)% .
If vector valued functions are used, the (%ind_BVA(2)% ,%ind_BVA_NUS(2)%) and
(%ind_BVA(3)% ,%ind_BVA_NUS(3)%) will be used as well.
The array is temporyry during a time step and cannot be used for postprocessing.

4.3.10. %ind_ClusterSurface%

clustering of free surface or of initial regular boundary

During startup, clustering of the whole boundary is performed and saved in this variable index.
During time integration, if BUBBLE_DoTheManagement is switched, the raw index of current bubble clustering

is stored in this variable. This index is then copied to %ind_bndBubble% .

MESHFREE · Indices · General · %ind_EdgeValue%

4.3.11. %ind_EdgeValue%

pointcloud configuration without interior points: this item marks points at the edge of such configurations

In several situations, there is a local pointcloud configuration without interior points. That might happen in airbag applications with only thin layers between the membranes, or it might be due to a degeneration of a 3D-water phase if thin layers of liquid evolve. These situations are characterized by the absence of interior MESHFREE points. A compact thin layer of numerical MESHFREE points does not harm the numerics, but if this layer ends blindly (edge of an airbag or water front of a thin film), this can lead to numerical problems. Hence, here we try to, at least, mark these blindly ending thin layers of MESHFREE points.

The edge value takes values between 0 and 1 and is a measure for evaluating if the above described problematic situation occurs.

This feature is used so far only for GASDYN applications. For LIQUID applications, i.e. for the evolution of thin 3D-point layers, the edge value is also computed, but we rather use %ind_TearOff% and %ind_IsolationFlag% in order to characterize the configuration of the point cloud.

MESHFREE · Indices · General · %ind_ForceApproximation%

4.3.12. %ind_ForceApproximation%

marks points which are scheduled for re-approximation

re-approximation becomes particularly necessary for newly created points in holes, or as well for points resulting from clustering.

MESHFREE · Indices · General · %ind_IN%

4.3.14. %ind_IN%

current local index of point in the MPI domain

All points, also inactive ones (dry points at the walls for example), have their proper index. Counting starts at 1 for each MPI-process

MESHFREE · Indices · General · %ind_IN_glob%

4.3.15. %ind_IN_glob%

current GLOBAL index of point in the MPI domain

counting of the index starts at 1 at MPI process 0 and continues in the next MPI-domain. Also inactive points count.

MESHFREE · Indices · General · %ind_IN_glob_reduced%

4.3.16. %ind_IN_glob_reduced%

current GLOBAl index of point, only active points

only active points are counted, such that we have a complete chain of indices without interruptions

MESHFREE · Indices · General · %ind_IsolationFlag%

4.3.17. %ind_IsolationFlag%

local high frequent part of local curvature, not considering contact angle effects

This entity is similar to %ind_kappa_prime% , however, unlike %ind_kappa_prime% , it does not take into account the contact angle information

MESHFREE · Indices · General · %ind_MARKER%

4.3.18. %ind_MARKER%

unique marker (integer number) is point is flagged as irreducible

see especially %MONITORPOINTS_CREATION_IrreducibleFPMpoint%

MESHFREE · Indices · General · %ind_MCT(1,1)%

4.3.19. %ind_MCT(1,1)%

transformation matrix for coordinate transformation

An infinitessimal vector DxT in the transformed space will be mapped to
to the real world space by

DxR = MCT*DxT

MESHFREE · Indices · General · %ind_MOVE%

4.3.28. %ind_MOVE%

index of boundary move condition

this entity carries the index of the MOVE -flag, that is usually provided in the alias section by MOVE $MOVEflag$

MESHFREE · Indices · General · %ind_MPIcommunicate%

4.3.29. %ind_MPIcommunicate%

number of MPI-processes to which this point has to be communicated

PURE POSTPROCESSING!!!!!!!!! In general, use this item to visualize the number of neighbor MPI-processes, only.

MESHFREE · Indices · General · %ind_Organize%

4.3.32. %ind_Organize%

Current status of point with respect to point cloud organization

It can take the following values:
%ORGANIZE_none%
%ORGANIZE_CandidateForFreeSurface%
%ORGANIZE_WasPushedToFreeSurface0%
%ORGANIZE_WasPushedToFreeSurface%
%ORGANIZE_WasCreatedNearMetaplanes%
%ORGANIZE_WasPushedBackFromBoundary%
%ORGANIZE_HasCreatedMonitorPoint%
%ORGANIZE_CreatedByShallowWater%
%ORGANIZE_CreatedByTouchDownOfFreeSurface%
%ORGANIZE_IsIsolated%
%ORGANIZE_WasNotConsideredForActivation%
%ORGANIZE_DeactivationDueToLackOfInteriorParticles%
%ORGANIZE_ActivationDueToLackOfFreeSurface%
%ORGANIZE_ExplicitelyCheckedForActivation%
%ORGANIZE_CandidateForAtivation%
%ORGANIZE_HasRunThroughActivationProcedure%
%ORGANIZE_IsNotActive%
%ORGANIZE_MeanReduction%
%ORGANIZE_MinReduction%
%ORGANIZE_MaxReduction%

4.3.33. %ind_OrganizeDTB%

status for the distance to boundary computation

Reveals the status of the latest distance to boundary computation.
Mostly used for debuggin greasons, thats why the numbers are not inuitive.

%ind_OrganizeDTB% == %ORGANIZE_none% == 0 -> nothing done for this point (i.e. regular boundary point etc.)
== 0.1 -> no distance check as no boundary MESHFREE point found in neighborhood and no boundary is on reduced filling mode
== 0.2 -> no distance check as all boundary MESHFREE points see the current point in inside direction (only if no boundary is in reduces filling mode)
== 0.3 -> marked for distance to boundary computation
== 0.4 -> no boundary found in neighborhood
== 0.45 -> point projects to a %BND_blind% -boundary
== 0.5 -> point projects to regular boundary
== 0.51 -> point project down to a nofill boundary (special: only first time step)
== 0.6 -> point is checked for penetration thorough boundary
== 0.61 -> intersection with boundary is found
== 0.62 -> intersection with boundary is found and point really will be reprojected, i.e. push back ontop of the boundary
== 0.63 -> intersection with boundary is found, but reproject is risky (too long projection distance) and thus point is deleted
== 0.7 -> regular MESHFREE point, enough distance to boundary
== 0.72 -> distance to boundary smaller than dist_LayerThickness , but point not treated as it stems from tear off at edges
== 0.73 -> distance to boundary smaller than dist_LayerThickness , but also less than zero, so further treatment launched (see 0.8 and bigger)
== 0.74 -> isolated MESHFREE point, pushed due to the value of dist_LayerThickness given in common_vairables.dat
== 0.75 -> MESHFREE point who provided to small layer thickness compared to the given value of dist_LayerThickness , hence position of point is adjusted
== 0.76 -> thickening thin layer due to more or less perpendicular interaction of free surface point with boundary
== 0.8 -> free surface point finally INSIDE
== 0.9 -> free surface point OUTSIDE
== 0.91 -> free surface point OUTSIDE, but stemming from tear off edge
== 0.92 -> free surface point OUTSIDE, but stemming from tear off edge
== 0.93 -> free surface point OUTSIDE, too for from boundary, i.e. deleted
== 0.94 -> free surface point OUTSIDE, that might have penetrated trough thin walls, i.e. deletd

== 0.95 -> free surface point OUTSIDE, reprjected to boundary if necessary
> 1 -> regular removal of point
== %ORGANIZE_CreatedByTouchDownOfFreeSurface% == 88 -> reprojection to boundary completed
== %ORGANIZE_IsInGap% == 77 -> ONLY regular boundary point: is in a geometrical gap

---

4.3.34. %ind_OrganizeDTMP%

status for the distance/projection to metaplanes

if point currently IS in contact with metaplane value=1, if it was previously, value=-1, if not contact, value=0

4.3.35. %ind_OrganizePC(1)%

state of filling interior MESHFREE points

Y %ind_OrganizePC(1)% == 0 : no action of interior-point-filling was taken for this point
Y %ind_OrganizePC(1)% == 1 : this MESHFREE point was scheduled to fill new interior MESHFREE point
Y %ind_OrganizePC(1)% == 2 : this point was scheduled to inject new interior MESHFREE point
Y %ind_OrganizePC(1)% == 3 : this point was scheduled and actually created another new interior MESHFREE point in its neighborhood
Y %ind_OrganizePC(1)% == 4 : this point was scheduled and actually injected new interior MESHFREE point in the direction of its boundary normal
Y %ind_OrganizePC(1)% == 5 : this point just was created by a MESHFREE point in its neighborhood
Y %ind_OrganizePC(1)% == 6 : this point was just injected by another (boundary) point in its neighborhood

---

4.3.36. %ind_OrganizePC(2)%

state of removing interior MESHFREE points

Y %ind_OrganizePC(2)% == 0 : no removal action taken based on this MESHFREE point
Y %ind_OrganizePC(2)% == 1 : this MESHFREE point is the result of a clustering operation of two MESHFREE points into one
Y %ind_OrganizePC(2)% == -1 : this MESHFREE point is marked for deletion and will be remove at the beginning of the next time step

---

4.3.37. %ind_OrganizePC(3)%

state of filling MESHFREE points at boundaries

Y %ind_OrganizePC(3)% == -2 : hole search not executed for this time step
Y %ind_OrganizePC(3)% == 0 : no action of surface/boundary filling taken for this point
Y %ind_OrganizePC(3)% == 1 : scheduled for hole search in its neighborhood

Y %ind_OrganizePC(3)% == 1.1 : point not allowed to fill because of then BOUNDARYFILLING -flag or an appropriate definition of ORGANIZE_ReducedFillingOfWalls
Y %ind_OrganizePC(3)% == 1.4 : point prepared successfully for local hole search
Y %ind_OrganizePC(3)% == 2 : actually created new point in its neighborhood
Y %ind_OrganizePC(3)% == 3 : point was just created by an already existing point in the neighborhood
Y %ind_OrganizePC(3)% == 8 : creation of boundary point motivated by thin layers

Special values for **filling of free surfaces** :

Y %ind_OrganizePC(3)% == 0.0 : no hole search scheduled for this point
Y %ind_OrganizePC(3)% == 0.1 : hole search scheduled but did not find enough neighbors for surface triangulation
Y %ind_OrganizePC(3)% == 0.2 : found enough neighbors, but no interior/wall points found in the neighbor stencil -> surface triangulatin oskipped
Y %ind_OrganizePC(3)% == 0.3 : surface triangulation was performed, did not find candidates for hole filling (all triangles small enough)
Y %ind_OrganizePC(3)% == 0.4 : same as 0.3; some candidates were rejected because triangle consisted of only on free surface point
Y %ind_OrganizePC(3)% == 0.5 : same as 0.3; some candidates were rejected because center of circumcircle was not insice the triangle itself
Y %ind_OrganizePC(3)% == 0.6 : same as 0.3; some candidates were rejected due to the angle criterium $\mathbf{n}_i^T \cdot \mathbf{n}_j < (-0.3)$ (any two normals at the triangle corners
Y %ind_OrganizePC(3)% == 0.7 : surface triangulatin was performed, found some candidates
Y %ind_OrganizePC(3)% == 2 : actually created a new free surface point in its neighborhood
Y %ind_OrganizePC(3)% == 3 : point was just created by an already existing neighbor point
Y %ind_OrganizePC(3)% == 4 : point was created AND scheduled for the "BringToSurface" algorithm
Y %ind_OrganizePC(3)% == 5 : point was created, and the "BringToSurfcae"-algorithm was effected
Y %ind_OrganizePC(3)% == 6 : point was previously interior and changed to %BND_free% , it is automaticlly scheduled for the bring-to-surface-treatment
Y %ind_OrganizePC(3)% == 7 : point was previously interior and changed to %BND_free% , additionally the "BringToSurfcae"-algorithm was effected
Y %ind_OrganizePC(3)% == -11: actually fulfills criterion to become free surface point, but rejected for obvious reasons
Y %ind_OrganizePC(3)% == -22: actually checked for being free surface, but does not fulfill the appropriate criteria

4.3.38. %ind_OrganizePC(4)%

state of removing/clustering of MESHFREE points at boundaries

Y %ind_OrganizePC(4)% == -2 : not scheduled for removal operation
Y %ind_OrganizePC(4)% == 0 : no removal action applied to this MESHFREE point
Y %ind_OrganizePC(4)% == 1 : two points clustered into one. They have been closer to each other than (dist_rip * H). Mean average is taken from both original points.
Y %ind_OrganizePC(4)% == 2 : two points clustered into one. They have been closer to each other than (0.01*H), which might come into being by creating new points (accross MPI-process bounds, concurrent OMP-threads)
Y %ind_OrganizePC(4)% == 3 : one point removed because too close to the present point. REMARK: two points cannot be clustered if one is interior and one is boundary. The boundary point is kept, the interior one is removed.

4.3.40. %ind_SlipState%

status of the slip status, i.e. sliding boundary points along slip walls

... currently experimental

4.3.42. %ind_SubDivision%

CURRENTLY INACTIVE: index for Kim`s postprocessing filter for MESHFREE points

does not work, so index switched off

4.3.45. %ind_TearOff%

marks a point in direct neighborhood of a FreeSurface-SolidWall junction

If a wall point is directly adjacent to a free surface, then %ind_TearOff% will
become positive and carries the index of the adjacent free surface point.

4.3.46. %ind_Vi%

volume represented by a point, unit=m^3

LIQUID Volume of point estimated from the delaunay triangulation around the point
DROPLETPHASE : Volume of spherical particle with radius %ind_d30% .

4.3.47. %ind_WettingCurvature%

additional curvature due to discrepancy between given and current contact angle between free surface and wall

If an free surface point has direct contact to a regular boundary/wall, this index
shows the additional surface curvature that comes into being due to differences between the
given contact angle (see BC_WettingAngle) and the current (measured) contact angle between
hte free surface and the regular boundary

4.3.48. %ind_WettingParticle%

contact of free surface points to regular wall boundary points

If an free surface point has direct contact to a regular boundary/wall, this index
contains the index of the wall point it has contact with.

4.3.49. %ind_act%

activation status of a boundary point

contains the number of time steps the point was active without break. For inactive points, we have 0 or (temporarily) a negative number

4.3.50. %ind_addvar%

additional variables that can be used for additional tasks (legacy code)

%ind_addvar% is kept for backwards compatibility, only. The current best practice is described in UserDefinedIndices .
Additional variables can be the following:
%ind_addvar(1)%
%ind_addvar(2)%
%ind_addvar(3)%
%ind_addvar(4)%
%ind_addvar(5)%
%ind_addvar(6)%
%ind_addvar(7)%
%ind_addvar(8)%
%ind_addvar(9)%
They can be freely used in the input file USER_common_variables .

If the user intends to use additional variables, the number of additional variables (N_addvar) has to be given in common_variables .

Using %ind_addvar(9)% for N_addvar = 3 will lead to serious problems during code execution.

List of members:

%ind_addvar(1)%	additional variable that can be used for additional tasks (legacy code)
%ind_addvar(2)%	additional variable that can be used for additional tasks (legacy code)
%ind_addvar(3)%	additional variable that can be used for additional tasks (legacy code)
%ind_addvar(4)%	additional variable that can be used for additional tasks (legacy code)
%ind_addvar(5)%	additional variable that can be used for additional tasks (legacy code)
%ind_addvar(6)%	additional variable that can be used for additional tasks (legacy code)
%ind_addvar(7)%	additional variable that can be used for additional tasks (legacy code)
%ind_addvar(8)%	additional variable that can be used for additional tasks (legacy code)
%ind_addvar(9)%	additional variable that can be used for additional tasks (legacy code)

4.3.51. %ind_bndBubble%

index of macroscopic bubbles

Macroscopic bubbles are deteced in the BubbleAlgorithm . A point that is part of the surface of such a bubble is marked with its index. Each enclosed volume (bubble) obtains its proper index. Paticles forming a bubble can be active free surface points (%BND_free%) or inactive boundary points (%BND_wall% , %BND_slip% , %BND_inflow% , %BND_outflow%).

4.3.55. %ind_cluster%

unique cluster index of pointcloud

Cluster computation is invoked by SCAN_ClustersOfConnectivity. It delivers a unique cluster index for each MESHFREE point, also in MPI-mode. The resulting cluster index is stored in this variable.

4.3.56. %ind_connectBcBubble%

if bubble connected tro outflow, this item holds to BC_PASSON of the outflow boundary

If connected to %BND_outflow% A N D BC_PASSON $OutflowPassonBC$ is given, then the boundary condition ind_connectBcBubble is aplied for free surface points

4.3.59. %ind_dbp%

dbp=distance between phases, unit=meters

if phase contact exists, this item provides the measured distance between the two phases

4.3.65. %ind_div_bar_c%

PURE POSTPROCESSING: the (divergence of velocity)^bar at the point in the numerical scheme where the correction pressure is computed

see the Scheme v-- and vp-

4.3.66. %ind_div_bar_pDyn%

PURE POSTPROCESSING: the (divergence of velocity)^bar at the point in the numerical scheme where the correction pressure is computed

see the Scheme v-- and vp-

4.3.70. %ind_dtb%

dtb=distance to boundary, unit=meters

Y %ind_dtb% contains the distance to boundary of the point, **if** the distance to boundary was computed. The distance to boundary is computed only if necessary for performance reasons.

distance to boundary computed	value in Y%ind_dtb%
yes	absolute distance to boundary
no	maximum distance (local smoothing length)

To evaluate for which points the distance to boundary was computed see Y %ind_dtb_status% , for the algorithm ORGANIZE_DistanceToBoundary_Version .

4.3.72. %ind_dtb_status%

status of the new distance-to-boundary-computation

Y %ind_dtb_status%	meaning	distance to boundary (dtb) computed
= 0.0		no
< 1.0	value is the nondimensional distance	yes
= 1.1	mother point, no boundary triangles found in the neigborhood	no
= 1.2	child point: no boundary triangles found in the neighborhood	no
= 1.3	child point: no dtb computation necessary because mother point to far from boundary	no
= 1.5	dtb computation effected, but measured distance bigger than the presumed maximum distance	no

Also see %ind_dtb% and ORGANIZE_DistanceToBoundary_Version .

4.3.73. %ind_event_AbortFPM%

current event status for event stopping MESHFREE

It can take the following values:
0 -- points which do not activate aborting of MESHFREE
1 -- points which activate stopping aborting at this time step

4.3.74. %ind_event_DeletePoint%

current event status for event deleting points

It can take the following values:
0 -- points not being a neighbor of at this time step deleted point

-1 -- neighboring points of at this time step deleted point

4.3.75. %ind_event_FunctionManipulation%

current event status for function manipulation event

It can take the following values:
0 -- points not influenced by any function manipulation event
1 -- points directly influenced by a function manipulation event in the current time step
-1 -- neighboring points of at this time step directly influenced points
0.1 -- points filled by at this time step directly influenced points
>1 -- points that have been previously affected by a function manipulation event

4.3.76. %ind_event_GeometricalFunctionManipulation%

current event status for geometrical function manipulation event

It can take the following values:
0 -- points not influenced by any geometrical function manipulation event
1 -- points directly influenced by a geometrical function manipulation event in the current time step
-1 -- neighboring points of at this time step directly influenced points
0.1 -- points filled by at this time step directly influenced points
>1 -- points that have been previously affected by a geometrical function manipulation event

Note: A geometrical function manipulation event changes at least one of:

- %ind_x(1)% , %ind_x(2)% , %ind_x(3)%
- %ind_n(1)% , %ind_n(2)% , %ind_n(3)%
- %ind_kob%
- %ind_sha(1)% , %ind_sha(2)% , %ind_sha(3)% , %ind_sha(4)%
- %ind_BC%

Points that have been influenced by a geometrical function manipulation event are marked for the free surface check irrelevant of their current kob-value (%ind_kob%).

4.3.77. %ind_event_Msg%

current event status for event print message

It can take the following values:
0 -- points which do not activate message
1 -- points which activate message printing at this time step

4.3.78. %ind_event_SaveResults%

current event status for event saving computational results

It can take the following values:

0 -- points which do not activate saving of computational results
1 -- points which activate saving of computational results at this time step

4.3.79. %ind_event_StopFPM%

current event status for event stopping MESHFREE

It can take the following values:
0 -- points which do not activate stopping MESHFREE
1 -- points which activate stopping MESHFREE at this time step

4.3.80. %ind_event_WriteRestart%

current event status for event writing of a restart file

It can take the following values:
0 -- points which do not activate writing of a restart file
1 -- points which activate writing of a restart file at this time step

4.3.81. %ind_event_WriteResume%

current event status for event writing of a resume file

It can take the following values:
0 -- points which do not activate writing of a resume file
1 -- points which activate writing of a resume file at this time step

4.3.82. %ind_h%

local smoothing length, unit=meters

see SmoothingLength

4.3.83. %ind_h_adaptive%

local smoothing length proposed for adaptive treatment of H, unit=meters

In case of USER_h_funct = 'ADTV' as well as USER_h_funct = 'ADDS', this variable contains the proposed value of smoothing length. See SmoothingLength .

4.3.84. %ind_iopp%

iopp=index of opposite point.

if a point is located at a phase boundary and a communicating partner (opposite point) was found, then this item provides the index of this point. See PHASE_distinction

---

4.3.85. %ind_k_Un(1)%

Stagevalue inside a higher order Runge-Kutta time integration method like SDIRK2

This is only used in Eulerian Framework. Size depends on spatial dimension nue and velocity-pressure-solver (v-- or vp-).

Therefore the 4-th component is needed in case of 3-dimensional problems (nue = 3) by using the coupled solver vp- .

4.3.86. %ind_k_Un(2)%

Stagevalue inside a higher order Runge-Kutta time integration method like SDIRK2

This is only used in Eulerian Framework. Size depends on spatial dimension nue and velocity-pressure-solver (v-- or vp-).

Therefore the 4-th component is needed in case of 3-dimensional problems (nue = 3) by using the coupled solver vp- .

4.3.87. %ind_k_Un(3)%

Stagevalue inside a higher order Runge-Kutta time integration method like SDIRK2

This is only used in Eulerian Framework. Size depends on spatial dimension nue and velocity-pressure-solver (v-- or vp-).

Therefore the 4-th component is needed in case of 3-dimensional problems (nue = 3) by using the coupled solver vp- .

4.3.88. %ind_k_Un(4)%

Stagevalue inside a higher order Runge-Kutta time integration method like SDIRK2

This is only used in Eulerian Framework. Size depends on spatial dimension nue and velocity-pressure-solver (v-- or vp-).

Therefore the 4-th component is needed in case of 3-dimensional problems (nue = 3) by using the coupled solver vp- .

4.3.89. %ind_kappa%

curvature of the free surface boundary, smooth part

the curvature of the free surface is split into a smooth part and a fluctuation part:

kappa_measured = kappa + kappa_prime

In general, the measurement of the curvature of the free surface is noisy due to the point
locations, the smooth part kappa
however provides a good value that goes into the boundary condition for the hydrostatic pressure.

4.3.90. %ind_kappa_prime%

curvature of the free surface boundary, noisy part

kappa_measured = kappa + kappa_prime

4.3.91. %ind_kinEnergy%

variable for saving the kinetic energy

$\|\mathbf{v}\|_2^2$ is stored instead of $E_{\text{kin}} = \frac{m}{2}\|\mathbf{v}\|_2^2$ because the value is only used to calculate $E_{\text{kin}}^{\text{old}} / E_{\text{kin}}^{\text{new}}$.

4.3.92. %ind_kob%

kob=kind of boundary

Type of boundary of the MESHFREE point in aspects related to **geometrical organization**

type of point (geom)	value of %ind_kob%
interior point	%BND_none%
free surface point	%BND_free%
regular boundary point	inherited from IDENT , i.e. %BND_slip% , %BND_wall% , %BND_inflow% etc.

Note: %ind_kob% resp. IDENT only describe the way of geometrical organization of the pointcloud. THEY DO NOT describe the type of physical boundary condition. Physical boundary conditions are flagged by BC (see also %ind_BC%) and defined by BC_v , BC_p etc (see BoundaryConditions).

4.3.93. %ind_lastDTB_t%

last time of distance-to-boundary-computation

Last time when the distance-to-boundary operation has been executed for this point, see also ORGANIZE_DistanceToBoundary_Version .

4.3.94. %ind_lastDTB_x(1)%

position of the last distance-to-boundary computation, x-component

Last position when the distance-to-boundary operation has been executed for this point, see also ORGANIZE_DistanceToBoundary_Version .

4.3.95. %ind_lastDTB_x(2)%

position of the last distance-to-boundary computation, y-component

Last position when the distance-to-boundary operation has been executed for this point, see also ORGANIZE_DistanceToBoundary_Version .

4.3.96. %ind_lastDTB_x(3)%

position of the last distance-to-boundary computation, z-component

Last position when the distance-to-boundary operation has been executed for this point, see also ORGANIZE_DistanceToBoundary_Version .

4.3.97. %ind_layer%

layer information of boundary point

layer index of the boundary element, which the MESHFREE point has closest distance to

4.3.98. %ind_log_rho%

Temporary logarithm of density rho

This is only used in Eulerian Framework. It is needed for the computation of compressible flows because of the required term v*grad(log(rho)) in the continuity equation.

4.3.99. %ind_med%

material index of point

Material Tag, defined by PhysicalProperties

4.3.101. %ind_memorize_DeletePoint%

current MEMORIZE_Write status for MEMORIZE_Write deleting points

It can take the following values:
0 -- points not being a neighbor of at this time step deleted point
-1 -- neighboring points of at this time step deleted point

4.3.102. %ind_memorize_KeepPoint%

current MEMORIZE_Write status for MEMORIZE_Write keeping points

It can take the following values:
1 -- point that is kept
0 -- points not being a neighbor of at this time step kept point
-1 -- neighboring points of at this time step kept point

4.3.103. %ind_memorize_ReadPoint%

current MEMORIZE_Read status

It can take the following values:
1 -- point read in from MEMORIZE_File at current time step
0 -- point not read in from MEMORIZE_File at current time step

4.3.104. %ind_mi_rep%

representative mass of the point

see RepresentativeMassAlgorithm and DefinitionRepresentativeMass . Only filled if Representative Mass algorithm is turned on:

```
RepresentativeMass_iData = ( 1, ...)
```


4.3.108. %ind_nML(1)%

direction of the midline, nML=NormalizeddirectionMidLine

Pure postprocessing: can be used as information in USER_common_variables , however there is no dependency of the MESHFREE -code on this variable(s)
The midline was computed in tank filling applications, i.e. a virtual line along the center of the filling pipe. In longitudinal direction, space was compressed, such that in this direction, fewer MESHFREE points have to be used

4.3.114. %ind_nbInteriorNeighbors%

number of regular INTERIOR neighbors found in the ball of radius h

The original number of neighbors after seach in the ball with radius h. For the simulation, the number of neighbors can be reduced by max_N_stencil and max_N_stencil_INTERIOR .

MESHFREE · Indices · General · %ind_nbRegularNeighbors%

4.3.115. %ind_nbRegularNeighbors%

number of regular neighbors found in the ball of radius h

The original number of neighbors after seach in the ball with radius h. For the simulation, the number of neighbors can be reduced by max_N_stencil and max_N_stencil_INTERIOR .

MESHFREE · Indices · General · %ind_next%

4.3.116. %ind_next%

if points lined up, here the index of the point next in line is stored

Lining up is an option if EULER is used. In this case, the transport operators (v*grad()) can be better approximated if points are lined up due to the velocity field. Invoke lining up by
KOP(...) = ... EULER ... POINTS:LINEUP
in the solver line of USER_common_variables . Besides this item, %ind_prev% is also important.

MESHFREE · Indices · General · %ind_np(1)%

4.3.117. %ind_np(1)%

particular direction or some vector, used as dummy variable

usually, in this variable the gradient of pressure is stored, i.e. grad_p = (Y %ind_np(1)% ,Y %ind_np(2)% ,IY %ind_np(3)%)

MESHFREE · Indices · General · %ind_ooh%

4.3.121. %ind_ooh%

ooh=OrderOfH, order of smooting length

this entity is deprecated, the order of smooting length i.e. one divided by local smoothing length was important for the formerly used box-based neighbor search (UseBoxSystemVersion=1). For the tree-based neighbor search (UseBoxSystemVersion=2), ind_ooh does not have any meaning

MESHFREE · Indices · General · %ind_pBubble%

4.3.122. %ind_pBubble%

internal pressure of macroscopic bubble

internal pressure af the bubble this point belongs to. It is computed according to the BubbleAlgorithm .

4.3.123. %ind_prev%

if points lined up, here the index of the point previous in line is stored

Lining up is an option if EULER is used. In this case, the transport operators (v*grad()) can be better approximated if points are lined up due to the velocity field. Invoke lining up by
KOP(...) = ... EULER ... POINTS:LINEUP
in the solver line of USER_common_variables . Besides this item, %ind_next% plays a role.

4.3.125. %ind_qualityOfGrad(1)%

quality of the gradient operator, item 1

Let $\mathbf{C}_i^\nabla$ be the Nx3-matrix containing the gradient operators of point with index "i", then the value contained in this item is $H_i\sqrt{\|\mathbf{C}_i^{\nabla T}\cdot\mathbf{C}_i^\nabla\|_2}$

where H_i is the local SMOOTH_LENGTH .

4.3.126. %ind_qualityOfGrad(2)%

quality of the gradient operator, item 2

biggest relative error if applying some linear function to the discrete gradient operator $\mathbf{C}^\nabla$

4.3.127. %ind_qualityOfGrad(3)%

quality of the gradient operator, item 3

biggest relative error if applying some QUADRATIC function to the discrete gradient operator $\mathbf{C}^\nabla$

4.3.128. %ind_r_rep%

representative density from the RepresentativeMassAlgorithm

see RepresentativeMassAlgorithm and DefinitionRepresentativeDensity .

4.3.129. %ind_r_residual%

residual of density

We assume the general equation of mass conservation

$$\frac{d\rho}{dt} + \rho \nabla^T \boldsymbol{v} = 0$$

and define the residuum as to be

$$r_\rho = \frac{\rho^{n+1} - \rho^n}{\Delta t} + \rho^{n+1} \nabla^T \boldsymbol{v}^{n+1}$$

4.3.134. %ind_sha(1)%

shape function for boundary points

kind of point	value
regular boundary point (placed on a boundary element %ind_BE1%)	A position of a boundary point can be expressed by the node points of the element, on which the point is placed: x = sha(1)*A + sha(2)*B + sha(3)*C + sha(4)*D where A, B, C, D are the node points of the boundary element. In case of a quad, all four are used, in case of a triangle, D is unused and sha(4)=0. In case of a line, C, D are void and therefore sha(3)=sha(4)=0.
interior, free surface or phase boundary point	the vector (Y %ind_sha(1)% ,Y %ind_sha(2)% ,Y %ind_sha(3)%) represents the boundary normal of the particular boundary point, which the point has its closest distance to.

4.3.135. %ind_sha(2)%

shape function for boundary points

see %ind_sha(1)%

4.3.136. %ind_sha(3)%

shape function for boundary points

see %ind_sha(1)%

4.3.137. %ind_sha(4)%

shape function for boundary points

see %ind_sha(1)%

4.3.138. %ind_st%

st=start time of point, point generation time in seconds of simulation time

Generation time: when was the point created within the simulation.

MESHFREE · Indices · General · %ind_t_Ux(1)%

4.3.139. %ind_t_Ux(1)%

Temporary x-derivative of physical entity U

This is only used in Eulerian Framework. It is needed for the MUSCL-reconstruction in order to approximate the function values at the auxiliary points, which are no part of the point cloud.
Size depends on dimension of the computed entity. For scalar entities like temperature it has size = 1, accordingly for velocity size = 2 resp. size = 3.

MESHFREE · Indices · General · %ind_t_Ux(2)%

4.3.140. %ind_t_Ux(2)%

Temporary x-derivative of physical entity U

This is only used in Eulerian Framework. It is needed for the MUSCL-reconstruction in order to approximate the function values at the auxiliary points, which are no part of the point cloud.
Size depends on dimension of the computed entity. For scalar entities like temperature it has size = 1, accordingly for velocity size = 2 resp. size = 3.

MESHFREE · Indices · General · %ind_t_Ux(3)%

4.3.141. %ind_t_Ux(3)%

Temporary x-derivative of physical entity U

This is only used in Eulerian Framework. It is needed for the MUSCL-reconstruction in order to approximate the function values at the auxiliary points, which are no part of the point cloud.
Size depends on dimension of the computed entity. For scalar entities like temperature it has size = 1, accordingly for velocity size = 2 resp. size = 3.

MESHFREE · Indices · General · %ind_t_Uy(1)%

4.3.142. %ind_t_Uy(1)%

Temporary y-derivative of physical entity U

This is only used in Eulerian Framework. It is needed for the MUSCL-reconstruction in order to approximate the function values at the auxiliary points, which are no part of the point cloud.
Size depends on dimension of the computed entity. For scalar entities like temperature it has size = 1, accordingly for velocity size = 2 resp. size = 3.

MESHFREE · Indices · General · %ind_t_Uy(2)%

4.3.143. %ind_t_Uy(2)%

Temporary y-derivative of physical entity U

This is only used in Eulerian Framework. It is needed for the MUSCL-reconstruction in order to approximate the function values at the auxiliary points, which are no part of the point cloud.

Size depends on dimension of the computed entity. For scalar entities like temperature it has size = 1, accordingly for velocity size = 2 resp. size = 3.

4.3.144. %ind_t_Uy(3)%

Temporary y-derivative of physical entity U

This is only used in Eulerian Framework. It is needed for the MUSCL-reconstruction in order to approximate the function values at the auxiliary points, which are no part of the point cloud.
Size depends on dimension of the computed entity. For scalar entities like temperature it has size = 1, accordingly for velocity size = 2 resp. size = 3.

4.3.145. %ind_t_Uz(1)%

Temporary z-derivative of physical entity U

This is only used in Eulerian Framework. It is needed for the MUSCL-reconstruction in order to approximate the function values at the auxiliary points, which are no part of the point cloud.
Size depends on dimension of the computed entity. For scalar entities like temperature it has size = 1, accordingly for velocity size = 3.

4.3.146. %ind_t_Uz(2)%

Temporary z-derivative of physical entity U

This is only used in Eulerian Framework. It is needed for the MUSCL-reconstruction in order to approximate the function values at the auxiliary points, which are no part of the point cloud.
Size depends on dimension of the computed entity. For scalar entities like temperature it has size = 1, accordingly for velocity size = 3.

4.3.147. %ind_t_Uz(3)%

Temporary z-derivative of physical entity U

This is only used in Eulerian Framework. It is needed for the MUSCL-reconstruction in order to approximate the function values at the auxiliary points, which are no part of the point cloud.
Size depends on dimension of the computed entity. For scalar entities like temperature it has size = 1, accordingly for velocity size = 3.

4.3.152. %ind_v_p(1)%

x-component of the velocity of boundary movement

each boundary point belongs to a boundary element, which travels with a certain velocity. The travelling

velocity is explicitely given by (Y %ind_v_p(1)% ,Y %ind_v_p(2)% ,Y %ind_v_p(3)%)
If the point is free surface and in contact to another phase, the boundary velocity is the speed of the opposite point/phase

4.3.155. %ind_v_residual(1)%

residual of velocity (x-component)

see %ind_v_residual(3)%

4.3.156. %ind_v_residual(2)%

residual of velocity (y-component)

see %ind_v_residual(3)%

4.3.157. %ind_v_residual(3)%

residual of velocity (z-component)

We assume the general equation of momentum as to be

$$\frac{d\boldsymbol{v}}{dt} + \frac{1}{\rho}\nabla p = \frac{1}{\rho}\left(\nabla^T \boldsymbol{S}\right)^T + \boldsymbol{g}$$

From this, we define the residuum

$$r_{\boldsymbol{v}} = \frac{\boldsymbol{v}^{n+1} - \boldsymbol{v}^n}{\Delta t} + \frac{1}{\rho}\nabla p^{n+1} - \frac{1}{\rho}\left(\nabla^T \boldsymbol{S}^{n+1}\right)^T - \boldsymbol{g}^{n+1}$$

4.3.158. %ind_v_trans(1)%

velocity a point is actually moving with, x-component, unit=m/s

in case of LAGRANGE , %ind_v% and %ind_v_trans% are the same, however in case of EULER ,
%ind_v_trans% really represents the velocity the
pointcloud is moving with, so it might be absolutely different from %ind_v%

4.3.161. %ind_vol%

numerical weight of point

this value is usually 1 for active points, and 0 for inactive points. For critical, but active points, the weight can be reduced, however this option is actually not used.

4.3.162. %ind_volBubble%

volume of macroscopic bubble

Volume of the bubble this point belongs to. It is computed according to the BubbleAlgorithm .

MESHFREE · Indices · General · %ind_x0(1)%

4.3.166. %ind_x0(1)%

x-component initial point position, unit=meters

with the ind_x0-variables, a comparison between initial location of point
at the beginning of the time step and the final location
at the end of the time step is possible

MESHFREE · Indices · General · %ind_x0(2)%

4.3.167. %ind_x0(2)%

y-component initial point position, unit=meters

with the ind_x0-variables, a comparison between initial location of point
at the beginning of the time step and the final location
at the end of the time step is possible

MESHFREE · Indices · General · %ind_x0(3)%

4.3.168. %ind_x0(3)%

z-component initial point position, unit=meters

with the ind_x0-variables, a comparison between initial location of point
at the beginning of the time step and the final location
at the end of the time step is possible

MESHFREE · Indices · General · %ind_xR(1)%

4.3.169. %ind_xR(1)%

x-component of point position in real coordinates, unit=meters

if coordinate transformation is used, ind_x delivers the position in the transormed world and ind_xR delivers the position in the real world

MESHFREE · Indices · General · %ind_xR(2)%

4.3.170. %ind_xR(2)%

y-component of point position in real coordinates, unit=meters

if coordinate transformation is used, ind_x delivers the position in the transormed world and in_xR delivers the position in

the real world

MESHFREE · Indices · General · %ind_xR(3)%

4.3.171. %ind_xR(3)%

z-component of point position in real coordinates, unit=meters

if coordinate transformation is used, ind_x delivers the position in the transormed world and in_xR delivers the position in the real world

MESHFREE · Indices · LIQUID

4.4. LIQUID

indices for the implicit (incompressible/weakly compressible) solver

The indices used here might be also used for other solvers like GASDYN , SHALLOWWATER etc.

List of members:	
%ind_alpha%	
%ind_betaDarcy%	porous material coupling parameter, unit: 1/s
%ind_BNDpnt_of_pnt_near BND%	index of closest boundary point for all points which are close to boundary
%ind_c%	correction pressure due to projecting the velocity field onto correct div(v) values
%ind_cD%	
%ind_CV%	specific heat capacity, unit: J/(kg*K)
%ind_CV_LatentHeat%	
%ind_d30%	
%ind_DarcyVersion%	How to compute the source terms of the Darcy contributions in the pressure equations
%ind_DiagPcorr%	compressibility of the fluid
%ind_diss%	
%ind_div%	measured, instantaneous divergence of velocity
%ind_div_bar%	compression rate due to given temperature or hydrostatic pressure or density time change rate
%ind_div_bar_0%	compression rate at the previous time step
%ind_div_tild%	devergence of preliminary velocity
%ind_divS(1)%	divergence of solid stress tensore, x-component [Pa/m]
%ind_divS(2)%	divergence of solid stress tensore, y-component [Pa/m]
%ind_divS(3)%	divergence of solid stress tensore, z-component [Pa/m]

%ind_divV_sw%	
%ind_divV_transport%	divergence of the transport velocity, internally used for EULER applications
%ind_dt_store%	variable for storing the intermediate time step size in case of subcyclings in Eulerian framework
%ind_dt_virt%	value of the current local virtual time step size [s]
%ind_dtbp%	distance to closest boundary point
%ind_eps%	k-epsilon model: turbulent dissipation [m^2/s^3]
%ind_eps_dot%	
%ind_eps_plastic%	plastic deformation, accumulated over time
%ind_eps_plastic_dot%	current time change rate of the plastic deformation [1/s]
%ind_eps_plastic_dot_dot%	
%ind_ETA%	viscosity, unit: Pa*s
%ind_ETA_eff%	effective dynamic viscosity (sum of laminar and turbulent viscosities), unit: Ns/(m^2)
%ind_ETA_sm%	total viscosity, consisting of physical, turbulent, and additional numerical viscosities; unit: Ns/(m^2)
%ind_g(1)%	
%ind_g(2)%	
%ind_g(3)%	
%ind_hwf_3d%	
%ind_k%	k-epsilon model: turbulent kinetic energy [m^2/s^2]
%ind_LAM%	heat conductivity, unit: W/(m*K)
%ind_lap_vn%	
%ind_LatentHeat%	
%ind_logVi_ist(1)%	time-integrated local defect volume
%ind_logVi_ist(2)%	time-integrated relative local defect volume
%ind_logVi_soll%	time-integrated relative required volume
%ind_MomSrc(1)%	
%ind_MomSrc(2)%	
%ind_MomSrc(3)%	
%ind_MUE%	shear modulus, unit: N/m^2
%ind_MUE_mean%	
%ind_MUE_relax%	

%ind_MUE_sm%	shear modulus, after numerical smoothing, unit: N/m^2
%ind_NUE_turb%	turbulent kinematic viscosity, unit m^2/s
%ind_p%	hydrostatic pressure
%ind_p_0%	hydrostatic pressure at previous time step
%ind_p_corr%	This index is deprecated. Please use ind_p_dyn for the same functionality.
%ind_p_corr_0%	This index is deprecated. Please use ind_p_dyn_0 for the same functionality
%ind_p_dyn%	dynamic pressure
%ind_p_dyn_0%	dynamic pressure at previous time step
%ind_pDivV%	
%ind_penalty%	
%ind_PenV%	
%ind_PHI%	
%ind_pnt_nearBND%	mark MESHFREE points near boundary
%ind_PSI%	
%ind_r%	density, unit: kg/m^3
%ind_r_0%	density at previous time step
%ind_r_AddDispPh%	
%ind_r_c%	
%ind_R_P%	partial derivative of density with respect to pressure
%ind_r_pDyn%	
%ind_r_sm%	
%ind_SIG%	surface tension, unit: N/m
%ind_SlidingState%	
%ind_Smises%	vonMises-norm of solid stress tensor [Pa]
%ind_Sn(1)%	Stress tensor times boundary normal, i.e. stresses acting on surface, unit=Pa
%ind_Sn(2)%	Stress tensor times boundary normal, i.e. stresses acting on surface, unit=Pa
%ind_Sn(3)%	Stress tensor times boundary normal, i.e. stresses acting on surface, unit=Pa
%ind_SrelaxTime%	
%ind_Sxx%	solid stress tensor xx-component [Pa]
%ind_Sxy%	solid stress tensor xy-component [Pa]
%ind_Sxz%	solid stress tensor xz-component [Pa]
%ind_Syy%	solid stress tensor yy-component [Pa]

%ind_Syz%	solid stress tensor yz-component [Pa]
%ind_Szz%	solid stress tensor zz-component [Pa]
%ind_T%	Temperature, unit: Kelvin, Celsius
%ind_T_0%	temperature [K, °C] at previous time step
%ind_tauW%	turbulent wall shear stress [N/m^2]
%ind_TurbulentWallLayer%	distance of artificial shift of MESHFREE points at boundary towards the interior if turbulence model is switched on
%ind_v(1)%	x-component of velocity vector
%ind_v(2)%	y-component of velocity vector
%ind_v(3)%	z-component of velocity vector
%ind_v0Darcy(1)%	velocity of the porous basis material, x-component, unit: m/s
%ind_v0Darcy(2)%	velocity of the porous basis material, y-component, unit: m/s
%ind_v0Darcy(3)%	velocity of the porous basis material, z-component, unit: m/s
%ind_v_0(1)%	velocity of the previous time step, x-component
%ind_v_0(2)%	velocity of the previous time step, y-component
%ind_v_0(3)%	velocity of the previous time step, z-component
%ind_v_3d(1)%	
%ind_v_3d(2)%	
%ind_v_3d(3)%	
%ind_v_tild(1)%	velocity before correction, x-component
%ind_v_tild(2)%	velocity before correction, y-component
%ind_v_tild(3)%	velocity before correction, z-component
%ind_v_times_v0%	scalar product (v-v_p)*(v0-v_p)
%ind_vn_a%	
%ind_vn_b%	
%ind_vn_n%	
%ind_vrel(1)%	
%ind_vrel(2)%	
%ind_vrel(3)%	

4.4.4. %ind_DarcyVersion%

How to compute the source terms of the Darcy contributions in the pressure equations

This index is used only if the user triggers it in the USER_common_vdaribles input file by defining it.
We try to provide four different ways of computing the Darcy term $\Theta(\mathbf{v})$ and its splitting into the dynamic and hydrostatic parts Θ_{hyd} and Θ_{dyn} , see DerivePoissonEquationForPressure and Definition is necessary by initialization

```
INITDATA ($MaterialFlag$,%ind_DarcyVersion%) = RightHandSideExpression
```

Refinement during runtime of MESHFREE by

```
CODI_eq ($MaterialFlag$,%ind_DarcyVersion%) = RightHandSideExpression
```

Make sure that the RightHandSideExpressions provide integers of 1, 2, 3, or 4.
Default: INITDATA ($MaterialFlag$,%ind_DarcyVersion%) = 1

The different option/versions are discussed in ComputationOfTHETA .

As soon as all research is concluded, the present property will move to the input option FLIQUID_ConsistentPressure_Version

MESHFREE · Indices · LIQUID · %ind_DiagPcorr%

4.4.5. %ind_DiagPcorr%

compressibility of the fluid

represents the term $\frac{1}{\Delta t}\frac{1}{\rho}\frac{\partial \rho}{\partial p}$. See for example DesiredAndNominalDivergenceOfVelocity

MESHFREE · Indices · LIQUID · %ind_ETA_eff%

4.4.7. %ind_ETA_eff%

effective dynamic viscosity (sum of laminar and turbulent viscosities), unit: Ns/(m^2)

We have

$$\eta_{\text{eff}} = \eta + c_\mu \rho \frac{k^2}{\epsilon}$$

See KepsilonAlgorithm .
Additionally, η_{eff} is smoothed, see %ind_ETA_sm% .

The viscosity is exactly the one used for the computation of the viscous stress tensor in EquationsToSolve .

MESHFREE · Indices · LIQUID · %ind_ETA_sm%

4.4.8. %ind_ETA_sm%

total viscosity, consisting of physical, turbulent, and additional numerical viscosities; unit: Ns/(m^2)

The variable represents the total viscosity $\hat{\eta}$ used in the VelocityAlgorithm .

$$\hat{\eta} = \eta + c_\mu \rho \frac{k^2}{\epsilon} + C \cdot \Delta t \cdot \mu$$

%ind_ETA_sm% is smoothed before usage in VelocityAlgorithm in order to stabilize the numerical solution by relaxing jumps, especially towards the boundary.
For smoothing, see also COMP_nbSmooth_Eta and COMP_facSmooth_Eta .
For stability, see also COMP_AdjustEtaEff .

%ind_ETA_sm% is the smoothed %ind_ETA_eff% if
$$\mu = 0$$

For turbulence modeling, see KepsilonAlgorithm .

The paramter C can be defined by COEFF_mue .

4.4.18. %ind_NUE_turb%

turbulent kinematic viscosity, unit m^2/s

$$\nu_{\text{turb}} = c_\mu \frac{k^2}{\epsilon}$$

and
$$c_\mu = 0.09$$

See KepsilonAlgorithm .

4.4.25. %ind_Smises%

vonMises-norm of solid stress tensor [Pa]

$\|\boldsymbol{S}_s\|_{Mises}$, see StressTensorAlgorithm

4.4.26. %ind_Sn(1)%

Stress tensor times boundary normal, i.e. stresses acting on surface, unit=Pa

x-component of
$$(\boldsymbol{S}_{visc} + \boldsymbol{S}_{solid} - (p_{hyd} + p_{dyn})\boldsymbol{I}) \cdot \boldsymbol{n}$$

4.4.27. %ind_Sn(2)%

Stress tensor times boundary normal, i.e. stresses acting on surface, unit=Pa

y-component of
$$(\boldsymbol{S}_{visc} + \boldsymbol{S}_{solid} - (p_{hyd} + p_{dyn})\boldsymbol{I}) \cdot \boldsymbol{n}$$

4.4.28. %ind_Sn(3)%

Stress tensor times boundary normal, i.e. stresses acting on surface, unit=Pa

z-component of

$$(\boldsymbol{S}_{visc} + \boldsymbol{S}_{solid} - (p_{hyd} + p_{dyn})\boldsymbol{I}) \cdot \boldsymbol{n}$$

4.4.30. %ind_Sxx%

solid stress tensor xx-component [Pa]

$\boldsymbol{S}_s^{xx}$, see StressTensorAlgorithm

4.4.31. %ind_Sxy%

solid stress tensor xy-component [Pa]

$\boldsymbol{S}_s^{xy}$, see StressTensorAlgorithm

4.4.32. %ind_Sxz%

solid stress tensor xz-component [Pa]

$\boldsymbol{S}_s^{xz}$, see StressTensorAlgorithm

4.4.33. %ind_Syy%

solid stress tensor yy-component [Pa]

$\boldsymbol{S}_s^{yy}$, see StressTensorAlgorithm

4.4.34. %ind_Syz%

solid stress tensor yz-component [Pa]

$\boldsymbol{S}_s^{yz}$, see StressTensorAlgorithm

4.4.35. %ind_Szz%

solid stress tensor zz-component [Pa]

$\boldsymbol{S}_s^{zz}$, see StressTensorAlgorithm

4.4.38. %ind_TurbulentWallLayer%

distance of artificial shift of MESHFREE points at boundary towards the interior if turbulence model is switched on

this provides an addtional point of support needed for the computation of the logarithmic velocity profile in the boundary layer

MESHFREE · Indices · LIQUID · %ind_betaDarcy%

4.4.40. %ind_betaDarcy%

porous material coupling parameter, unit: 1/s

This index stores the porous material coupling parameter β in EquationsToSolve .

MESHFREE · Indices · LIQUID · %ind_c%

4.4.41. %ind_c%

correction pressure due to projecting the velocity field onto correct div(v) values

see CorrectionPressureAlgorithm

MESHFREE · Indices · LIQUID · %ind_d30%

4.4.43. %ind_d30%

mean diameter

MESHFREE · Indices · LIQUID · %ind_divS(1)%

4.4.46. %ind_divS(1)%

divergence of solid stress tensore, x-component [Pa/m]

the solid stress tensor is $\boldsymbol{S}_s$, see StressTensorAlgorithm

MESHFREE · Indices · LIQUID · %ind_divS(2)%

4.4.47. %ind_divS(2)%

divergence of solid stress tensore, y-component [Pa/m]

the solid stress tensor is $\boldsymbol{S}_s$, see StressTensorAlgorithm

MESHFREE · Indices · LIQUID · %ind_divS(3)%

4.4.48. %ind_divS(3)%

divergence of solid stress tensore, z-component [Pa/m]

the solid stress tensor is $\boldsymbol{S}_s$, see StressTensorAlgorithm

MESHFREE · Indices · LIQUID · %ind_div_bar%

4.4.51. %ind_div_bar%

compression rate due to given temperature or hydrostatic pressure or density time change rate

will be different from zero only if the density is dependent on time and/or temperature and/or pressure

MESHFREE · Indices · LIQUID · %ind_div_bar_0%

4.4.52. %ind_div_bar_0%

compression rate at the previous time step

see %ind_div_bar%

MESHFREE · Indices · LIQUID · %ind_div_tild%

4.4.53. %ind_div_tild%

devergence of preliminary velocity

the preliminary velocity is marked $\tilde{v}$, especially see VelocityAlgorithm and CorrectionPressureAlgorithm

MESHFREE · Indices · LIQUID · %ind_dt_virt%

4.4.55. %ind_dt_virt%

value of the current local virtual time step size [s]

see VirtualTimeStepSize

MESHFREE · Indices · LIQUID · %ind_eps%

4.4.57. %ind_eps%

k-epsilon model: turbulent dissipation [m^2/s^3]

See KepsilonAlgorithm .

MESHFREE · Indices · LIQUID · %ind_k%

4.4.66. %ind_k%

k-epsilon model: turbulent kinetic energy [m^2/s^2]

See KepsilonAlgorithm .

MESHFREE · Indices · LIQUID · %ind_p%

4.4.71. %ind_p%

hydrostatic pressure

compute the hydrostatic pressure prior to the velocity computations, see HydrostaticPressureAlgorithm

MESHFREE · Indices · LIQUID · %ind_p_corr%

4.4.74. %ind_p_corr%

This index is deprecated. Please use ind_p_dyn for the same functionality.

compute the dynamic pressure after computing the velocity at the new time level. Especially for incompressible flows, the dynmic pressure is uniquely determined by the velocity, see DynamicPressureAlgorithm

MESHFREE · Indices · LIQUID · %ind_p_dyn%

4.4.76. %ind_p_dyn%

dynamic pressure

compute the dynamic pressure after computing the velocity at the new time level. Especially for incompressible flows, the dynmic pressure is uniquely determined by the velocity, see DynamicPressureAlgorithm

MESHFREE · Indices · LIQUID · %ind_pnt_nearBND%

4.4.79. %ind_pnt_nearBND%

mark MESHFREE points near boundary

This has only relevance if transport equation are numerically solved, i.e. in case of EULER

MESHFREE · Indices · LIQUID · %ind_tauW%

4.4.86. %ind_tauW%

turbulent wall shear stress [N/m^2]

This quantity is computed for each slip boundary. If a noslip boundary is concerned, this value shall be negligible.

MESHFREE · Indices · LIQUID · %ind_v_times_v0%

4.4.102. %ind_v_times_v0%

scalar product (v-v_p)(v0-v_p)*

v=correcnt velocity, v_p=velocity of the boundary, v0=velocity of the previous time step. If negative, the sense of the flow turned around wrt the boundary.

MESHFREE · Indices · MANIFOLD

4.5. MANIFOLD

indices for the manifold phase

The indices used here might be also used for other solvers like LIQUID , GASDYN , etc.

List of members:

%ind_c%	
%ind_div%	
%ind_div_tild%	divergence of preliminary velocity

%ind_dtb%	(mean) distance to boundary (distance to other manifold chamber). Only two chambers possible.
%ind_ETA%	
%ind_ETA_sm%	
%ind_g(1)%	
%ind_g(2)%	
%ind_g(3)%	
%ind_kappa%	(mean) curvature of surface(- 0.5*surface divergence of manifold normal for 2-surfaces in 3 space)
%ind_MUE%	
%ind_n_ManBnd(1)%	x-component of boundary normal of manifold
%ind_n_ManBnd(2)%	y-component of boundary normal of manifold
%ind_n_ManBnd(3)%	z-component of boundary normal of manifold
%ind_p%	
%ind_p_corr%	
%ind_p_dyn%	
%ind_r%	
%ind_r_sm%	
%ind_SIG%	
%ind_T%	Temperature, unit: Kelvin, Celsius
%ind_v(1)%	x-component of velocity vector (velocity relative to manifold - should be tangential)
%ind_v(2)%	y-component of velocity vector (velocity relative to manifold - should be tangential)
%ind_v(3)%	z-component of velocity vector (velocity relative to manifold - should be tangential)
%ind_v_0(1)%	ind_v of previous time step, x-component
%ind_v_0(2)%	ind_v of previous time step, y-component
%ind_v_0(3)%	ind_v of previous time step, z-component
%ind_v_p(1)%	x-component of manifold velocity vector
%ind_v_p(2)%	y-component of manifold velocity vector
%ind_v_p(3)%	z-component of manifold velocity vector
%ind_v_p_0(1)%	ind_v_p of previous time step, x-component
%ind_v_p_0(2)%	ind_v_p of previous time step, y-component
%ind_v_p_0(3)%	ind_v_p of previous time step, z-component
%ind_v_tild(1)%	
%ind_v_tild(2)%	

%ind_v_tild(3)%

4.6. POPBAL

Indices for the population balance solver

This list is not yet complete

4.7. SHALLOWWATER

Indices for the shallow water solver

See also SHALLOWWATER .

List of members:

%ind_div%	
%ind_div_control%	
%ind_divV_sw%	
%ind_divV_uw%	
%ind_ETA%	
%ind_ETA_eff%	
%ind_g(1)%	
%ind_g(2)%	
%ind_g(3)%	
%ind_gradP_uw(1)%	
%ind_gradP_uw(2)%	
%ind_gradP_uw(3)%	
%ind_hwf%	Height Of water/liquid Film, unit: m
%ind_hwf_0%	height of liquid layer previous time step
%ind_hwf_3d%	
%ind_hwf_control%	
%ind_kob_she%	
%ind_lap_vn%	
%ind_n_sm(1)%	

%ind_n_sm(2)%	
%ind_n_sm(3)%	
%ind_nR_sm(1)%	
%ind_nR_sm(2)%	
%ind_nR_sm(3)%	
%ind_p%	This index is deprecated. Please use ind_p_dyn for the same functionality.
%ind_p_uw%	
%ind_PHI%	
%ind_r%	
%ind_SIG%	
%ind_SlidingState%	
%ind_T%	
%ind_Umbrella(1)%	
%ind_Umbrella(2)%	
%ind_Umbrella(3)%	
%ind_v(1)%	velocity of water film, x-component, unit: m/s
%ind_v(2)%	velocity of water film, y-component, unit: m/s
%ind_v(3)%	velocity of water film, z-component, unit: m/s
%ind_v_3d(1)%	
%ind_v_3d(2)%	
%ind_v_3d(3)%	
%ind_v_uw(1)%	
%ind_v_uw(2)%	
%ind_v_uw(3)%	
%ind_vtang(1)%	
%ind_vtang(2)%	
%ind_vtang(3)%	
%ind_vtang_0(1)%	
%ind_vtang_0(2)%	
%ind_vtang_0(3)%	

4.8. TRANSPORT

MESHFREE indices for TRANSPORT, i.e. solving hyperbolic problems

PhD thesis of Tobias Seifarth

List of members:	
%ind_v(1)%	x-component of velocity vector
%ind_v(2)%	y-component of velocity vector
%ind_v(3)%	z-component of velocity vector
%ind_divV_transport%	divergence of the transport velocity
%ind_solute_rate%	solute rate of a stone like halite
%ind_dt_store%	variable for storing the intermediate time step size in case of subcyclings in Eulerian framework
%ind_T%	Temperature
%ind_T_0%	
%ind_p%	pressure
%ind_p_dyn%	correction pressure
%ind_r%	density
%ind_h%	smoothing length
%ind_pnt_nearBND%	identifies MESHFREE points close to the boundary
%ind_dtbp%	distance to closest boundary point
%ind_BNDpnt_of_pnt_nearBND%	index of closest boundary point for all points which are close to boundary
%ind_p_corr%	This index is deprecated. Please use ind_p_dyn for the same functionality

MESHFREE · Indices · UserDefinedIndices

4.9. UserDefinedIndices

user defined indices

As a postprocessing feature, users can define own indices. At the startup phase, MESHFREE scans the input file (USER_common_variables) for occurences of indices of the forms:

```
%indU_...%
%indC_...%
```

No matter, where it occurs or for what reason, MESHFREE will create this index as additional index. It is then treated like a usual MESHFREE index of the form %ind_...%.

Types of user defined indices

There are two types of indices and they differ in the way they are updated:

- %indU_...% are continuously evaluated indices; whenever a new point is created, its value for %indU_...% is interpolated from neighbor values.
- %indC_...% are discretely evaluated indices; whenever a new point is created, its value for %indC_...% is inherited from the originating point.

Assignment and usage

The user defined indices of the form %indU_...% can be used, especially, in the CODI - and EVENT -context. However, all other functionalities like INITDATA , SAVE_ITEM , RightHandSideExpression etc. work in the same way.
The user defined indices of the form %indC_...% can be used in order to discretely colorize the simulation domain for analysis. In the %indU_...% case, this would smear out as the values would be interpolated.

Logging

The complete collection of indices and other MESHFREE variables of the form %...% can be found in the file List_of_FPMvariables.log in the (hidden) directory .FPM_log FPM_ID=nnnnnnnnn/.
For an example, see tut3d_08 .

---

5. __Constants__

typical %...%-constants that can be used in the input files

On this site, a collection of all %...%-keywords are updated, which may be used within the MESHFREE -input files.

The %...%-keywords are given in alphabetical order.

By clicking on one of the items, one finds a list of links, where the
given keyword appears in one or the other way.

As the documentation is dynamically growing and developing, the links to the given keywords will grow appropriately, which might help navigating the documentation.

See also Indices .

List of members:

%ACTIVE_noinit%

%AND%

%ASSIGN_FUNCTIONVALUE%

%AVERAGE_BND%

%AVERAGE_FS%

%AVERAGE_INT%

%AVERAGE_XYPLANE%

%BCON_CG%

%BCON_contact%

%BCON_E_tot%

%BCON_explicit%

%BCON_far_field%

%BCON_free%

%BCON_free_NoVisc%

%BCON_implicit%

%BCON_inflow%

%BCON_Ma%

%BCON_Mdot%

%BCON_none%

%BCON_outflow%

%BCON_p%

%BCON_p_tot%

%BCON_PAMCRASH_CG%

%BCON_PAMCRASH_Mdot%

%BCON_PAMCRASH_RG%

%BCON_PAMCRASH_T%

%BCON_RG%

%BCON_rho%

%BCON_rho_va%

%BCON_rho_vb%

%BCON_rho_vn%

%BCON_s%

%BCON_s_ini%	
%BCON_slip%	
%BCON_T%	
%BCON_T_tot%	
%BCON_va%	
%BCON_vb%	
%BCON_Vdot%	
%BCON_visc%	
%BCON_vn%	
%BCON_wall%	
%BCON_wall_nosl%	
%BE_INTEGRATION_DIRECT%	
%BE_INTEGRATION_DIRECT_TIME%	
%BND_arcl%	
%BND_arcs%	
%BND_AVERAGE%	
%BND_blind%	
%BND_BlindAndEmpty%	
%BND_BNDDOT%	
%BND_CAUCHY%	
%BND_COLLISION%	
%BND_contact%	
%BND_contact_Explicit%	
%BND_corner%	
%BND_COULOMB_SLIDE%	
%BND_COULOMB_SLIP%	
%BND_COULOMB_STICK%	
%BND_count_BE%	parameter for the real()-function (equation parser)
%BND_count_NP%	parameter for the real()-function (equation parser)
%BND_cube%	
%BND_cut%	cutting off points at metaplanes if used as IDENT
%BND_cylinder%	

%BND_DIRICH%

%BND_disk%

%BND_DRYFRICTION%

%BND_DRYFRICTION_InContact%

%BND_edge%

%BND_far_field%

%BND_fixed%

%BND_free%

%BND_free_Barodesy%

%BND_free_HypoPlast%

%BND_free_implicit%

%BND_free_implicit_InContact%

%BND_free_implicit_InContact_Explicit%

%BND_free_InContact%

%BND_free_InContact_Explicit%

%BND_free_NoVisc%

%BND_HEATFLUX%

%BND_HELMHOLTZ%

%BND_IGES%

%BND_IGES_curve%

%BND_IGES_ignore%

%BND_IGES_surface%

%BND_IGES_trafo%

%BND_inflow%

%BND_INTERPHASE%

%BND_INTERPHASE_dfdn%

%BND_INTERPHASE_f%

%BND_LAPLAC%

%BND_line%

%BND_Manifold_Free%

%BND_Manifold_Interior%

%BND_Manifold_Slip%

%BND_NEUMANN%	
%BND_NEUMANN_DIRICHLET%	
%BND_node%	
%BND_none%	
%BND_NUSSEL%	
%BND_outflow%	
%BND_plane%	
%BND_point%	
%BND_quad%	
%BND_RADIATION%	
%BND_ROBIN%	
%BND_slip%	
%BND_slip_InContact%	
%BND_slip_InContact_Explicit%	
%BND_SYSTUS%	
%BND_tria%	
%BND_tria6N%	
%BND_void%	
%BND_VONNEU%	
%BND_wall%	
%BND_wall_InContact%	
%BND_wall_InContact_Explicit%	
%BND_wall_NoLayerThickness%	
%BND_wall_nosl%	
%BNDpoints_ExtractFromNodes%	
%BNDSLIP_ReprojectedAfterPassingOpenEdge%	mark state of slip movement of MESHFREE points along boundary
%BNDSLIP_TearOffAtOpenEdge%	mark state of slip movement of MESHFREE points along boundary
%BNDSLIP_TearOffAtRegularEdge%	mark state of slip movement of MESHFREE points along boundary
%BOUNDARYFILLING_Always%	
%BOUNDARYFILLING_Never%	
%BOUNDARYFILLING_OnlyIfActiveItself%	

%BOUNDARYFILLING_OnlyInActiveNeighborhood%	
%BUBBLE_EQN_TruePressure%	parameter for the real()-function (equation parser)
%CHAMBER_BGK%	
%CHAMBER_DROPLETPHASE%	
%CHAMBER_Euler%	
%CHAMBER_EulerExpl%	
%CHAMBER_GASDYN%	
%CHAMBER_IT_000%	
%CHAMBER_IT_tes%	
%CHAMBER_IT_v00%	
%CHAMBER_IT_vp0%	
%CHAMBER_IT_vpT%	
%CHAMBER_KEPS%	
%CHAMBER_Lagrange%	
%CHAMBER_LineUp%	
%CHAMBER_LIQUID%	
%CHAMBER_MANIFOLD%	
%CHAMBER_NoLineUp%	
%CHAMBER_None%	
%CHAMBER_PARTICLEPHASE%	
%CHAMBER_POPBAL%	
%CHAMBER_SHALLOWWATER%	
%CHAMBER_STANDBY%	
%CLOCK_STATISTICS_FLIQUID%	parameter for the real()-function (equation parser)
%CLOCK_STATISTICS_ORGANIZE%	parameter for the real()-function (equation parser)
%CLOCK_STATISTICS_TOTAL_FLIQUID%	parameter for the real()-function (equation parser)
%CLOCK_STATISTICS_TOTAL_ORGANIZE%	parameter for the real()-function (equation parser)
%CLOCK_STATISTICS_TOTAL_SAMG%	parameter for the real()-function (equation parser)
%CODI_explicit%	
%CODI_implicit%	
%ConsistencyChecksAtStartup_STOP%	

%ConsistencyChecksAtStartup_WARNING%	
%CONSTRUCT_Area%	
%CONSTRUCT_BoxMax%	
%CONSTRUCT_BoxMidPoint%	
%CONSTRUCT_BoxMidPoint_Abs%	
%CONSTRUCT_BoxMin%	
%CONSTRUCT_COG%	
%CONSTRUCT_EstablishCurveVolumeVersusHeight%	
%CONSTRUCT_IncludeIGESfaces%	provoke usage of IGES faces in CONSTRUCT statements
%CONSTRUCT_Normal%	
%CONSTRUCT_NormalDividedByArea%	
%CONSTRUCT_PointBasedOnAbsoluteVolume%	
%CONSTRUCT_PointBasedOnRelativeVolume%	
%CONSTRUCT_Tangent1%	
%CONSTRUCT_Tangent2%	
%CONSTRUCT_Volume%	
%CONSTRUCT_VolumeForGivenHeight%	
%CONVERT_toInteger%	
%COORDTRANS_cone%	
%COORDTRANS_linear%	
%COORDTRANS_radial%	
%COORDTRANS_ring%	
%COORDTRANS_spherical%	
%COUPLE_PAM%	
%COUPLE_SYSTUS%	
%CouplingBFT_OtherSimulation_IsFPM%	
%CouplingBFT_RequestMyselfToWait%	
%CouplingBFT_RequestOtherProcessToWait%	
%CPU_STATISTICS_FLIQUID%	parameter for the real()-function (equation parser)
%CPU_STATISTICS_ORGANIZE%	parameter for the real()-function (equation parser)
%CPU_STATISTICS_TOTAL_FLIQUID%	parameter for the real()-function (equation parser)

%CPU_STATISTICS_TOTAL_ORGANIZE%	parameter for the real()-function (equation parser)
%CUMU_ASSIGN%	
%CUMU_INTERVAL%	
%CUMU_NONE%	
%CUMU_SIMULATION%	
%CUMU_SMOOTH%	
%CUMU_SMOOTH_AreaBased%	
%CUMU_SMOOTH_StopAtEdges%	
%DIFFOP_gradient_DELAUNAY%	
%DIFFOP_gradient_GASDYN%	
%DIFFOP_gradient_MLS%	
%DIFFOP_laplace_GASDYN%	
%DIFFOP_laplace_LS%	
%DIFFOP_laplace_MLS%	
%DIFFOP_laplace_optimized%	
%DIFFOP_laplace_simplex%	
%DropletSource_doNotCreateDropletsOutside%	optional parameter for the DropletSource
%DropletSource_provideCounter%	parameter for the real()-function (equation parser)
%DropletSource_provideCurrentVolume%	parameter for the real()-function (equation parser)
%DropletSource_provideTargetVolume%	parameter for the real()-function (equation parser)
%ElapsedTimeIntegrationCycle%	parameter for the real()-function (equation parser)
%ElapsedTimePointOrganization%	parameter for the real()-function (equation parser)
%EQN_JOINT_F(1)%	
%EQN_JOINT_F(2)%	
%EQN_JOINT_F(3)%	
%EQN_JOINT_M(1)%	
%EQN_JOINT_M(2)%	
%EQN_JOINT_M(3)%	
%EQN_JOINT_x(1)%	
%EQN_JOINT_x(2)%	
%EQN_JOINT_x(3)%	

%EQN_nbsum_filtered%	Select filtered list
%EQN_nbsum_nonfiltered%	Select non-filtered list
%EQN_Proj_ALL%	projection of a MESHFREE-entity from a different chamber using all types of points (interior and boundary)
%EQN_Proj_BND%	projection of a MESHFREE-entity from a different chamber using only boundary points
%EQN_Proj_INT%	projection of a MESHFREE-entity from a different chamber using only interior points
%EQN_Reduct_Accumulated%	
%EQN_Reduct_iCluster%	
%EtaGrad_Classical%	
%EtaGrad_Identity%	
%EVENT_AbortFPM%	
%EVENT_DeletePoint%	
%EVENT_FunctionManipulation%	
%EVENT_Msg%	
%EVENT_PerformAfterHowManyTimeCycles%	
%EVENT_SaveResults%	
%EVENT_StopFPM%	
%EVENT_WriteRestart%	
%EVENT_WriteResume%	
%FLIQUID_NbParticles%	parameter for the real()-function (equation parser)
%FPM_KineticEnergy%	parameter for the real()-function (equation parser)
%FPM_KineticEnergy_Defect_gradPv%	parameter for the real()-function (equation parser)
%FPM_KineticEnergy_Defect_O2%	parameter for the real()-function (equation parser)
%FPM_KineticEnergy_Defect_rhogDv%	parameter for the real()-function (equation parser)
%FPM_KineticEnergy_DifferenceInOrganize%	parameter for the real()-function (equation parser)
%FPM_KineticEnergy_DifferenceInOrganize2%	parameter for the real()-function (equation parser)
%FPM_KineticEnergy_DifferenceInTimeStep%	parameter for the real()-function (equation parser)
%FPM_RepMass_CreatedByDropletSource%	parameter for the real()-function (equation parser)
%FPM_RepMass_CreatedByInflowOutflow%	parameter for the real()-function (equation parser)
%FPM_RepMass_DeletedAtMetaplanes%	parameter for the real()-function (equation parser)
%FPM_VOLUME_ACTUAL%	parameter for the real()-function (equation parser)

%FPM_VOLUME_DeletedAtMetaplanes%	parameter for the real()-function (equation parser)
%FPM_VOLUME_TARGET%	parameter for the real()-function (equation parser)
%GASDYN_Mass%	parameter for the real()-function (equation parser)
%GASDYN_MassAnalytical%	parameter for the real()-function (equation parser)
%GASDYN_MassCorrection%	parameter for the real()-function (equation parser)
%GASDYN_TotalEnergy%	parameter for the real()-function (equation parser)
%GASDYN_TotalEnergyAnalytical%	parameter for the real()-function (equation parser)
%GASDYN_TotalEnergyCorrection%	parameter for the real()-function (equation parser)
%GEO_close%	
%GEO_Inside%	
%GEO_open%	
%GEO_Outside%	
%GEO_removeBasedOnCOG%	
%GEO_removeBasedOnNodes%	
%GEO_RemoveClusterByIndex%	
%GEO_RemoveClusterClosestToGivenPoint%	
%GEO_Tube%	
%GEO_Vector%	
%GradtEtaGrad_DirectApproximation%	
%GradtEtaGrad_Identity%	
%GradtEtaGrad_None%	
%GradtEtaGrad_StarStencil%	
%H_constant%	
%H_linear%	
%H_radial%	
%H_ring%	
%H_spherical%	
%HEAT_EQ_1D_BC%	optional parameter for temperature boundary condition
%INTEGRATION_ABSFLUX%	
%INTEGRATION_ABSFLUX_TIME%	
%INTEGRATION_BND%	
%INTEGRATION_BND_DIRECT%	

%INTEGRATION_BND_DIRECT_Proj_BND%	
%INTEGRATION_BND_DIRECT_TIME%	
%INTEGRATION_BND_DIRECT_TIME_Proj_BND%	
%INTEGRATION_BND_OUTSIDE%	
%INTEGRATION_BND_TIME%	
%INTEGRATION_Comment%	comment/remark specifier for integration statements
%INTEGRATION_FilterByTime%	
%INTEGRATION_FilterByTimestepCounter%	
%INTEGRATION_FLUX%	
%INTEGRATION_FLUX_DROPLETPHASE%	
%INTEGRATION_FLUX_TIME%	
%INTEGRATION_FS%	
%INTEGRATION_FS_DIRECT%	
%INTEGRATION_FS_DIRECT_TIME%	
%INTEGRATION_FS_TIME%	
%INTEGRATION_Header%	header information identifier for INTEGRATION
%INTEGRATION_INT%	
%INTEGRATION_INT_TIME%	
%INTEGRATION_Percentile%	restrict intergration/min/max to a selected percentile-subset regarding a given function
%INTEGRATION_Remark%	comment/remark specifier for intergration statements
%INTEGRATION_SkipByTime%	
%INTEGRATION_SkipByTimestepCounter%	
%INTEGRATION_Values(1)%	deprecated
%INTEGRATION_Values(2)%	deprecated
%INTEGRATION_Values(3)%	deprecated
%INTEGRATION_Values(4)%	deprecated
%INTEGRATION_Values(5)%	deprecated
%MASSFLOW_DROPLETPHASE%	
%MAXIMUM_BE%	
%MAXIMUM_BENP%	
%MAXIMUM_BND%	

%MAXIMUM_FS%	
%MAXIMUM_INT%	
%MED_ADBLUE%	
%MED_air%	
%MED_BARODESY%	
%MED_CARREAU%	
%MED_DOUGH%	
%MED_foam%	
%MED_fuel%	
%MED_GLASS%	
%MED_HOOK%	
%MED_HYPOPLAST%	
%MED_JOHNSON_COOK%	specifier for the Johnson Cook Stress flow stress model
%MED_JOHNSON_COOK_PROJ%	
%MED_LIQUID_FILM%	
%MED_none%	
%MED_PUR%	
%MED_REDLICH_KWONG%	
%MED_VFT%	
%MED_WATER%	
%MED_YIELDSTRESS%	
%MED_YIELDSTRESS_PROJ%	
%MEM_STATISTICS_ALLOC%	parameter for the real()-function (equation parser)
%MEM_STATISTICS_AVAIL%	parameter for the real()-function (equation parser)
%MEMORIZE_AdditionalFunctionManipulation%	
%MEMORIZE_Cycle%	
%MEMORIZE_DeletePoint%	
%MEMORIZE_KeepPoint%	
%MEMORIZEDelete_NbParticles%	parameter for the real()-function (equation parser)
%MEMORIZEKeep_NbParticles%	parameter for the real()-function (equation parser)
%MINIMUM_BE%	

%MINIMUM_BENP%	
%MINIMUM_BND%	
%MINIMUM_FS%	
%MINIMUM_INT%	
%MONITOR_NbParticles%	parameter for the real()-function (equation parser)
%MONITORPOINTS_CREATION_AtBoundary%	
%MONITORPOINTS_CREATION_Inside%	
%MONITORPOINTS_CREATION_IrreducibleFPMpoint%	
%MONITORPOINTS_CREATION_PenetrationOfBlindAndEmptyBoundary%	
%MOVE_bogen%	
%MOVE_concat%	
%MOVE_ElasticBeam%	
%MOVE_foam%	
%MOVE_InvokeDataCaching%	
%MOVE_MassSpringDashpot%	
%MOVE_position%	
%MOVE_ProjectionOfMovementOfAnotherPart%	
%MOVE_ReducedModel%	
%MOVE_rigid%	
%MOVE_rigid_noinertia%	
%MOVE_rotation%	
%MOVE_TowardsPhaseBoundary%	
%MOVE_TranslationRotation%	
%MOVE_velocity%	
%MOVE_vertuschka%	
%MOVE_VirtualRotation%	
%MPI_NbProcesses%	parameter for the real()-function (equation parser)
%NumberTimeStepsExecuted%	parameter for the real()-function (equation parser)
%OMP_NbProcesses%	parameter for the real()-function (equation parser)
%OR%	

%ORGANIZE_ActivationDueToLackOfFreeSurface%	possible value for %ind_Organize%
%ORGANIZE_CandidateForAtivation%	possible value for %ind_Organize%
%ORGANIZE_CandidateForFreeSurface%	possible value for %ind_Organize%, value=1
%ORGANIZE_CreatedByShallowWater%	possible value for %ind_Organize%, value=10
%ORGANIZE_CreatedByTouchDownOfFreeSurface%	possible value for %ind_OrganizeDTB%, value=88
%ORGANIZE_DeactivationDueToLackOfInteriorParticles%	possible value for %ind_Organize%
%ORGANIZE_ExplicitelyCheckedForActivation%	possible value for %ind_Organize%
%ORGANIZE_HasCreatedMonitorPoint%	possible value for %ind_Organize%, value=9
%ORGANIZE_HasRunThroughActivationProcedure%	possible value for %ind_Organize%
%ORGANIZE_IsInGap%	possible value for %ind_OrganizeDTB%, value=77
%ORGANIZE_IsIsolated%	possible value for %ind_Organize%, value=100
%ORGANIZE_IsNotActive%	possible value for %ind_Organize%
%ORGANIZE_MaxReduction%	possible value for %ind_Organize%
%ORGANIZE_MeanReduction%	possible value for %ind_Organize%
%ORGANIZE_MinReduction%	possible value for %ind_Organize%
%ORGANIZE_NbParticles%	parameter for the real()-function (equation parser)
%ORGANIZE_none%	possible value for %ind_Organize%, value=0
%ORGANIZE_WasCreatedNearMetaplanes%	possible value for %ind_Organize%, value=6
%ORGANIZE_WasNotConsideredForActivation%	possible value for %ind_Organize%
%ORGANIZE_WasPushedBackFromBoundary%	possible value for %ind_Organize%, value=8
%ORGANIZE_WasPushedToFreeSurface%	possible value for %ind_Organize%, value=5
%ORGANIZE_WasPushedToFreeSurface0%	possible value for %ind_Organize%, value=3
%PBE_Gaussian_OPMSP%	
%PBE_Gaussian_OPMSP_diff%	
%POINT_APPROXIMATE%	
%POINT_APPROXIMATE_ProjBNDOnly%	
%POINT_DIRECT%	
%PointCloudReduction_UseOldTimeStep%	comment/remark specifier for intergration statements
%POSTBND_ENGY%	

%POSTBND_ENGYint%	
%POSTBND_FRCE%	
%POSTBND_FRCEint%	
%POSTBND_HEAT%	
%POSTBND_HEATint%	
%POSTBND_MASS%	
%POSTBND_MASSint%	
%POSTBND_MOM%	
%POSTBND_MOMint%	
%POSTBND_VOL%	
%POSTBND_VOLint%	
%POSTVOL_ENGY%	
%POSTVOL_MASS%	
%POSTVOL_MOM%	
%POSTVOL_VOL%	
%PUBLICVALUE%	
%PUBLICVALUE_CLOCKstatistics%	
%PUBLICVALUE_CPUstatistics%	
%PUBLICVALUE_SUM%	
%PUBLICVALUE_TIME%	
%PUBLICVALUE_xValueOfBNDpoint%	parameter for the real()-function (equation parser)
%PUBLICVALUE_yValueOfBNDpoint%	parameter for the real()-function (equation parser)
%PUBLICVALUE_zValueOfBNDpoint%	parameter for the real()-function (equation parser)
%RealTimeSimulation%	parameter for the real()-function (equation parser)
%RepeatCurrentTimeStep_BasedOnReducedPointCloud%	
%RepeatCurrentTimeStep_BasedOnSamePointCloud%	
%RESTART_sequence%	
%RESTART_single%	
%SAVE_FreeUnit%	parameter for the real()-function (equation parser)
%SAVE_FreeUnit100%	parameter for the real()-function (equation parser)
%SAVE_scalar%	

%SAVE_vector%	
%SPM_CompressedRowFormat%	
%SUM_BENP%	
%SUMMATION_BND%	
%SUMMATION_INT%	
%SurfaceTriangulation_NbStencil%	parameter for the real()-function (equation parser)
%TIME_InitTime%	parameter for the real()-function (equation parser)
%TIME_StartTime%	parameter for the real()-function (equation parser)
%TIME_StepStartTime%	parameter for the real()-function (equation parser)
%TIME_StepWallTime%	parameter for the real()-function (equation parser)
%TIME_WallTime%	parameter for the real()-function (equation parser)
%TOUCH_always%	
%TOUCH_geometrical%	
%TOUCH_liquid%	
%TOUCH_never%	
%TOUCH_reflection%	
%TOUCH_solid%	
%VMEM_STATISTICS_ALLOC%	parameter for the real()-function (equation parser)
%VMEM_STATISTICS_AVAIL%	parameter for the real()-function (equation parser)

MESHFREE · __Constants__ · %BNDSLIP_ReprojectedAfterPassingOpenEdge%

5.56. %BNDSLIP_ReprojectedAfterPassingOpenEdge%

mark state of slip movement of MESHFREE points along boundary

the value of the slip state can be found in %ind_SlipState%

MESHFREE · __Constants__ · %BNDSLIP_TearOffAtOpenEdge%

5.57. %BNDSLIP_TearOffAtOpenEdge%

mark state of slip movement of MESHFREE points along boundary

the value of the slip state can be found in %ind_SlipState%

MESHFREE · __Constants__ · %BNDSLIP_TearOffAtRegularEdge%

5.58. %BNDSLIP_TearOffAtRegularEdge%

mark state of slip movement of MESHFREE points along boundary

the value of the slip state can be found in %ind_SlipState%

MESHFREE · __Constants__ · %BND_count_BE%

5.97. %BND_count_BE%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %BND_count_NP%

5.98. %BND_count_NP%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %BUBBLE_EQN_TruePressure%

5.139. %BUBBLE_EQN_TruePressure%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %CLOCK_STATISTICS_FLIQUID%

5.161. %CLOCK_STATISTICS_FLIQUID%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %CLOCK_STATISTICS_ORGANIZE%

5.162. %CLOCK_STATISTICS_ORGANIZE%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %CLOCK_STATISTICS_TOTAL_FLIQUID%

5.163. %CLOCK_STATISTICS_TOTAL_FLIQUID%

parameter for the real()-function (equation parser)

see real()

5.164. %CLOCK_STATISTICS_TOTAL_ORGANIZE%

parameter for the real()-function (equation parser)

see real()

5.165. %CLOCK_STATISTICS_TOTAL_SAMG%

parameter for the real()-function (equation parser)

see real()

5.192. %CPU_STATISTICS_FLIQUID%

parameter for the real()-function (equation parser)

see real()

5.193. %CPU_STATISTICS_ORGANIZE%

parameter for the real()-function (equation parser)

see real()

5.194. %CPU_STATISTICS_TOTAL_FLIQUID%

parameter for the real()-function (equation parser)

see real()

5.195. %CPU_STATISTICS_TOTAL_ORGANIZE%

parameter for the real()-function (equation parser)

see real()

5.216. %DropletSource_doNotCreateDropletsOutside%

optional parameter for the DropletSource

see DropletSource

5.217. %DropletSource_provideCounter%

parameter for the real()-function (equation parser)

see real()

5.218. %DropletSource_provideCurrentVolume%

parameter for the real()-function (equation parser)

see real()

5.219. %DropletSource_provideTargetVolume%

parameter for the real()-function (equation parser)

see real()

5.229. %EQN_Proj_ALL%

projection of a MESHFREE-entity from a different chamber using all types of points (interior and boundary)

```
[ ... projY(iChamber, %ind_Entity%, %EQN_Proj_ALL% ) ... ]
```

The projection of the MESHFREE -entity **%ind_Entity%** from the chamber with index **iChamber** is done by a smooth, least-squares approximation using all types of points, i.e. interior and boundary points.

5.230. %EQN_Proj_BND%

projection of a MESHFREE-entity from a different chamber using only boundary points

```
[ ... projY(iChamber, %ind_Entity%, %EQN_Proj_BND% ) ... ]
```

The projection of the MESHFREE -entity **%ind_Entity%** from the chamber with index **iChamber** is done by a smooth, least-squares approximation using only boundary points.

5.231. %EQN_Proj_INT%

projection of a MESHFREE-entity from a different chamber using only interior points

```
[ ... projY(iChamber, %ind_Entity%, %EQN_Proj_INT% ) ... ]
```

The projection of the MESHFREE -entity **%ind_Entity%** from the chamber with index **iChamber** is done by a smooth, least-squares approximation using only interior points.

MESHFREE · __Constants__ · %EQN_nbsum_filtered%

5.234. %EQN_nbsum_filtered%

Select filtered list

See nbsum()

MESHFREE · __Constants__ · %EQN_nbsum_nonfiltered%

5.235. %EQN_nbsum_nonfiltered%

Select non-filtered list

See nbsum()

MESHFREE · __Constants__ · %ElapsedTimeIntegrationCycle%

5.245. %ElapsedTimeIntegrationCycle%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %ElapsedTimePointOrganization%

5.246. %ElapsedTimePointOrganization%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %FLIQUID_NbParticles%

5.249. %FLIQUID_NbParticles%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %FPM_KineticEnergy%

5.250. %FPM_KineticEnergy%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %FPM_KineticEnergy_Defect_O2%

5.251. %FPM_KineticEnergy_Defect_O2%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %FPM_KineticEnergy_Defect_gradPv%

5.252. %FPM_KineticEnergy_Defect_gradPv%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %FPM_KineticEnergy_Defect_rhogDv%

5.253. %FPM_KineticEnergy_Defect_rhogDv%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %FPM_KineticEnergy_DifferenceInOrganize%

5.254. %FPM_KineticEnergy_DifferenceInOrganize%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %FPM_KineticEnergy_DifferenceInOrganize2%

5.255. %FPM_KineticEnergy_DifferenceInOrganize2%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %FPM_KineticEnergy_DifferenceInTimeStep%

5.256. %FPM_KineticEnergy_DifferenceInTimeStep%

parameter for the real()-function (equation parser)

see real()

5.257. %FPM_RepMass_CreatedByDropletSource%

parameter for the real()-function (equation parser)

see real()

5.258. %FPM_RepMass_CreatedByInflowOutflow%

parameter for the real()-function (equation parser)

see real()

5.259. %FPM_RepMass_DeletedAtMetaplanes%

parameter for the real()-function (equation parser)

see real()

5.260. %FPM_VOLUME_ACTUAL%

parameter for the real()-function (equation parser)

see real()

5.261. %FPM_VOLUME_DeletedAtMetaplanes%

parameter for the real()-function (equation parser)

see real()

5.262. %FPM_VOLUME_TARGET%

parameter for the real()-function (equation parser)

see real()

5.263. %GASDYN_Mass%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %GASDYN_MassAnalytical%

5.264. %GASDYN_MassAnalytical%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %GASDYN_MassCorrection%

5.265. %GASDYN_MassCorrection%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %GASDYN_TotalEnergy%

5.266. %GASDYN_TotalEnergy%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %GASDYN_TotalEnergyAnalytical%

5.267. %GASDYN_TotalEnergyAnalytical%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %GASDYN_TotalEnergyCorrection%

5.268. %GASDYN_TotalEnergyCorrection%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %INTEGRATION_Header%

5.308. %INTEGRATION_Header%

header information identifier for INTEGRATION

See HeaderInfoOrComments .

MESHFREE · __Constants__ · %INTEGRATION_Percentile%

5.311. %INTEGRATION_Percentile%

restrict intergration/min/max to a selected percentile-subset regarding a given function

See INTEGRATION

MESHFREE · __Constants__ · %INTEGRATION_Values(1)%

5.315. %INTEGRATION_Values(1)%

deprecated

instead, use integ()

MESHFREE · __Constants__ · %INTEGRATION_Values(2)%

5.316. %INTEGRATION_Values(2)%

deprecated

instead, use integ()

MESHFREE · __Constants__ · %INTEGRATION_Values(3)%

5.317. %INTEGRATION_Values(3)%

deprecated

instead, use integ()

MESHFREE · __Constants__ · %INTEGRATION_Values(4)%

5.318. %INTEGRATION_Values(4)%

deprecated

instead, use integ()

MESHFREE · __Constants__ · %INTEGRATION_Values(5)%

5.319. %INTEGRATION_Values(5)%

deprecated

instead, use integ()

MESHFREE · __Constants__ · %MED_JOHNSON_COOK%

5.333. %MED_JOHNSON_COOK%

specifier for the Johnson Cook Stress flow stress model

This constant can be used to prescribe Johnson-Cook elasticity behavior in terms of the shear modulus.
See mue .

5.346. %MEMORIZEDelete_NbParticles%

parameter for the real()-function (equation parser)

see real()

5.347. %MEMORIZEKeep_NbParticles%

parameter for the real()-function (equation parser)

see real()

5.352. %MEM_STATISTICS_ALLOC%

parameter for the real()-function (equation parser)

see real()

5.353. %MEM_STATISTICS_AVAIL%

parameter for the real()-function (equation parser)

see real()

5.363. %MONITOR_NbParticles%

parameter for the real()-function (equation parser)

see real()

5.381. %MPI_NbProcesses%

parameter for the real()-function (equation parser)

see real()

5.382. %NumberTimeStepsExecuted%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %OMP_NbProcesses%

5.383. %OMP_NbProcesses%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %ORGANIZE_CandidateForFreeSurface%

5.387. %ORGANIZE_CandidateForFreeSurface%

possible value for %ind_Organize%, value=1

for debugging only

MESHFREE · __Constants__ · %ORGANIZE_CreatedByShallowWater%

5.388. %ORGANIZE_CreatedByShallowWater%

possible value for %ind_Organize%, value=10

for debugging only

MESHFREE · __Constants__ · %ORGANIZE_CreatedByTouchDownOfFreeSurface%

5.389. %ORGANIZE_CreatedByTouchDownOfFreeSurface%

possible value for %ind_OrganizeDTB%, value=88

%ind_Organize% carries this flag, if MESHFREE point is at regular boundary but was created there by touch down of a free surface point

MESHFREE · __Constants__ · %ORGANIZE_HasCreatedMonitorPoint%

5.392. %ORGANIZE_HasCreatedMonitorPoint%

possible value for %ind_Organize%, value=9

for debugging only

MESHFREE · __Constants__ · %ORGANIZE_IsInGap%

5.394. %ORGANIZE_IsInGap%

possible value for %ind_OrganizeDTB%, value=77

%ind_OrganizeDTB% carries this flag, if MESHFREE point is at regular boundary but inside a gap smaller than 0.1 * SMOOTH_LENGTH

MESHFREE · __Constants__ · %ORGANIZE_IsIsolated%

5.395. %ORGANIZE_IsIsolated%

possible value for %ind_Organize%, value=100

%ind_Organize% carries this flag, if MESHFREE point does not have any relevant neighbor.

MESHFREE · __Constants__ · %ORGANIZE_NbParticles%

5.400. %ORGANIZE_NbParticles%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %ORGANIZE_WasCreatedNearMetaplanes%

5.401. %ORGANIZE_WasCreatedNearMetaplanes%

possible value for %ind_Organize%, value=6

for debugging only

MESHFREE · __Constants__ · %ORGANIZE_WasPushedBackFromBoundary%

5.403. %ORGANIZE_WasPushedBackFromBoundary%

possible value for %ind_Organize%, value=8

%ind_Organize% will carry this flag, if the point was pushed back from boundary. This will happen if the TOUCH flag of the boundary is set to %TOUCH_reflection% .

MESHFREE · __Constants__ · %ORGANIZE_WasPushedToFreeSurface%

5.404. %ORGANIZE_WasPushedToFreeSurface%

possible value for %ind_Organize%, value=5

for debugging only

MESHFREE · __Constants__ · %ORGANIZE_WasPushedToFreeSurface0%

5.405. %ORGANIZE_WasPushedToFreeSurface0%

possible value for %ind_Organize%, value=3

for debugging only

MESHFREE · __Constants__ · %ORGANIZE_none%

5.406. %ORGANIZE_none%

possible value for %ind_Organize%, value=0

%ind_Organize% will carry this flag, if no special organization procedure applied for this MESHFREE point

MESHFREE · __Constants__ · %PUBLICVALUE_xValueOfBNDpoint%

5.433. %PUBLICVALUE_xValueOfBNDpoint%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %PUBLICVALUE_yValueOfBNDpoint%

5.434. %PUBLICVALUE_yValueOfBNDpoint%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %PUBLICVALUE_zValueOfBNDpoint%

5.435. %PUBLICVALUE_zValueOfBNDpoint%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %RealTimeSimulation%

5.439. %RealTimeSimulation%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %SAVE_FreeUnit%

5.442. %SAVE_FreeUnit%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %SAVE_FreeUnit100%

5.443. %SAVE_FreeUnit100%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %SurfaceTriangulation_NbStencil%

5.450. %SurfaceTriangulation_NbStencil%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %TIME_InitTime%

5.451. %TIME_InitTime%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %TIME_StartTime%

5.452. %TIME_StartTime%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %TIME_StepStartTime%

5.453. %TIME_StepStartTime%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %TIME_StepWallTime%

5.454. %TIME_StepWallTime%

parameter for the real()-function (equation parser)

see real()

MESHFREE · __Constants__ · %TIME_WallTime%

5.455. %TIME_WallTime%

parameter for the real()-function (equation parser)

see real()

5.462. %VMEM_STATISTICS_ALLOC%

parameter for the real()-function (equation parser)

see real()

5.463. %VMEM_STATISTICS_AVAIL%

parameter for the real()-function (equation parser)

see real()

6. RunTimeTools

tools regarding the run time

Current options:

- ComputationalSteering : communication with a running MESHFREE simulation
- TIMECHECK : measurement of the performance of a running MESHFREE simulation

List of members:

ComputationalSteering	communication with running MESHFREE-job
TIMECHECK	measure performance (simulation time) for different tasks of MESHFREE

6.1. ComputationalSteering

communication with running MESHFREE-job

Write a command into the file with the name SIGNAL in the project folder, i.e.
in the folder where the input files USER_common_variables.dat and common_variables.dat
are located.

Under linux, the most easy way is to use the echo command, for instance
linux> echo quit > SIGNAL

One can also load SIGNAL into a regular editor.

Note: After reading of the SIGNAL file, MESHFREE will completely erase its contents.
Some editors might give an automatic warning, that the file has changed on disc.

The computational steering can run in two different modes, which you can switch between using the common variable SIGNAL_LaunchComputationalSteering :

-

SIGNAL_LaunchComputationalSteering = true
parallel, see ParallelReadingOfSignalFile

- SIGNAL_LaunchComputationalSteering = false
sequential, see SequentialReadingOfSignalFile (default)

List of members:	
ParallelReadingOfSignalFile	communication with running MESHFREE-job by separate (parallel) thread
SequentialReadingOfSignalFile	communication with running MESHFREE-job by sequential reading of SIGNAL-file
step-by-step-execution	execute MESHFREE step by step

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile

6.1.1. ParallelReadingOfSignalFile

communication with running MESHFREE-job by separate (parallel) thread

MESHFREE starts a separate thread, that mostly sleeps, but once in a second, it checks if the contents of the SIGNAL-file has changed. If changed, it triggers the appropriate actions by signal handlers.

Advantage of this is, that MESHFREE will not have to pause in order to interprete the SIGNAL-file.

The parallel processing of the SIGNAL-file is invoked ONLY in the common_variables.dat by the option
SIGNAL_LaunchComputationalSteering = true

List of members:	
batchmode	pause to MESHFREE execution, allow MESHFREE batch commands that modify the current state of the point cloud
batchmode_off	leave the batchmode
batchmode_on	enter the batchmode
bi	stop the currently running iteration of sparse linear systems
break_iteration	stop the currently running iteration of sparse linear systems
checkpoint	write a MESHFREE resume file after the end of the current time step and terminate MESHFREE (parallel reading of signal file)
pause	let MESHFREE sleep until the next pause command is launched
pause_off	continue MESHFREE execution after pause_on command was given
pause_on	let MESHFREE sleep until the pause_off command is launched
plot	save computational results after the end of the current time step (parallel reading of signal file)
qualitycheck	force a quality check of the MESHFREE point cloud after the next organization step (parallel reading of signal file)
quit	quit MESHFREE execution after the current time cycle (parallel reading of signal file)
reread_all	reloads both USER_common_variables.dat and common_variables.dat (parallel reading of signal file)
reread_cv	reload common_variables.dat (parallel reading of the signal file)
reread_Ucv	reload USER_common_variables.dat (parallel reading of signal file)
reset_Vanalytical	resets the analytical volume of each chamber to the current values
save	write a MESHFREE restart file after the end of the current time step (parallel reading of signal file)
set_OMP_threads	redefines the number of OMP threads to be used (parallel reading of signal file)
step=NNN	execute a given number of steps, if in the step-by-step execution mode (parallel reading of signal file)
step	execute the next step, if in the step-by-step execution mode (parallel reading of signal file)
stepbystep=false	switch off step-by-step execution modus of MESHFREE (parallel reading of signal file)
stepbystep=true	switch on step-by-step execution modus of MESHFREE (parallel reading of signal file)
time_check	write out a detailed time check listing
time_check_sum	write out a sum-up conclusion of the time check
write_cv	write the complete set of numerical parameters to file (parallel reading of signal file)

batchmode

pause to MESHFREE execution, allow MESHFREE batch commands that modify the current state of the point cloud

This feature might help in debugging or further developing MESHFREE .

List of members:

deleteParticlesOn{}	delete MESHFREE points by an arithmetic criterion
distanceToBND{}	recompute the distance of each point with respect to the boundary
echo{}	simple test writeout in order to check the response of MESHFREE
evaluateEquation{}	evaluate equation pointwise
include{}	read in more geometry elements
ORGANIZE_DevelopperCheck{}	call the MESHFREE subroutine ORGANIZE_DevelopperCheck
organize_points{}	Execute the complete point organization subroutine of MESHFREE
plot{}	write result output
POINTCLOUD_SetInitialPointToBE{}	call the MESHFREE subroutine POINTCLOUD_SetInitialPointToBE
propagateFunction{}	propagate a function with restricted gradient
quickview{}	produce a quickview image
recomputeMPIbisection{}	recompute MPI bisection on the spot
recomputeSearchTree{}	Recompute the point search tree
removePartsOfBEbyAlias{}	remove boundary parts defined by alias name(s)
sort_BE_into_boxes{}	reconstruct the search tree for boundary elements

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · batchmode · ORGANIZE_DevelopperCheck{}

ORGANIZE_DevelopperCheck{}

call the MESHFREE subroutine ORGANIZE_DevelopperCheck

```
ORGANIZE_DevelopperCheck{}
```

Execute a call to the MESHFREE -subroutine ORGANIZE_DevelopperCheck.
Purpose: clearly debugging

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · batchmode · POINTCLOUD_SetInitialPointToBE{}

POINTCLOUD_SetInitialPointToBE{}

call the MESHFREE subroutine POINTCLOUD_SetInitialPointToBE

```
POINTCLOUD_SetInitialPointToBE{}
```

Place new points at empty boundary elements.
Purpose: initialize the filling procedure of new boundary elements, imported by the include{ } command.

deleteParticlesOn{}

delete MESHFREE points by an arithmetic criterion

```
deleteParticlesOn{ [BodyOfConditionEquation] }
```

A MESHFREE point is deleted if the evaluation of the [BodyOfConditionEquation] is positive, for example deleteParticlesOn{ [Y %ind_T% -300] } would delete all MESHFREE points whose temperature is bigger than 300.

distanceToBND{}

recompute the distance of each point with respect to the boundary

```
distanceToBND{}
```


evaluateEquation{}

evaluate equation pointwise

```
evaluateEquation{ %ind_f% , [EquationBody] }
```

the given equation [EquationBody] is evaluated pointwise, the result is copied into Y%ind_f% and can therefore be reused
EquationBody: is of the classical form as described in Equations
%ind_f% : is the function index where to copy the result in the Y-array

There is the following option

```
evaluateEquation{ 0 , [EquationBody] }
```

if 0 or a negative number is given instead of %ind_f%, then the equation is NOT executed pointwise, but only once.
The result is written directly into the .signallog file.

include{}

read in more geometry elements

```
include{ FileName.xyz}, scale{...}, offset{...}
```

This statement is of the type of the typical geometry file include statements, see also the section of BoundaryElements

organize_points{}

Execute the complete point organization subroutine of MESHFREE

```
organize_points{}
```

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · batchmode · plot{}

plot{}

write result output

```
plot{}
```

Immediately produce a result output according to the SAVE_format statement given in USER_common_variables
Purpose: provide a way to (step-by-step) check the results of the batchmode operations

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · batchmode · propagateFunction{}

propagateFunction{}

propagate a function with restricted gradient

```
propagateFunction{ AllowedGradient, %ind_f% , %ind_attched_1% , %ind_attched_2%, ... }
```

The function propagation allow a certain gradient only, i.e. after function propagation,

$$f_j <= f_i + \|\boldsymbol{x}_j - \boldsymbol{x}_i\| \cdot \mathrm{AllowedGradient}$$

for neighbor points with the indices i and j.

%ind_attched_n% are optional and can be used as color function in order to sketch the graph of function distribution.

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · batchmode · quickview{}

quickview{}

produce a quickview image

```
quickview{}
```

Immediately produce a quickview of the present state.
Purpose: For quick checks of all the present batchmode operations, most of
all in order to check the result of
- evaluateEquation{}
- distanceToBND{}
- propagateFunction{}

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · batchmode · recomputeMPIbisection{}

recomputeMPIbisection{}

recompute MPI bisection on the spot

This command redefines the MPI-bisection (i.e. domain decomposition), redistributes the MESHFREE points and re-

establishes the
communicatin list.

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · batchmode · recomputeSearchTree{}

recomputeSearchTree{}

Recompute the point search tree

```
recomputeSearchTree{}
```

Recompute the point search tree according to the value UseBoxSystemVersion given in common_variables .

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · batchmode · removePartsOfBEbyAlias{}

removePartsOfBEbyAlias{}

remove boundary parts defined by alias name(s)

```
removePartsOfBEbyAlias{"AliasToBeRemoved"}
```

All of the boundary elements that are originally tagged with the given alias, are removed.

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · batchmode · sort_BE_into_boxes{}

sort_BE_into_boxes{}

reconstruct the search tree for boundary elements

```
sort_BE_into_boxes{}
```

This call recomputres the bisection search tree for the boundary elements. Most suitable for
- debugging reasons or
- reading in additional geometry during execution

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · batchmode_off

batchmode_off

leave the batchmode

see MESHFREE::RunTimeTools::ComputationalSteering::batchmode .

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · batchmode_on

batchmode_on

enter the batchmode

see MESHFREE::RunTimeTools::ComputationalSteering::batchmode .

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · bi

bi

stop the currently running iteration of sparse linear systems

same as break_iteration . See there.

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · break_iteration

break_iteration

stop the currently running iteration of sparse linear systems

ONLY FOR SCIENTIFIC REASONS, DEBUGGING, TESTING, or if you really know what you do.
Stops the currently running BiCGstab iteration of sparse linear systems, before convergence
is reached. Shortens simulation time in debugging/testing cases etc.

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · checkpoint

checkpoint

write a MESHFREE resume file after the end of the current time step and terminate MESHFREE (parallel reading of signal file)

This writes out a resume file (named .resume) which is like a regular restart file, but with special meaning. The file is not located in the result directory specified by SAVE_path like the regular restart files. Instead, the resume file is written to the directory where MESHFREE is executed.

If at the start of MESHFREE , a resume file is available in the current directory it will be used. MESHFREE will not start from the beginning and restart numbers are ignored as well.

Alternatively, a file named .checkpoint can be written to the current working directory of MESHFREE .

A third option of triggering writing of a resume file, even dependent on the simulation result, is given via the EVENT %EVENT_WriteResume% .

Note: Do not run several instances of MESHFREE from the same directory when using the checkpoint/resume feature!

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · pause

pause

let MESHFREE sleep until the next pause command is launched

interupts but not stops the MESHFREE execution. If the pause command is launched a second time,
MESHFREE continues execution.

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · pause_off

pause_off

continue MESHFREE execution after pause_on command was given

see also pause and pause_on .

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · pause_on

pause_on

let MESHFREE sleep until the pause_off command is launched

MESHFREE sleep : pause_on
MESHFREE continue: pause_off

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · plot

plot

save computational results after the end of the current time step (parallel reading of signal file)

MESHFREE pretends as if it was a regularly triggered output and lines it in correct order into the output files.

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · qualitycheck

qualitycheck

force a quality check of the MESHFREE point cloud after the next organization step (parallel reading of signal file)

MESHFREE performs a quality check and puts down the results into the file QUALITYCHECK.case in the result-folder SAVE_path .

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · reread_Ucv

reread_Ucv

reload USER_common_variables.dat (parallel reading of signal file)

After having effected changes in the input file (for example modification of the boundary conditions,
material properties, smoothing length, ...), MESHFREE will reload USER_common_variables.dat
on the fly, i.e. without stopping the program. The changes made will take effect with the next timestep executed.

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · reread_all

reread_all

reloads both USER_common_variables.dat and common_variables.dat (parallel reading of signal file)

Does both actions at the same time:
reread_Ucv
reread_cv
See there.

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · reread_cv

reread_cv

reload common_variables.dat (parallel reading of the signal file)

After having effected changes in the input file (for example modification of numerical parameters),
MESHFREE will reload common_variables.dat on the fly, i.e. without stopping the program.
The changes made will take effect with the next timestep executed.

---

reset_Vanalytical

resets the analytical volume of each chamber to the current values

ONLY FOR TESTING AND DEBUGGING, or if you know what you do.
By default, MESHFREE computes a mass/volume balance for each chamber individually.
By the initial volume and the time integral of all inflows and outflows, MESHFREE is always up to date
about the current mass/volume, that should (theoretically) be present in a chamber.
By this command, this theoretical value is reset to the actually measured mass/volume in the chamber.
That makes sense for example in the following situation: remove a considerable number of points
during a batchmode session, see MESHFREE::RunTimeTools::ComputationalSteering::batchmode .

---

save

write a MESHFREE restart file after the end of the current time step (parallel reading of signal file)

MESHFREE pretends as if it was a regularly triggered restart file. Especially if %RESTART_sequence% is given in the Restart settings,
the new restart file obtains the next ordinal number in the sequence of restart files.

Note: In case of using %RESTART_sequence% to define the RestartStepSize , the user can limit the number of kept restart
files triggered by SIGNAL similarly to the number of kept restart files triggered by standard. See %RESTART_sequence% for details.

---

set_OMP_threads

redefines the number of OMP threads to be used (parallel reading of signal file)

```
set_OMP_threads=4
```

Set the number of OMP threads to 4. If the environment variable OMP_NUM_THREADS is defined, we have
set_OMP_threads = min(set_OMP_threads , OMP_NUM_THREADS)

---

time_check

write out a detailed time check listing

The time check listing can be given only if COMP_TimeCheck = 1 or COMP_TimeCheck = 2 is
given in common_variables.dat. The listing is appended to the .signallog file in the MESHFREE project folder.

---

time_check_sum

time_check_sum

write out a sum-up conclusion of the time check

The sum-up time check can be given only if COMP_TimeCheck = 1 or COMP_TimeCheck = 2 is given in common_variables.dat. The listing is appended to the .signallog file in the MESHFREE project folder.

MESHFREE · RunTimeTools · ComputationalSteering · ParallelReadingOfSignalFile · write_cv

write_cv

write the complete set of numerical parameters to file (parallel reading of signal file)

After changing common_variables.dat, 'write_cv' will write the complete set of numerical parameters in the file .common_variables_CompleteConfiguration.dat, to be found in the actual result-folder SAVE_path .

MESHFREE · RunTimeTools · ComputationalSteering · SequentialReadingOfSignalFile

6.1.2. SequentialReadingOfSignalFile

communication with running MESHFREE-job by sequential reading of SIGNAL-file

At the beginning of each time step, MESHFREE reads the contents of the SIGNAL-file situated in the project folder and interprets the given commands. This is the sequential way as all MESHFREE -business has to stop for a moment while the program reads and interprets the SIGNAL-file.

Note: The sequential processing is the default! ParallelReadingOfSignalFile can be launched only in common_variables.dat by the option

SIGNAL_LaunchComputationalSteering = true

List of members:

checkpoint	write a MESHFREE resume file after the end of the current time step and terminate MESHFREE (sequential reading of signal file)
plot	save computational results after the end of the current time step (sequential reading of signal file)
qualitycheck	force a quality check of the MESHFREE point cloud after the next organization step (sequential reading of signal file)
quit	quit MESHFREE execution after the current time cycle (sequential reading of signal file)
reread_all	reloads both USER_common_variables.dat and common_variables.dat (sequential reading of signal file)
reread_cv	reload common_variables.dat (sequential reading of the signal file)
reread_Ucv	reload USER_common_variables.dat (sequential reading of signal file)
save	write a MESHFREE restart file after the end of the current time step (sequential reading of signal file)
set_OMP_threads	redefines the number of OMP threads to be used (sequential reading of signal file)
step=NNN	execute a given number of steps, if in the step-by-step execution mode (sequential reading of the signal file)
step	execute the next step, if in the step-by-step execution mode (sequential reading of the signal file)
stepbystep=false	switch off step-by-step execution modus of MESHFREE (sequential reading of the signal file)
stepbystep=true	switch on step-by-step execution modus of MESHFREE (sequential reading of signal file)
write_cv	write the complete set of numerical parameters to file (sequential reading of signal file)

MESHFREE · RunTimeTools · ComputationalSteering · SequentialReadingOfSignalFile · checkpoint

checkpoint

write a MESHFREE resume file after the end of the current time step and terminate MESHFREE (sequential reading of signal file)

See checkpoint for details.

MESHFREE · RunTimeTools · ComputationalSteering · SequentialReadingOfSignalFile · plot

plot

save computational results after the end of the current time step (sequential reading of signal file)

See plot for details.

MESHFREE · RunTimeTools · ComputationalSteering · SequentialReadingOfSignalFile · qualitycheck

qualitycheck

force a quality check of the MESHFREE point cloud after the next organization step (sequential reading of signal file)

See qualitycheck for details.

reread_Ucv

reload USER_common_variables.dat (sequential reading of signal file)

See reread_Ucv for details.

reread_all

reloads both USER_common_variables.dat and common_variables.dat (sequential reading of signal file)

See reread_all for details.

reread_cv

reload common_variables.dat (sequential reading of the signal file)

See reread_cv for details.

save

write a MESHFREE restart file after the end of the current time step (sequential reading of signal file)

See save for details.

set_OMP_threads

redefines the number of OMP threads to be used (sequential reading of signal file)

See set_OMP_threads for details.

write_cv

write the complete set of numerical parameters to file (sequential reading of signal file)

See write_cv for details.

6.1.3. step-by-step-execution

execute MESHFREE step by step

MESHFREE has a number of break points. MESHFREE
- stops at each beakpoint
- write a quickview file

this feature helps to debug MESHFREE -applications in their startup phase.
Especially, it is easy to screen the point filling process.

Currently, the breakpoints are
- after each point-cloud-filling loop for boundary points
- after each point-cloud-filling loop for interior points

Control the step-by-step-execution functionality by the SIGNAL file and by command-line arguments:
- **--executeStepByStep** -> in order to trigger the step-by-step-execution directly at startup, start MESHFREE with this command-line option .
- **stepbystep=true** -> in order to trigger the step-by-step-execution during execution, write this command into the SIGNAL-file .
- **plot** -> in order to trigger writing a case-file for better postprocessing, write this command into the SIGNAL-file .
- **step** -> in order to trigger one step, write this command into the SIGNAL-file .
- **step=NNN** -> run NNN step-cycles (same as NNN-times triggering the step-signal) .
- **stepbystep=false** -> in order to switch off the step-by-step-execution modus, write this command into the SIGNAL-file .

6.2. TIMECHECK

measure performance (simulation time) for different tasks of MESHFREE

Switch on the performance analysis by COMP_TimeCheck .
MESHFREE launches several stop watches. The stop watches have names (see NamesOfStopWatches), which mark the task whose performance is to be observed.
The stop watches are hierarchical and nested.
The hierarchy level is controlled by TIMECHECK_Level .

List of members:

NamesOfStopWatches	currently implemented stop watches

6.2.1. NamesOfStopWatches

currently implemented stop watches

The following stop watches are currently implemented

ADMIN_TIME_INTEG -> time for one full time cycle, excluding the saving operations
ADMIN_TIME_INTEG.ORGANIZE -> time for the MESHFREE point organization
ADMIN_TIME_INTEG.ORGANIZE.TimeStepManagement
ADMIN_TIME_INTEG.ORGANIZE.BE_Movement
ADMIN_TIME_INTEG.ORGANIZE.ComputSteering
ADMIN_TIME_INTEG.ORGANIZE.PREPARATION
ADMIN_TIME_INTEG.ORGANIZE.PREPARATION2
ADMIN_TIME_INTEG.ORGANIZE.PREPARATION3

ADMIN_TIME_INTEG.ORGANIZE.PREPARATION4
ADMIN_TIME_INTEG.ORGANIZE.GapDetection
ADMIN_TIME_INTEG.ORGANIZE.BISE_REDISTRIBUTION
ADMIN_TIME_INTEG.ORGANIZE.BE
ADMIN_TIME_INTEG.ORGANIZE.BE.ComputeH
ADMIN_TIME_INTEG.ORGANIZE.BE.SortBE
ADMIN_TIME_INTEG.ORGANIZE.BE.SortBE.PREP
ADMIN_TIME_INTEG.ORGANIZE.BE.SortBE.SORT
ADMIN_TIME_INTEG.ORGANIZE.BE.SortBE.SORT.part1
ADMIN_TIME_INTEG.ORGANIZE.BE.SortBE.SORT.part2
ADMIN_TIME_INTEG.ORGANIZE.BE.DeactBE
ADMIN_TIME_INTEG.ORGANIZE.BE.RepairBE
ADMIN_TIME_INTEG.ORGANIZE.ParticleTree
ADMIN_TIME_INTEG.ORGANIZE.EstablishCON
ADMIN_TIME_INTEG.ORGANIZE.DIST_TO_BND
ADMIN_TIME_INTEG.ORGANIZE.ACTIVATE_BND
ADMIN_TIME_INTEG.ORGANIZE.FILL_BND
ADMIN_TIME_INTEG.ORGANIZE.REMOVE_BND
ADMIN_TIME_INTEG.ORGANIZE.FILL_FREE_SURFACE
ADMIN_TIME_INTEG.ORGANIZE.FILL_INT
ADMIN_TIME_INTEG.ORGANIZE.FILL_MANIFOLD
ADMIN_TIME_INTEG.ORGANIZE.REMOVE_MANIFOLD
ADMIN_TIME_INTEG.ORGANIZE.REMOVE_INT
ADMIN_TIME_INTEG.ORGANIZE.CHECK_FREE_SURFACE
ADMIN_TIME_INTEG.ORGANIZE.COMMUNICATION
ADMIN_TIME_INTEG.ORGANIZE.COMMUNICATION.BISECT
ADMIN_TIME_INTEG.ORGANIZE.COMMUNICATION.PROCRECMP
ADMIN_TIME_INTEG.ORGANIZE.COMMUNICATION.REDISTR
ADMIN_TIME_INTEG.ORGANIZE.COMMUNICATION.COMMLIST
ADMIN_TIME_INTEG.ORGANIZE.COMMUNICATION.COMMUN
ADMIN_TIME_INTEG.ORGANIZE.COMMUNICATION.NEIGHLIST
ADMIN_TIME_INTEG.ORGANIZE.COMMUNICATION.GLOBIND
ADMIN_TIME_INTEG.ORGANIZE.COMMUNICATION.GLOBINDRED
ADMIN_TIME_INTEG.ORGANIZE.NEIGHBORLISTREDUCTION
ADMIN_TIME_INTEG.ORGANIZE.NEIGHBORLISTREDUCTION.RME -> remove multiple entries from neighbor list
ADMIN_TIME_INTEG.ORGANIZE.NEIGHBORLISTREDUCTION.CC
ADMIN_TIME_INTEG.ORGANIZE.NEIGHBORLISTREDUCTION.CC2
ADMIN_TIME_INTEG.ORGANIZE.NEIGHBORLISTREDUCTION.ONL
ADMIN_TIME_INTEG.ORGANIZE.APPROXIMATE
ADMIN_TIME_INTEG.ORGANIZE.OPPOSITE_POINTS
ADMIN_TIME_INTEG.ORGANIZE.NEIGHBORLISTS
ADMIN_TIME_INTEG.ORGANIZE.FINALIZE
ADMIN_TIME_INTEG.ORGANIZE.REMOVE_FROM_REGION
ADMIN_TIME_INTEG.TIMEINTEGRATION -> whole time integration (LIQUID , DROPLETPHASE , etc)
ADMIN_TIME_INTEG.FLIQUID -> pure numerics in incompressible solver
ADMIN_TIME_INTEG.FLIQUID.DIFF_OPERATORS
ADMIN_TIME_INTEG.FLIQUID.PREPARATION
ADMIN_TIME_INTEG.FLIQUID.PHYDROSTATIC
ADMIN_TIME_INTEG.FLIQUID.TEMPERAURE
ADMIN_TIME_INTEG.FLIQUID.SOLVE_V
ADMIN_TIME_INTEG.FLIQUID.SOLVE_V.MxVprepare
ADMIN_TIME_INTEG.FLIQUID.SOLVE_V.SolveMatrix
ADMIN_TIME_INTEG.FLIQUID.PCORRECTION
ADMIN_TIME_INTEG.FLIQUID.PDYNAMIC
ADMIN_TIME_INTEG.FLIQUID.KEPSILON
ADMIN_TIME_INTEG.FLIQUID.POSTPROCESSING
ADMIN_TIME_INTEG.SPM_BiCGstab
ADMIN_TIME_INTEG.SPM_BiCGstab_CommCheck
ADMIN_TIME_INTEG.MPIbarrier
ADMIN_TIME_INTEG.MPIbarrier2
ADMIN_TIME_INTEG.MPIcommunicate
ADMIN_TIME_INTEG.MPIreduction
'ADMIN_TIME_INTEG ___DELAUNAY___' -> collection of the computation times used for delaunay triangulation (non-hierarchical)

ADMIN_TIME_INTEG.TRANSPORT
ADMIN_TIME_INTEG.TRANSPORT.SOLVE_EXPL_STEP
ADMIN_TIME_INTEG.TRANSPORT.BIG_LOOP_divOp
ADMIN_TIME_INTEG.TRANSPORT.Establish_Diff_Ops
ADMIN_TIME_INTEG.TRANSPORT.EXPL_TIME_INTEGRATION
ADMIN_TIME_INTEG.TRANSPORT.EXPL_SUBCYCLE
ADMIN_TIME_INTEG.TRANSPORT.STAGE_LOOP
ADMIN_TIME_INTEG.ORGANIZE.PrepareNumerics
ADMIN_TIME_INTEG.ORGANIZE.Misc -> collection of miscellaneous MESHFREE point organization (e.g. DropletSource , EVENT , MEMORIZE , MONITORPOINTS , STANDBY)
ADMIN_TIME_INTEG.ORGANIZE.Misc.DropletSource
ADMIN_TIME_INTEG.ORGANIZE.Misc.EVENT
ADMIN_TIME_INTEG.ORGANIZE.Misc.MEMORIZE
ADMIN_TIME_INTEG.ORGANIZE.Misc.MONITORPOINTS
ADMIN_TIME_INTEG.ORGANIZE.Misc.OppositePoints
ADMIN_TIME_INTEG.ORGANIZE.Misc.STANDBY
ADMIN_TIME_INTEG.SAVE
ADMIN_TIME_INTEG.SAMG
ADMIN_TIME_INTEG.MANIFOLD
ADMIN_TIME_INTEG.POSTPROC -> postprocessing after all time integrations for LIQUID , GASDYN , ... are complete
ADMIN_TIME_INTEG.POSTPROC.INTEGRATION -> process all INTEGRATION statements of USER_common_variables
ADMIN_TIME_INTEG.POSTPROC.dtLocal -> collect all local time step sizes from all chambers
ADMIN_TIME_INTEG.POSTPROC.ODE -> process the ODE definitions of USER_common_variables
ADMIN_TIME_INTEG.POSTPROC.MISC -> minor activities
ADMIN_TIME_INTEG.DROPLETPHASE -> collection of computation times related to DROPLETPHASE chambers
ADMIN_TIME_INTEG.DROPLETPHASE.PrepSubcyc -> computation time spent on preparing the subcycling in F_of_t_and_Y_DROPLETPHASE
ADMIN_TIME_INTEG.DROPLETPHASE.PWCollision -> computation time spent on resolving (P)article-(W)article collisions in F_of_t_and_Y_DROPLETPHASE
ADMIN_TIME_INTEG.DROPLETPHASE.ApplyBC -> computation time spent on applying boundary conditions in F_of_t_and_Y_DROPLETPHASE
ADMIN_TIME_INTEG.DROPLETPHASE.BodyForces -> computation time spent on calculating body forces in F_of_t_and_Y_DROPLETPHASE
ADMIN_TIME_INTEG.DROPLETPHASE.UserDefVar -> computation time spent on processing user defined variables in F_of_t_and_Y_DROPLETPHASE
ADMIN_TIME_INTEG.DROPLETPHASE.LayerAcc -> computation time spent on computation for acceleration of wall layer in F_of_t_and_Y_DROPLETPHASE

7. Solvers

Overview of numerical and geometrical algorithms used in MESHFREE

- Geometry algorithms mostly focus on the point cloud management.
- Numerics algorithms focus on the partial differential equations (PDE) to be solved and the GFDM numerics to solve the given PDE.

List of members:	
Geometry	Algorithms dedicated for the point cloud management in MESHFREE
Numerics	PDE to be solved and the meshfree algorithms to solve the PDE

7.1. Geometry

Algorithms dedicated for the point cloud management in MESHFREE

We will address questions like
- nearest neighbor search
- handle/exclude critical neighbors from the neighbor lists
- find geometry clusters

List of members:	
ExcludeCriticalNeighbors	Exclude critical neighbors from the neighborlists of MESHFREE points
VoronoiTesselation	how to tesselate a given cloud of points

7.1.1. ExcludeCriticalNeighbors

Exclude critical neighbors from the neighborlists of MESHFREE points

Exclusion of critical neighbors is essential if the geometry contains thin parts (blades, ribbons, etc.) whose diameter is much smaller than the local SmoothingLength .

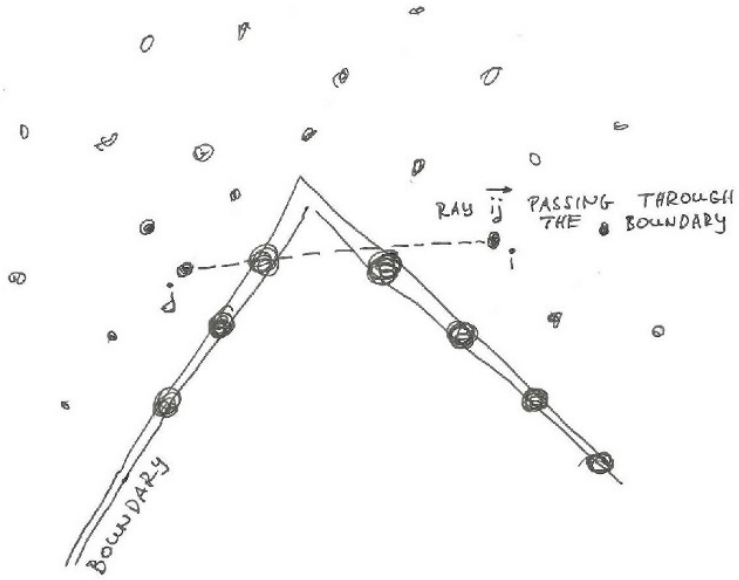
There are several ideas how to achieve this (see items below). See also NEIGHBOR_FilterMethod , this parameter controls the choice of
the selection methods chosen.

List of members:	
GeometryBased	Exclude critical neighbors based on the given, triangulated geometry
ReplugNeighbors	Replug neighbor MESHFREE points by passon-analysis
NormalBased	Exclude critical neighbors from the neighborlists of MESHFREE boundary points
PositionBased	Exclude critical neighbors from the neighborlists of MESHFREE points

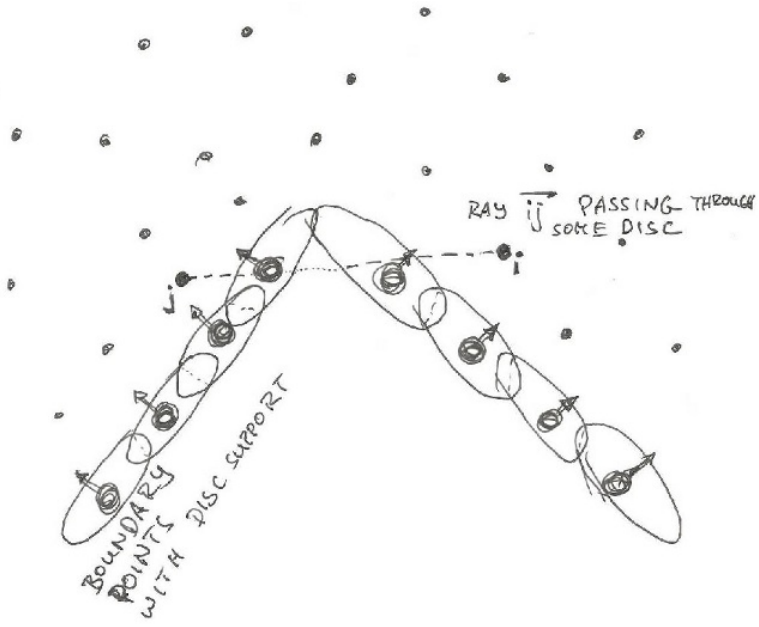
GeometryBased

Exclude critical neighbors based on the given, triangulated geometry

1.) The picture shows the exclusion algorithm based on the geometry.
IF the segment between two MESHFREE points passes through one of the given boundary triangles, then they are excluded as neighbors.
See the picture below.



2.) Due to high computational effort, the method above might come with a very fine triangular resolution of the boundary, recently this algorithm switched to an approximative version. Here, all boundary points form discs about their particular position and the given normal. The radius of the disc is 0.3*SmoothingLength.
The collection of discs forms an approximation of the given rigid boundary. See the picture below.



Two MESHFREE points are excluded as neighbors, IF their connecting segment passes through one such disc.

Both algorithms shown here might lead to the situation, that we exclude too many neighbors from each other, especially at convex boundaries.
A solution to this dilemma is given by the ReplugNeighbors algorithm.

See NEIGHBOR_FilterMethod in particular.

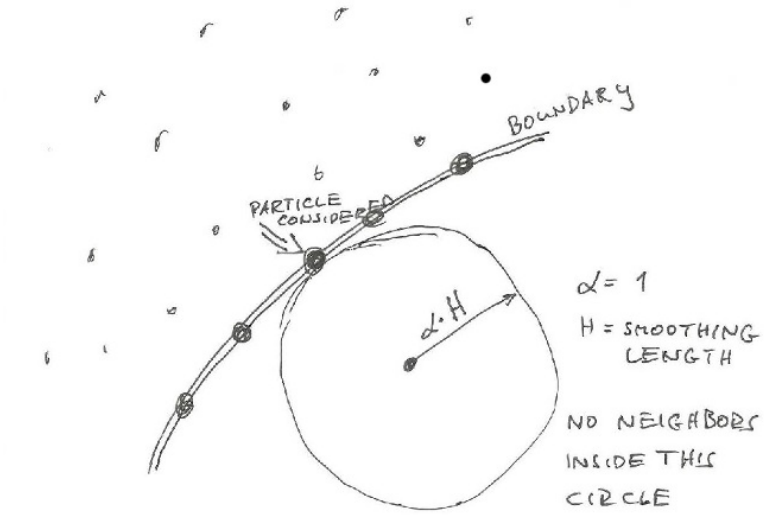
MESHFREE · Solvers · Geometry · ExcludeCriticalNeighbors · NormalBased

NormalBased

Exclude critical neighbors from the neighborlists of MESHFREE boundary points

The picture shows the most simple exclusion algorithms chosen. It is a purely algebraic constraint.

If a MESHFREE point is at the boundary, it is not allowed to see neighbors which are in a ball "behind the normal".



See NEIGHBOR_FilterMethod in particular.

MESHFREE · Solvers · Geometry · ExcludeCriticalNeighbors · PositionBased

PositionBased

Exclude critical neighbors from the neighborlists of MESHFREE points

The picture shows the exclusion algorithm based on the position, boundary distance and boundary normal of two MESHFREE points.
This constraint is purely algebraic.

Two MESHFREE points i and j are excluded from each other if they meet the following position-normal-constraint.

$$(\boldsymbol{x}_j - \boldsymbol{x}_i)^T \cdot \boldsymbol{n}_i < 0$$

OR

$$(\boldsymbol{x}_i - \boldsymbol{x}_j)^T \cdot \boldsymbol{n}_j < 0$$

See NEIGHBOR_FilterMethod in particular.

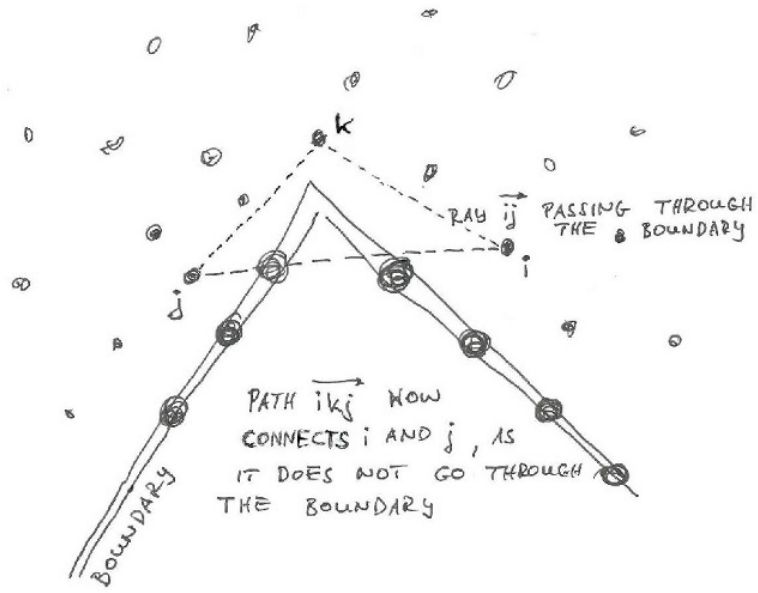
MESHFREE · Solvers · Geometry · ExcludeCriticalNeighbors · ReplugNeighbors

ReplugNeighbors

Replug neighbor MESHFREE points by passon-analysis

If a MESHFREE point was thrown out from the neighborhood list by the GeometryBased algorithm, then
it might be put back due to the following passon or connectivity idea:

After the GeometryBased algorithm is completed for all MESHFREE points, two MESHFREE points might be plugged back as neighbors
IF both find the same (third) point in their remaining neighorhood list. See the picture below.



See NEIGHBOR_FilterMethod in particular.

7.1.2. VoronoiTesselation

how to tesselate a given cloud of points

Given a set of points around the origin.
In order to produce a complete Voronoi cell, one has to execute the algorithm of SingleFace
for ever point $\boldsymbol{p}_k$ in the point cloud $k = 1, \ldots$.

Many of the points will produce void faces due to extinctng intersections. However, some of the points will produce regular faces, which will form a closed shell around the origin.
The volume of the cell is computed

$$V_{\text{cell}} = \sum_{i=1}^{N_{\text{faces}}} \frac{1}{3} A_{\text{face}}(i) \|\boldsymbol{p}_i\|$$

List of members:

SingleFace	how to tesselate a given cloud of points

SingleFace

how to tesselate a given cloud of points

Given a set of points around the origin. We would like to compute the Voronoi face produced by the point with index k with respect to the origin.
The point is called $\boldsymbol{p}$, it spans a plane with the face normal $\boldsymbol{n} = \frac{\boldsymbol{p}}{\|\boldsymbol{p}\|} = (n_x, n_y, n_z)^T$.
The Voronoi face will be in this plane. It is given by a polyeder of n_{Seg} segments, each segment uniqely associated with one point in the pointcloud.

Suppose a given Voronoi face, that is a number of connected segments in the plane spanned by $\boldsymbol{p}$. Now we add a new point with the index m to
the pointcloud, we call the point $\boldsymbol{q}_m$, and we want to know whether the plane, spanned by the new point, will cut the existing Voronoi face.

The axis formed by the planes of $\boldsymbol{p}$ and $\boldsymbol{q}_m$ is given by the point $\boldsymbol{s}_m$ and the direction vector $\boldsymbol{t}_m$.
we have

$$\boldsymbol{s}_m = \alpha \cdot \boldsymbol{p} + \beta \cdot \boldsymbol{q}_m \text{ with } \alpha = \frac{p^2 q^2 - \boldsymbol{p}^T \boldsymbol{q} q^2}{p^2 q^2 - (\boldsymbol{p}^T \boldsymbol{q})^2} \text{ and } \beta = \frac{q^2 p^2 - \boldsymbol{p}^T \boldsymbol{q} p^2}{p^2 q^2 - (\boldsymbol{p}^T \boldsymbol{q})^2}$$

The direction of the segment is

$$\boldsymbol{t}_m = \frac{\boldsymbol{p} \times \boldsymbol{q}_m}{\|\boldsymbol{p} \times \boldsymbol{q}_m\|}$$

Now we check for the given segments in the Vornoi-face, their $\boldsymbol{s}_i, \boldsymbol{t}_i$ being computed in the same way as above.

$$\boldsymbol{s}_{mi} = \boldsymbol{s}_m + \alpha_m \boldsymbol{t}_m = \boldsymbol{s}_i + \alpha_i \boldsymbol{t}_i \text{ with } \alpha_i = \frac{(\boldsymbol{s}_m - \boldsymbol{s}_i) \cdot \boldsymbol{q}_m}{\boldsymbol{t}_i \cdot \boldsymbol{q}_m} \text{ and } \alpha_m = -\frac{(\boldsymbol{s}_m - \boldsymbol{s}_i) \cdot \boldsymbol{q}_i}{\boldsymbol{t}_m \cdot \boldsymbol{q}_i}$$

If two segments intersect, then the intersection points form a startpoint for one of the two segments, and an ending point for the other one.
The decision is as follows: if $(\boldsymbol{t}_m \times \boldsymbol{t}_i) \cdot \boldsymbol{p} > 0$, then the segment m has an end point, i has a start point .

The area of the face can be computed, if the segment polyeder is closed. In this case,

$$A_{\text{face}} = \frac{1}{2} \oint_{\partial\Omega} \begin{pmatrix} n_y z - n_z y \\ n_z x - n_x z \\ n_x y - n_y x \end{pmatrix} \cdot \boldsymbol{t} \, d(\partial\Omega)$$

See also FPMDOCU_VoronoiFace.pdf

7.2. Numerics

PDE to be solved and the meshfree algorithms to solve the PDE

We differentiate between the different solvers for

- LIQUID : implicit solver for incompressible / weakly compressible problems in fluid and continuum mechanics
- GASDYN : explicit solver for gasdynamics flows
- SHALLOWWATER : explicit solver of the shallow water equations
- DROPLETPHASE : explicit solver for particle/droplet movement, mostly in interaction with a fluid flows
- MANIFOLD: solver for flow equations defined on a manifold/surface (EXPERIMENTAL)
- STANDBY : pointcloud containing data, that might be retrieved by approxY()

List of members:	
LIQUID	Implicit solver for incompressible and weakly compressible flow phenomena
GASDYN	Explicit solver of compressible flow phenomena
DROPLETPHASE	Explicit solver for droplets which may interact and collect as water films along boundaries
SHALLOWWATER	Solver for shallow water equations
TRANSPORT	TRANSPORT
STANDBY	stanby with data, no numerical algorithm applied on the data otherwise

7.2.1. DROPLETPHASE

Explicit solver for droplets which may interact and collect as water films along boundaries

For using the DROPLETPHASE solver, choose the kind of problem as

```
KOP(1) = DROPLETPHASE H:MIN_FACTOR(1.0)
```

The value H:MIN_FACTOR is optional and can be set between 0 and 1. It allows to adaptively reduce the smoothing length H, if too many points collect
at the same place. This can prevent an extreme increase in the number of neighbor points, which contributes to a significantly improved performance
in such situations. The default value is 1, which means that H is not changed.

Each point represents a volume of material. We assume droplet form, the volume is given with respect to the given diameter %ind_d30% .
Since points in the DROPLETPHASE represent discrete physical entities rather than numerical discretization points, they are not subject to the same point organization as, for example, the LIQUID phase.

Solver capabilities

There are several types of droplet dynamics which can be modeled within the DROPLETPHASE solver:

- **Free flight droplets**
  See FreeFlight
- **Liquid layer on a wall**
  See LiquidLayer
- **Collisions between droplets**
  See DropletCollisions

Boundary conditions

- Boundary conditions for DROPLETPHASE are described in DROPLETPHASE__BC__
- Due to the special treatment of DROPLETPHASE , %TOUCH_reflection% is often suitable for boundaries

Online Examples

- **(Non-interacting) Sand particles in water jet with one-way coupling**
  See WaterSand
- **Colliding droplets in cone geometry**
  See CollidingDropletsInCone
- **One-Way coupling of droplets and air in channel with filter**
  See ChannelWithFilter

List of members:

FreeFlight	DROPLETPHASE - Modeling of free flight droplets
LiquidLayer	DROPLETPHASE - Modeling of liquid layers
DropletCollisions	DROPLETPHASE - Modeling of collisions between droplets

DropletCollisions

DROPLETPHASE - Modeling of collisions between droplets

Through the use of

- ParticleInteraction
- %BND_COLLISION%

one may define the parameters of a collision model between droplets and boundaries. In this case, droplets will, assuming sensible choices of parameters, no longer pass through each other and behave more closely to granular matter.

Repulsive Normal Forces

The two parameters k_n, e_n in

```
ParticleInteraction( $Material$ ) = (k_n, e_n)
BC_v ( $BC1$ ) = ( %BND_COLLISION% , k_n, e_n)
```

determine the constants k_n and c_n within the linear spring-damper model

$$\mathbf{f}_{ij}^{n} = \begin{cases} -\left[k_n \delta_{ij} - c_n (\mathbf{v}_{ij} \cdot \mathbf{n}_{ij})\right] \mathbf{n}_{ij} & \delta_{ij} > 0 \\ 0 & \text{else} \end{cases}$$

which is employed to calculate forces along contact normal $\mathbf{n}_{ij}$ from overlap δ_{ij} and relative velocity $\mathbf{v}_{ij}$. In particular, e_n represents the coefficient of restitution, i.e. the ratio of post- to pre-collisional velocity, from which the damper constant c_n is calculated internally.

Defaults:

- If e_n is set to a negative value, the damping coefficient c_n will be set to the absolute value of e_n.
- If k_n is set to zero or a negative value, no collision forces will be calculated.

Notes:

- Interacting DROPLETPHASE particles are allowed to have different size and spring stiffness, however the coefficient of restitution must be identical.
- During the separation phase, the damper force might produce attractive contributions which lead to a reversal of sign. By default, this is prevented by setting the total force to zero as soon as the attractive damper force becomes larger in magnitude than the repulsive elastic force. See DP_UseOnlyRepulsiveContactForce .

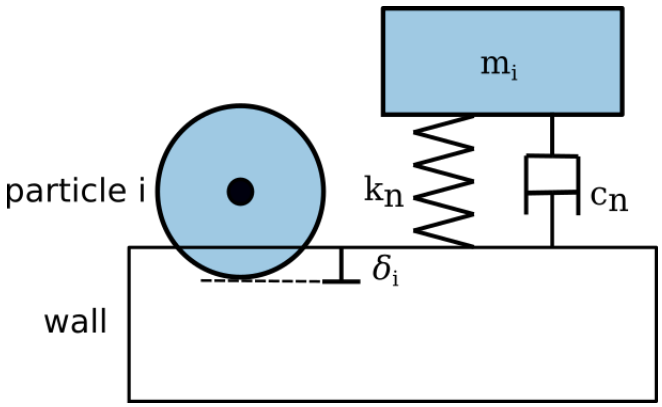


Figure: Sketch of Particle-Wall contact model

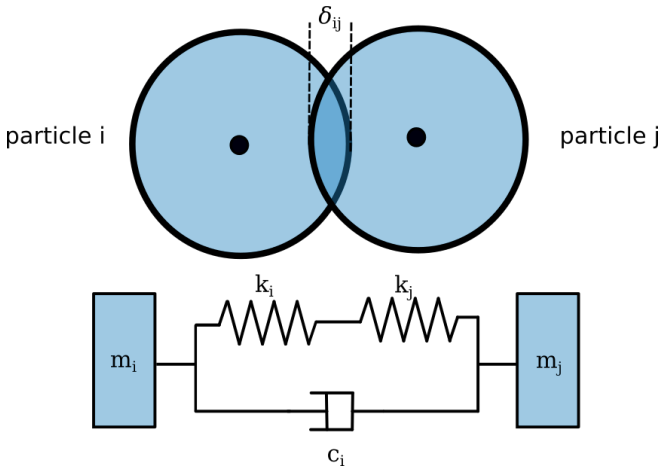


Figure: Sketch of Particle-Particle contact model

Attractive normal forces

The two parameters E_a, R_a in

```
ParticleInteraction( $Material$ ) = (k_n, e_n, E_a, R_a)
BC_v ( $BC1$ ) = ( %BND_COLLISION% , k_n, e_n, E_a, R_a)
```

are reserved for attractive forces along the normal direction. While R_a determines the range of adhesive forces (relative to the particle diameter D_i), E_a determines the energy level of the adhesive potential

$$\mathbf{f}_{ij}^{a} = \begin{cases} -k_a \delta_{ij} \mathbf{n}_{ij} & -\frac{1}{2} R_a D_i < \delta_{ij} < 0 \\ \left[-2F_{max}^{a} - k_a \delta_{ij}\right] \mathbf{n}_{ij} & -R_a D_i < \delta_{ij} < -\frac{1}{2} R_a D_i \\ 0 & \text{else} \end{cases}$$

i.e. k_a and F_{max}^{a} are chosen so that the integral over this force expression is given by E_a: $F_{max}^{a} = 2\frac{E_a}{R_a D_i}$

Defaults:

- If E_a is set to zero or a negative value, no attractive forces will be calculated.
- If R_a is set to zero or a negative value, it is overwritten by the default value of one.

Frictional forces

The parameter mu in

```
ParticleInteraction( $Material$ ) = (k_n, e_n, E_a, R_a, mu)
BC_v ( $BC1$ ) = ( %BND_COLLISION% , k_n, e_n, E_a, R_a, mu)
```

determines the coefficient of friction μ in Coulombs law of friction

$$\mathbf{f}_{ij}^t = -\mu \|\mathbf{f}_{ij}^n\| \mathbf{t}_{ij}$$

which is used to calculate forces along the tangential direction $\mathbf{t}_{ij}$ due to friction between the particles or between particles and boundaries.

Defaults:

- If mu is set to zero or a negative value, no friction forces will be calculated.

Notes:

- The coefficients of friction must be identical for all interacting DROPLETPHASE particles.

Important notes

Choice of smoothing length

- In order to resolve the collision dynamics correctly, the smoothing length h has to be at minimum 1.5 * D30. Otherwise the neighborhood lists are not correct and collision detection might be late or missed.
- Generally: other than in LIQUID the smoothing length generally doesn´t refine the resolution of the simulation, but is the quantity to define the neighbourhood information.

Timestep Management

- The stability of a simulation of interacting droplets can be improved by decreasing the value of COEFF_dt_d30 and COEFF_dt_coll .
- If the DROPLETPHASE timestep becomes too small, one may use a subcycling as described in COMP_DropletphaseSubcycles

---

MESHFREE · Solvers · Numerics · DROPLETPHASE · FreeFlight

FreeFlight

DROPLETPHASE - Modeling of free flight droplets

In free flight, the acceleration of the droplets is computed as

$$\frac{d\mathbf{v}_{drop}}{dt} = \hat{\mathbf{g}}$$

If in interaction with airflow (or another medium), the effective acceleration has to be set by the user as

$$\hat{\mathbf{g}} = \frac{c_D}{2} \frac{\rho_{air}}{\rho_{drop}} (\mathbf{v}_{air} - \mathbf{v}_{drop}) \|\mathbf{v}_{air} - \mathbf{v}_{drop}\| \frac{1}{d_{drop}} - \frac{1}{\rho_{drop}} \nabla p_{air} + \mathbf{g}$$

Typically this acceleration would be specified via gravity with $\mathbf{v}_{air}$ and ∇p_{air} being the velocity and pressure gradient of a LIQUID phase at droplet positions. The latter quantities can be calculated for example via approxY() .

MESHFREE · Solvers · Numerics · DROPLETPHASE · LiquidLayer

LiquidLayer

DROPLETPHASE - Modeling of liquid layers

For droplets collecting on a surface and forming a liquid layer, the acceleration of the droplets is computed as

$$\frac{d\mathbf{v}_{drop}}{dt} = \frac{\eta_{drop}^{normal}}{\rho_{drop}} \frac{\mathbf{v}_{geometry} - \mathbf{v}_{drop}}{H_{film}^2} + \frac{\eta_{drop}^{tangential}}{\rho_{drop}} \Delta(\mathbf{v}) + \hat{\mathbf{g}}$$

If in interaction with airflow (or another medium), the effective acceleration has to be set by the user as

$$\hat{\mathbf{g}} = \frac{1}{\rho_{drop}} \frac{\tau_{W,air}}{H_{film}} - \frac{1}{\rho_{drop}} \nabla p_{air} + \mathbf{g}$$

The computation of the height of the water film H_{film} is SPH-like in the sense

$$H_{film,i} = \sum_j W_{ij} V_j$$

where

- $V_j = \frac{\pi}{6} d^3_{drop,j}$ is the volume of the droplet (volume package represented by the MESHFREE point).
- in general we have the approximation kernel $\Pi u = \sum_j W_{ij} u_j \frac{V_j}{H_{film,j}}$ thus the layer thickness is an eigenfunction of the approximation kernel.

The gradient of the film thickness is

$$\nabla H_{film,i} = \sum_j \nabla W_{ij} V_j = \sum_j W_{ij} \left(-\frac{2\alpha}{h^2} \right) \mathbf{x}_{ij} V_j$$

which is needed to compare the current angle with the defined contact angle.

The curvature of the liquid film can be computed by the second derivatives, as for example

$$\frac{\partial^2}{\partial x^2} H_{film,i} = \sum_j W_{ij} \left(-\frac{2\alpha}{h^2} \right) V_j + \sum_j W_{ij} \left(-\frac{2\alpha}{h^2} \right)^2 x^2_{ij} V_j$$

MESHFREE · Solvers · Numerics · GASDYN

7.2.2. GASDYN

Explicit solver of compressible flow phenomena

We want to solve the system of conservation equations of gasdynamics

$$\begin{aligned}
&\tfrac{d}{dt}\rho + \rho \cdot \nabla^T \mathbf{v} = 0 \\
&\tfrac{d}{dt}(\rho \mathbf{v}) + (\rho \mathbf{v}) \cdot \nabla^T \mathbf{v} + \nabla p = 0 \\
&\tfrac{d}{dt}(\rho E) + (\rho E) \cdot \nabla^T \mathbf{v} + \nabla^T (p \cdot \mathbf{v}) = 0
\end{aligned}$$

where

- ρ is the density,
- $\mathbf{v} = (u, v, w)^T$ is the velocity,
- p is the pressure,
- $\rho E = \int_0^T c_v dT + \frac{\rho}{2} \mathbf{v}^T \mathbf{v}$ is the total energy,
- T is the temperature (Kelvin),
- c_v is the specific heat capacity.
- We have to complete the set of equations by the equation of state, in the most simple case by the perfect gas law
  $$p = \rho R T$$
  with R being the specific gas constant.
- The sound speed can be derived by
  $$c^2 = kRT$$
  where $k = \frac{c_v + R}{c_v}$ is the isentropic exponent.

For the numerical integration, we introduce the so called upwind velocity $\bar{\mathbf{v}}$ and upwind pressure $\bar{p}$, leading to a stabilization of the scheme. Numerically, we solve the modified conservation equations

$$\frac{d}{dt}\rho+\rho\cdot\nabla^T\bar{\mathbf{v}}=0$$
$$\frac{d}{dt}(\rho\mathbf{v})+(\rho\mathbf{v})\cdot\nabla^T\bar{\mathbf{v}}+\nabla\bar{p}=0$$
$$\frac{d}{dt}(\rho E)+(\rho E)\cdot\nabla^T\bar{\mathbf{v}}+\nabla^T(\bar{p}\cdot\bar{\mathbf{v}})=0$$

The sub-sections of this documentation item now provide different ways of approaching $\bar{\mathbf{v}}$ and $\bar{p}$.

List of members:

GeneralizedUpwind	Generalized way of computing upwind terms
SimplifiedFastUpwindTerms	Simplified, fast way of computing upwind terms
ClassicalUpwindTerms	Classical/original way of computing upwind terms

MESHFREE · Solvers · Numerics · GASDYN · ClassicalUpwindTerms

ClassicalUpwindTerms

Classical/original way of computing upwind terms

The classical formulation of the upwind terms. We use the upwind pressure and velocity of the form

$$\bar{p}=p-\frac{\rho c}{2}(\mathbf{v}^+-\mathbf{v}^-)^T\mathbf{n}$$
$$\bar{\mathbf{v}}=\mathbf{v}-\frac{1}{2\rho c}(p^+-p^-)\mathbf{n}$$

where $\mathbf{n}=(n_x,n_y,n_z)^T=\frac{1}{\|\nabla p\|}\nabla p$ is the upwind direction given by the pressure gradient.
Here, we evaluate $\mathbf{v}^+$, $\mathbf{v}^-$ and p^+ , p^- using MESHFREEs approximation tools (east squares approximation) at the upstream location

$$p^+(\mathbf{x})=\tilde{p}(\mathbf{x}+\alpha h\mathbf{n})$$

$$p^-(\mathbf{x})=\tilde{p}(\mathbf{x}-\alpha h\mathbf{n})$$

$$\mathbf{v}^+(\mathbf{x})=\tilde{\mathbf{v}}(\mathbf{x}+\alpha h\mathbf{n})$$

$$\mathbf{v}^-(\mathbf{x})=\tilde{\mathbf{v}}(\mathbf{x}-\alpha h\mathbf{n})$$

where α is the upwind parameter that defines the relative distance (compared to smoothing length h) to go upstream in order to evaluate the upwind quantities.
The derivatives of the upwind quantities are consequently

$$\tilde{\nabla}\bar{p}=\tilde{\nabla}p-\tilde{\nabla}\left(\frac{\rho c}{2}\right)(\mathbf{v}^+-\mathbf{v}^-)^T\mathbf{n}-\frac{\rho c}{2}\left(\left(\tilde{\nabla}(\mathbf{v}^{+T}\mathbf{n})-\tilde{\nabla}(\mathbf{v}^{-T}\mathbf{n})\right)\right)$$
$$\tilde{\nabla}^T\bar{\mathbf{v}}=\tilde{\nabla}^T\mathbf{v}-\tilde{\nabla}^T\left(\frac{1}{2\rho c}\right)(p^+-p^-)\mathbf{n}-\frac{1}{2\rho c}\left(\tilde{\nabla}^Tp^+-\tilde{\nabla}^Tp^-\right)\mathbf{n}$$

MESHFREE · Solvers · Numerics · GASDYN · GeneralizedUpwind

GeneralizedUpwind

Generalized way of computing upwind terms

This section here generalizes the way of computing upwind pressure and upwind velocity, stated in SimplifiedFastUpwindTerms and ClassicalUpwindTerms .

Method 1: original method, see ClassicalUpwindTerms

$$\bar{p}_{orig}=p-\frac{\rho c}{2}(\mathbf{v}^+-\mathbf{v}^-)^T\mathbf{n}\text{ with }\mathbf{v}^+(\mathbf{x})=\tilde{\mathbf{v}}(\mathbf{x}+\alpha h\mathbf{n})\text{ and }\mathbf{v}^-(\mathbf{x})=\tilde{\mathbf{v}}(\mathbf{x}-\alpha h\mathbf{n})$$
$$\bar{\mathbf{v}}_{orig}=\mathbf{v}-\frac{1}{2\rho c}(p^+-p^-)\mathbf{n}\text{ with }p^+(\mathbf{x})=\tilde{p}(\mathbf{x}+\alpha h\mathbf{n})\text{ and }p^-(\mathbf{x})=\tilde{p}(\mathbf{x}-\alpha h\mathbf{n})$$

With
αh the upwind step length

$\mathbf{n} = \frac{1}{\|\nabla p\|}\nabla p$ the upwind direction

Method 2: simplified and fast upwind quantities, see SimplifiedFastUpwindTerms

$$\bar{p}_{simp} = p - \rho c L\left(\tilde{\nabla}^T \mathbf{v}\right)$$
$$\bar{\mathbf{v}}_{simp} = \mathbf{v} - \frac{L}{\rho c}\tilde{\nabla} p$$

With

$L = \begin{cases} \text{either } \beta \cdot \Delta t \cdot c \\ \text{or } \gamma \cdot h \end{cases}$ the upwind step length (needed in the Taylor series expansion)

Hint: choosing $\beta = 0.5$ will lead to second order in time integration.

Combined method

We can bring both methods together into one

$$\bar{p}_{general} = p - \frac{\rho c}{2}\left(\mathbf{v}^+ - \mathbf{v}^-\right)^T \mathbf{n} - \rho c L\left(\tilde{\nabla}^T \mathbf{v}\right)$$
$$\bar{\mathbf{v}}_{general} = \mathbf{v} - \frac{1}{2\rho c}\left(p^+ - p^-\right)\mathbf{n} - \frac{L}{\rho c}\tilde{\nabla} p$$

- Define the parameter of α in GASDYN_UpwindOffset ,
- define the parameter of β in GASDYN_Upwind_Lbeta ,
- define the parameter of γ in GASDYN_Upwind_Lgamma .

Upwind lengths for different solvers?

- **FPM1** (original)
  $\alpha = 0.2$
  $\beta = 0$
  $\gamma = 0$
- **FPM1** (simplified, currently implemented in VPS)
  $\alpha = 0$
  $\beta = 0$
  $\gamma = 0.2$
- **FPM3**
  $\alpha = 0$
  $\beta = \begin{cases} 0.5 \; if \; \nabla^T \mathbf{v} > 0 \text{ (rarefaction)} \\ 0 \text{ elsewise} \end{cases}$
  $\gamma = \begin{cases} 0.0 \; if \; \nabla^T \mathbf{v} > 0 \text{ (rarefaction)} \\ 0.2 \text{ elsewise} \end{cases}$

SimplifiedFastUpwindTerms

Simplified, fast way of computing upwind terms

The upwind scheme of section ClassicalUpwindTerms has a major drawback: the evaluation of physical quantities at the locations $\mathbf{x}^+ = \mathbf{x} + \alpha h \mathbf{n}$ and $\mathbf{x}^- = \mathbf{x} - \alpha h \mathbf{n}$. Depending on the upwind direction $\mathbf{n}$, these locations might be outside of the flow domain. The idea of the simplified scheme is to approximate the upwind values by first order Taylor series expansion.

$$\bar{p} = p - \rho c L\left(\mathbf{n}^T \tilde{\nabla}\left(\mathbf{v}^T \mathbf{n}\right)\right) = p - \rho c L\left(n_x^2 u_{\tilde{x}} + n_y^2 v_{\tilde{y}} + n_z^2 w_{\tilde{z}} + \Phi_{mixed}\right)$$
$$\bar{\mathbf{v}} = \mathbf{v} - \frac{L}{\rho c}\tilde{\nabla} p$$

where

- $L = \alpha h$ is the upwind step size,
- $\Phi_{mixed} = n_x n_y v_{\tilde{x}} + n_x n_z w_{\tilde{x}} + n_y n_x u_{\tilde{y}} + n_y n_z w_{\tilde{y}} + n_z n_x u_{\tilde{z}} + n_z n_y v_{\tilde{z}}$ are mixed terms we assume to be of minor importance and therefore neglect them.

The derivatives of the upwind quantities are then

$$\tilde{\nabla}\bar{p} = \tilde{\nabla}p - \tilde{\nabla}\left(\rho c L\right)\left(n_x^2 u_{\tilde{x}} + n_y^2 v_{\tilde{y}} + n_z^2 w_{\tilde{z}}\right) - \left(\rho c L\right)\left(n_x^2 \begin{pmatrix} u_{\tilde{x}\tilde{x}} \\ u_{\tilde{x}\tilde{y}} \\ u_{\tilde{x}\tilde{z}} \end{pmatrix} + n_y^2 \begin{pmatrix} v_{\tilde{y}\tilde{x}} \\ v_{\tilde{y}\tilde{y}} \\ v_{\tilde{y}\tilde{z}} \end{pmatrix} + n_z^2 \begin{pmatrix} w_{\tilde{z}\tilde{x}} \\ w_{\tilde{z}\tilde{y}} \\ w_{\tilde{z}\tilde{z}} \end{pmatrix}\right)$$

$$\tilde{\nabla}^T\bar{\mathbf{v}} = \tilde{\nabla}^T\mathbf{v} - \tilde{\nabla}^T\left(\tfrac{L}{\rho c}\tilde{\nabla}p\right)$$

The even more simplified, fast upwind scheme comes now. In the equations above, the pressure formulation is difficult. However, one could further simplify

$$\bar{p} = p - \rho c L\left(\tilde{\nabla}^T\mathbf{v}\right)$$
$$\bar{\mathbf{v}} = \mathbf{v} - \tfrac{L}{\rho c}\tilde{\nabla}p$$

With this we are on the safe side, the damping can only be bigger (never smaller!) than the one of the original equations on top of this page, never bigger.
The derivatives of the upwind quantities are now

$$\tilde{\nabla}\bar{p} = \tilde{\nabla}p - \tilde{\nabla}\left(q\left(\tilde{\nabla}^T\mathbf{v}\right)\right) = \tilde{\nabla}p - \tilde{\nabla}^T\left(q\left(\tilde{\nabla}\mathbf{v}\right)\right) - \tilde{\nabla}\times\left(q\left(\tilde{\nabla}\times\mathbf{v}\right)\right) - \begin{pmatrix} q_x v_y - q_y v_x + q_x w_z - q_z w_x \\ q_y u_x - q_x u_y + q_y w_z - q_z w_y \\ q_z u_x - q_x u_z + q_z v_y - q_y v_z \end{pmatrix}$$

$$\tilde{\nabla}^T\bar{\mathbf{v}} = \tilde{\nabla}^T\mathbf{v} - \tilde{\nabla}^T\left(\tfrac{L}{\rho c}\tilde{\nabla}p\right)$$

where $q = \rho c L$ is simply a shortcut for more compact writing.
With this scheme we are able to prove/show the existence of mathematical damping in the upwind schemes. It is obvious that with the terms $\tilde{\nabla}^T\left(\rho c L\left(\tilde{\nabla}\mathbf{v}\right)\right)$ and $\tilde{\nabla}^T\left(\tfrac{L}{\rho c}\tilde{\nabla}p\right)$ we have mathematical damping for velocity and pressure, which act as stabilization of the scheme.

7.2.3. LIQUID

Implicit solver for incompressible and weakly compressible flow phenomena

The most advanced solver in the complete MESHFREE framework. It can handle incompressible or weakly compressible flow phenomena, i.e. the
solver accepts density formulations that might depend on time, pressure, and all other parameters needed.

List of members:

Algorithms	General collection of numerical algorithms used in MESHFREE
EquationsToSolve	differential equations to be solved by MESHFREE
Scheme	Scheme

Algorithms

General collection of numerical algorithms used in MESHFREE

Official documentation of the MESHFREE GFDM numerical solution algorithms. In future consequence, it shall be consistent with future GFDM papers.

List of members:	
BubbleAlgorithm	Bubble Algorithm in order to capture internal pressure of air/gas entrapments
TimeIntegrationAlgorithm	TimeIntegrationAlgorithm
VelocityAlgorithm	VelocityAlgorithm
VolumeCorrection	Volume Correction Algorithms in MESHFREE
DynamicPressureAlgorithm	DynamicPressureAlgorithm
CorrectionPressureAlgorithm	compute the correction pressure according to a Chorin-like correction ansatz
KepsilonAlgorithm	turbulence modelling using the k-epsilon model
HydrostaticPressureAlgorithm	HydrostaticPressureAlgorithm
TemperatureAlgorithm	TemperatureAlgorithm
PreparationAlgorithm	PreparationAlgorithm
StressTensorAlgorithm	update the solid stress tensor towards the next time level

BubbleAlgorithm

Bubble Algorithm in order to capture internal pressure of air/gas entrapments

The BubbleAlgorithm is switched on if BUBBLE_DoTheManagement is different from zero.

BUBBLE_DoTheManagement = 1 (original implementation) : BubbleSemiimplicitPressure is applied
BUBBLE_DoTheManagement = 2 : BubbleImplicitPressure is applied

The Bubble Algorithm clusters the free surfaces by connectivities, computes the volumes of the individual clusters (possibly entrapped also with the geometry) and computes the pressures corresponding to the tracked volume changes.

The relevant MESHFREE quantities are

Quantity	Index
BubbleDetection	%ind_bndBubble%
BubbleVolume	%ind_volBubble%
BubblePressure	%ind_pBubble%

List of members:	
BubblePressure	approximate pressure of closed bubbles
BubbleDetection	detect topologically connected clusters of free surface
BubbleVolume	approximate volume of topologically connected AND closed clusters of free surface

BubbleDetection

detect topologically connected clusters of free surface

MESHFREE searches for completely enclosed partitions of a free surface, or free surface in conjunction with inactive wall points. Topologically connected free surface partitions are marked by a dedicated index, the index can be reviewed in the variable %ind_bndBubble% .

MESHFREE · Solvers · Numerics · LIQUID · Algorithms · BubbleAlgorithm · BubblePressure

BubblePressure

approximate pressure of closed bubbles

The bubble pressure may be computed in different ways, see below. The pressure values stored in %ind_pBubble% are the ones of the BubbleSemiimplicitPressure , as they are numerically stable.

List of members:

BubbleTruePressure	true bubble pressure computed based on the volumetric compression
BubbleSemiimplicitPressure	bubble pressure by integration of the correction pressure values along the bubble surface
BubbleImplicitPressure	bubble pressure by integration of the correction pressure values along the bubble surface

MESHFREE · Solvers · Numerics · LIQUID · Algorithms · BubbleAlgorithm · BubblePressure · BubbleImplicitPressure

BubbleImplicitPressure

bubble pressure by integration of the correction pressure values along the bubble surface

We require that the pressure gradient along the bubble surface assumes a certain value.
The bubble pressure is updated by

$$p_b^{n+1} = p_{hyd,b}^{n+1} + c_b^{n+1}$$

That also means, that both algorithms BubbleHydrostaticPressure and BubbleCorrectionPressure have to be employed.

List of members:

BubbleCorrectionPressure	bubble correction pressure by implicit requirement along the bubble surface
BubbleHydrostaticPressure	bubble hydrostatic pressure by implicit requirement for hydrostatic pressure along the bubble surface

MESHFREE · Solvers · Numerics · LIQUID · Algorithms · BubbleAlgorithm · BubblePressure · BubbleImplicitPressure · BubbleCorrectionPressure

BubbleCorrectionPressure

bubble correction pressure by implicit requirement along the bubble surface

By the first derivative to the BubbleTruePressure formulation we have

$$\frac{dp_b}{dt}V_b + \kappa p_b \frac{dV_b}{dt} = V_b \cdot \left.\frac{dp_b}{dt}\right|_{penalty}$$

The right-hand-side normally would be zero, however we make the ansatz of a penalty term, that takes into account approximation errors or bubble volume dilution due to point cloud management (adding/removing points).

We consider the bubble in the current state as it is, and we would like to compute the correction pressure (%ind_c%) such that a certain desired volume change turns out, or pressure increases.
The time derivative of the pressure is given by

$$\frac{dp_b}{dt} \approx \frac{p_{hyd,b}^{n+1} + p_{dyn,b}^{n} + c_b^{n+1} - p_b^n}{\Delta t}$$

the volume change rate of the bubble is

$$\frac{dV_b}{dt} = \oint\limits_{\partial\Omega_b} \mathbf{v}^{n+1} \cdot \mathbf{n} dA = \oint\limits_{\partial\Omega_b} \left(\tilde{\mathbf{v}}^{n+1} - \frac{\Delta t_{virt}}{\rho} \nabla c^{n+1} \right) \cdot \mathbf{n} dA$$

So we have

$$\begin{aligned}
&\frac{p_{hyd,b}^{n+1} + p_{dyn,b}^{n} + c_b^{n+1} - p_b^n}{\Delta t} V_b + \kappa p_b \cdot \oint\limits_{\partial\Omega_b} \left(\tilde{\mathbf{v}}^{n+1} - \frac{\Delta t_{virt}}{\rho} \nabla c^{n+1} \right) \cdot \mathbf{n} dA && = V_b \left. \frac{dp_b}{dt} \right|_{penalty} \\
&p_{hyd,b}^{n+1} + p_{dyn,b}^{n} + c_b^{n+1} - p_b^n + \Delta t \frac{\kappa p_b}{V_b} \cdot \oint\limits_{\partial\Omega_b} \left(\tilde{\mathbf{v}}^{n+1} - \frac{\Delta t_{virt}}{\rho} \nabla c^{n+1} \right) \cdot \mathbf{n} dA && = \Delta t \left. \frac{dp_b}{dt} \right|_{penalty} \\
&c_b^{n+1} - \Delta t \frac{\kappa p_b}{V_b} \cdot \oint\limits_{\partial\Omega_b} \frac{\Delta t_{virt}}{\rho} \nabla c^{n+1} \cdot \mathbf{n} dA && = p_b^n - p_{hyd,b}^{n+1} - p_{dyn,b}^{n} - \Delta t \frac{\kappa p_b}{V_b} \cdot \oint\limits_{\partial\Omega_b} \tilde{\mathbf{v}}^{n+1} \cdot \mathbf{n} dA + \Delta t \left. \frac{dp_b}{dt} \right|_{penalty}
\end{aligned}$$

Finally, we need to replace the surface integral by their appropriate sums, thus

$$c_b^{n+1} - \Delta t \frac{\kappa p_b}{V_b} \cdot \sum_{i \in \partial\Omega_b} \frac{\Delta t_{virt}}{\rho} \nabla c_i^{n+1} \cdot \mathbf{n}_i A_i = p_b^n - p_{hyd,b}^{n+1} - p_{dyn,b}^{n} - \Delta t \frac{\kappa p_b}{V_b} \cdot \sum_{i \in \partial\Omega_b} \tilde{\mathbf{v}}^{n+1} \cdot \mathbf{n}_i A_i + \Delta t \cdot \left. \frac{dp_b}{dt} \right|_{penalty}$$

This is an implicit formulation, as the bubble correction pressure depends on the correction pressure gradient along the bubble surface.

Last question to solve is, what is $\left. \frac{dp_b}{dt} \right|_{penalty}$? See in BubblePressurePenaltyChangeRate .
The answer to this question stems from the potential discrepancy between the true bubble pressure, computed only from the current volume and the original volume and pressure of the bubble (see BubbleTruePressure). See especially BubblePressurePenaltyChangeRate .

Be sure to set the appropriate boundary conditions in order to invoke this algorithm:

```
BC_p (0) = ( %BND_free_implicit%, A ) # A does not have a meaning for %ind_c% - conditions
BCON (0,%ind_c%) = ( %BND_free_implicit%, A ) # this is equivalent, it allows to set forth different conditions for
%ind_p% and %ind_c%
```

Otherwise, if setting for example conditions like

```
BCON (0,%ind_c%) = ( %BND_free% , A ) # results in Dirichlet behavior along the free surface for %ind_c%
BCON (0,%ind_c%) = ( %BND_DIRICH% , A ) # dito
```

then the implicit character is not invoked, and, instead, the following condition solved at the bounds of the bubble

$$c_b^{n+1} = A$$

List of members:

BubblePressurePenaltyChangeRate	target volume change rate of a bubble

BubblePressurePenaltyChangeRate

target volume change rate of a bubble

We require that the bubble always comes back to its original value of $p_b^0 \left(V_b^0\right)^\kappa$ (see BubbleTruePressure), i.e. the correct bubble pressure to be computed by

$$p^n_{true,b} = p^0_b \left(\frac{V^0_b}{V^n_b} \right)^\kappa$$

This equation exactly states the bubble pressure due to the volume change the bubble was undergoing since it was formed.
Only this pressure is the repreeantative bubble pressure.
However, if we would immediatle apply this pressure in the computations, the numerics might become unstable towards small or stiff bubbles.
Therefore, we work with the numerically computed pressure (based on implicit condiderations, see BubbleImplicitPressure) and try to smoothly conduct the
numerical bubble pressure towards the true value.

So let us bring the two pressures into coincidence by a prenalty formulation

$$\left. \frac{dp_b}{dt} \right|_{penalty} = \alpha_b \frac{p^n_{true,b} - p^n_b}{\Delta t}$$

where $\alpha_b < 1$ is free to choose, currently hard coded to a value of 0.1 .

BubbleHydrostaticPressure

bubble hydrostatic pressure by implicit requirement for hydrostatic pressure along the bubble surface

According to BubbleTruePressure , we have ($\kappa = 1$):

$$\frac{dp_b}{dt} \cdot V_b + p_b \cdot \frac{dV_b}{dt} = 0$$

We can numerically discretize

$$\frac{p^{n+1}_b - p^n_b}{\Delta t} \cdot V^n_b + p^n_b \cdot \int\limits_{\partial\Omega_b} \boldsymbol{v}^{n+1} \cdot \boldsymbol{n} dA = 0$$

and moreover, with the relevant parts of the momentum equation (see EquationsToSolve)

$$\frac{p^{n+1}_b - p^n_b}{\Delta t} \cdot V^n_b + p^n_b \cdot \int\limits_{\partial\Omega_b} \left(\boldsymbol{v}^n - \frac{\Delta t}{\rho} \nabla p^{n+1}_b + \Delta t \boldsymbol{g} \right) \cdot \boldsymbol{n} dA = 0$$

For the hydrostatic bubble pressure, we assume constant gravity and currently volume conserving flow field, and let us require that the new hydrostatic pressure resolves the static pressure field, i.e.

$$\frac{p^{n+1}_{hyd,b} - p^n_b}{\Delta t} \cdot V^n_b + p^n_b \cdot \int\limits_{\partial\Omega_b} \frac{\Delta t}{\rho} \nabla p^{n+1}_{hyd,b} \boldsymbol{n} dA = 0$$

and finally

$$p^{n+1}_{hyd,b} + \frac{\Delta t \cdot p^n_b}{V^n_b} \sum_{i \in \partial\Omega_b} \frac{\Delta t}{\rho} \nabla p^{n+1}_i \boldsymbol{n}_i A_i = p^n_b$$

$p^{n+1}_{hyd,b}$ can now be applied implicitely as boundary condition for the HydrostaticPressureAlgorithm along the free surface connected with the bubble concerned.

Important remark: The implicit / semiimplicit bubble pressure becomes effective ONLY, if appropriate boundary conditions:

```
BC_p (0) = ( %BND_free_implicit%, [Y %ind_pBubble% ] )
```

If, for example, one would set forth the boundary condition

```
BC_p (0) = ( %BND_free% , [Y %ind_pBubble% ] )
BC_p (0) = ( %BND_DIRICH% , [Y %ind_pBubble% ] )
```

that would give up the implicit character, as one woul solve the equation

$$p_{hyd,b}^{n+1} = p_b^n$$

If, for example, one would set forth the boundary condition

```
BC_p (0) = ( %BND_free_implicit%, 0 )
```

that would give up the implicit character, as one woul solve the equation

$$p_{hyd,b}^{n+1} + \frac{\Delta t \cdot p_b^n}{V_b^n} \sum_{i \in \partial\Omega_b} \frac{\Delta t}{\rho} \nabla p_i^{n+1} \boldsymbol{n}_i A_i = 0$$

i.e. one would reproject the inner bubble pressure to the reference pressure, given by the parameter BUBBLE_pOffset . So, please be careful when setting the boundary conditions.

BubbleSemiimplicitPressure

bubble pressure by integration of the correction pressure values along the bubble surface

$$\begin{aligned} p_{\text{bubble}}(t+dt) &= p_{\text{bubble}}(t) + c_{\text{bubble}}(t+dt) \\ p_{\text{bubble}}^{n+1} &= p_{\text{bubble}}^{n} + c_{\text{bubble}}^{n+1} \\ p_b^{n+1} &= p_b^n + c_b^{n+1} \end{aligned}$$

where $c_{\text{bubble}}(t) = c_b^{n+1}$ is the correction pressure (produced by employing the BubbleCorrectionPressure algorithm , see also %ind_c%) at the bubble surface for the current time step.

BubbleTruePressure

true bubble pressure computed based on the volumetric compression

$$p_{\text{bubble}}^{\text{true}}(t) \cdot V_{\text{bubble}}^{\kappa}(t) = p_0 \cdot V_0^{\kappa} = const.$$

Currently we work with $\kappa = 1$. Currently, the only way to interrogate the true bubble pressure $p_{\text{bubble}}^{\text{true}}(t)$ is by an Equations statement, see %BUBBLE_EQN_TruePressure% .

BubbleVolume

approximate volume of topologically connected AND closed clusters of free surface

The volume of a given bubble is computed by a simple surface integral:

$$V_{\text{bubble}} = \int_{\partial\Omega_{\text{bubble}}} (x - x_0) n^x dA = \int_{\partial\Omega_{\text{bubble}}} (y - y_0) n^y dA = \int_{\partial\Omega_{\text{bubble}}} (z - z_0) n^z dA$$

where $\boldsymbol{n} = (n^x, n^y, n^z)^T$ is the boundary normal, $\boldsymbol{x}_0 = (x_0, y_0, z_0)^T$ is a refernce position. The approximation of these surface integral is

$$V_{\text{bubble}} \approx \sum_{\partial\Omega_{\text{bubble}}} (x_i - x_0) n_i^x A_i \approx \sum_{\partial\Omega_{\text{bubble}}} (y_i - y_0) n_i^y A_i \approx \sum_{\partial\Omega_{\text{bubble}}} (z_i - z_0) n_i^z A_i$$

The volume of a bubble is stored in the variable %ind_volBubble% .

The three sums represent approximations of the bubble volume seen from the three different principal directions. There are checks for the bubbles whether they are valid and depending on this the value %ind_volBubble% is set.

Bubble Volume	Meaning
> 0	regular bubble
negative real volume	the bubble is regular, but touches not only wall or slip boundaries, e.g. an outflow boundary is touched.
-1e11	eigenvalue check. Sort out droplets.
-1e12	the bubble touches an open edge. Control the limit value for the edges by BUBBLE_EdgeValue .
-1e23	Irregularity check, sum over normal times area element must be close to zero.

CorrectionPressureAlgorithm

compute the correction pressure according to a Chorin-like correction ansatz

Let us suppose we have a velocity field $\tilde{\mathbf{v}}^{n+1}$ that stems from the numerical integration of the momentum equation (see EquationsToSOlve), i.e. we have computed

$$\frac{\tilde{\mathbf{v}}^{n+1}-\mathbf{v}^{n}}{\Delta t}+\frac{1}{\rho}\nabla p^{n}=\frac{1}{\rho}\left(\nabla^{T}\mathbf{S}_{s}^{n}\right)^{T}+\frac{1}{\rho}\left(\nabla^{T}\mathbf{S}_{v}\left(\tilde{\mathbf{v}}^{n+1}\right)\right)^{T}+\mathbf{g}-\beta\cdot\tilde{\mathbf{v}}^{n+1}$$

The resulting velocity field $\tilde{\mathbf{v}}^{n+1}$ does most probably not provide the correct value of divergence of velocity $\nabla^{T}\tilde{\mathbf{v}}^{n+1}$. Let us suppose there is a correction to the pressure c that exactly leads to the correct divergence, that is

$$\frac{\mathbf{v}^{n+1}-\mathbf{v}^{n}}{\Delta t}+\frac{1}{\rho}\nabla p^{n}+\frac{1}{\rho}\nabla c=\frac{1}{\rho}\left(\nabla^{T}\mathbf{S}_{s}^{n}\right)^{T}+\frac{1}{\rho}\left(\nabla^{T}\mathbf{S}_{v}\left(\mathbf{v}^{n+1}\right)\right)^{T}+\mathbf{g}-\beta\cdot\mathbf{v}^{n+1}$$

With the presumption that $\nabla^{T}\mathbf{v}^{n+1}$ has the correct value. In order to find the correction pressure c we subtract the two equations from one another, that is

$$\left(\frac{1+\Delta t\beta}{\Delta t}\right)\cdot\left(\mathbf{v}^{n+1}-\tilde{\mathbf{v}}^{n+1}\right)+\frac{1}{\rho}\nabla c=+\frac{1}{\rho}\left(\nabla^{T}\mathbf{S}_{v}\left(\mathbf{v}^{n+1}\right)\right)^{T}-\frac{1}{\rho}\left(\nabla^{T}\mathbf{S}_{v}\left(\tilde{\mathbf{v}}^{n+1}\right)\right)^{T}$$

Written in another way, we have

$$\left(\mathbf{v}^{n+1}-\tilde{\mathbf{v}}^{n+1}\right)+\left(\frac{\Delta t}{1+\Delta t\beta}\right)\frac{1}{\rho}\nabla c=\left(\frac{\Delta t}{1+\Delta t\beta}\right)\frac{1}{\rho}\left(\nabla^{T}\mathbf{S}_{v}\left(\mathbf{v}^{n+1}\right)\right)^{T}-\left(\frac{\Delta t}{1+\Delta t\beta}\right)\frac{1}{\rho}\left(\nabla^{T}\mathbf{S}_{v}\left(\tilde{\mathbf{v}}^{n+1}\right)\right)^{T}$$

For incompressible problems with constant viscosity, we can simplify

$$\left(\mathbf{v}^{n+1}-\tilde{\mathbf{v}}^{n+1}\right)+\left(\frac{\Delta t}{1+\Delta t\beta}\right)\frac{1}{\rho}\nabla c=\left(\frac{\Delta t}{1+\Delta t\beta}\right)\frac{\eta}{\rho}\Delta\left(\mathbf{v}^{n+1}-\tilde{\mathbf{v}}^{n+1}\right)$$

The correction pressure stems from the simplified correction ansatz of a given velocity field (marked by tilde) towards a velocity field with a desired divergence of velocity.

$$\mathbf{v}^{n+1}=\tilde{\mathbf{v}}^{n+1}-\frac{\Delta t_{virt}}{\left(1+\Delta t_{\beta}\cdot\beta\right)}\frac{1}{\rho}\nabla c$$

For the term Δt_{virt} see VirtualTimeStepSize .

By application of the divergence operator from left, we obtain

$$\left(\nabla^{T}\mathbf{v}^{n+1}\right)_{target}-\nabla^{T}\tilde{\mathbf{v}}^{n+1}+\nabla^{T}\left(\frac{\Delta t_{virt}}{\left(1+\Delta t_{\beta}\cdot\beta\right)}\frac{1}{\rho}\nabla c\right)=0$$

The desired divergence of the velocity is depending on the compressibility of the fluid as well as on temporal changes of the density due to other effects
such as chemical reaction, expansion due to heating, etc.

Derivation of this term is found in DesiredAndNominalDivergenceOfVelocity .

Having a formulation for the divergence of velocity, the equation to be solved for the correction pressure is

$$-\frac{1}{\rho}\frac{\partial \rho}{\partial p}\frac{1}{\Delta t}c+\nabla^T\left(\frac{\Delta t_{virt}}{\left(1+\Delta t_\beta\cdot\beta\right)}\frac{1}{\rho}\nabla c\right)-\nabla^T\tilde{\mathbf{v}}^{n+1}=-\left(\overline{\nabla^T\mathbf{v}}\right)_c^{n+1}$$

which numerically leads to the (linear) system to be solved

$$-\frac{1}{\rho}\frac{\partial \rho}{\partial p}\frac{1}{\Delta t}c_i+\sum_j d_{ij}^{\nabla^T\left(\frac{\Delta t_{virt}}{\left(1+\Delta t_\beta\cdot\beta\right)}\frac{1}{\rho}\nabla\right)}\cdot c_{ij}-\nabla^T\tilde{\mathbf{v}}^{n+1}=-\left(\overline{\nabla^T\mathbf{v}}\right)_c^{n+1}$$

The result of this equation is stored in %ind_c% .

List of members:

DesiredAndNominalDivergenceOfVelocity	derive a formulation for the desired divergence of velocity
VirtualTimeStepSize	virtual time step size to control the correction pressure or the divergence of velocity

MESHFREE · Solvers · Numerics · LIQUID · Algorithms · CorrectionPressureAlgorithm · DesiredAndNominalDivergenceOfVelocity

DesiredAndNominalDivergenceOfVelocity

derive a formulation for the desired divergence of velocity

From the mass conservation (see EquationsToSolve) we can derive

$$\begin{aligned}
\left(\nabla^T\mathbf{v}\right)_{target} &= -\tfrac{d}{dt}\left(\log(\rho)\right)\\
\left(\nabla^T\mathbf{v}\right)_{target}^{n+1} &\approx -\tfrac{1}{\Delta t}\left(\log\left(\rho_c^{n+1}\right)-\log\left(\rho_c^{n}\right)\right)\\
&\approx -\tfrac{1}{\Delta t}\left(\log\left(\rho\left(t^{n+1},p_{hyd}^{n+1}+\tilde{p}_{dyn}^{n}+c,T^{n+1},A_v^n\right)\right)-\log\left(\rho\left(t^{n},p_{hyd}^{n}+p_{dyn}^{n},T^{n},A_v^{n-1}\right)\right)\right)\\
&\approx -\tfrac{1}{\Delta t}\left(\log\left(\rho\left(t^{n+1},p_{hyd}^{n+1}+\tilde{p}_{dyn}^{n},T^{n+1},A_v^n\right)\right)+\tfrac{1}{\rho}\tfrac{\partial\rho}{\partial p}c-\log\left(\rho\left(t^{n},p_{hyd}^{n}+p_{dyn}^{n},T^{n},A_v^{n-1}\right)\right)\right)\\
&\approx -\tfrac{1}{\Delta t}\left(\log\left(\tfrac{\rho\left(t^{n+1},p_{hyd}^{n+1}+\tilde{p}_{dyn}^{n},T^{n+1},A_v^n\right)}{\rho\left(t^{n},p_{hyd}^{n}+p_{dyn}^{n},T^{n},A_v^{n-1}\right)}\right)\right)-\tfrac{1}{\Delta t}\tfrac{1}{\rho}\tfrac{\partial\rho}{\partial p}c\\
&\approx \left(\overline{\nabla^T\mathbf{v}}\right)_c^{n+1}-\tfrac{1}{\Delta t}\tfrac{1}{\rho}\tfrac{\partial\rho}{\partial p}c
\end{aligned}$$

The term $\frac{1}{\Delta t}\frac{1}{\rho}\frac{\partial\rho}{\partial p}$ represents the compressibility of the fluid. The term is saved in %ind_DiagPcorr% .
Finally, the definition of the nominal (compression free) divergence of velocity is

$$\begin{aligned}
\left(\overline{\nabla^T\mathbf{v}}\right)_c^{n+1} &\equiv -\tfrac{1}{\Delta t}\left(\log\left(\tfrac{\rho\left(t^{n+1},p_{hyd}^{n+1}+\tilde{p}_{dyn}^{n},T^{n+1},A_v^n\right)}{\rho\left(t^{n},p_{hyd}^{n}+p_{dyn}^{n},T^{n},A_v^{n-1}\right)}\right)\right)\\
&= -\tfrac{1}{\Delta t}\left(\log\left(\tfrac{\rho\left(t^{n+1},p_{hyd}^{n+1}+\tilde{p}_{dyn}^{n},T^{n+1},A_v^n\right)}{\rho_{p_{dyn}}^{n}}\right)\right)
\end{aligned}$$

MESHFREE · Solvers · Numerics · LIQUID · Algorithms · CorrectionPressureAlgorithm · VirtualTimeStepSize

VirtualTimeStepSize

virtual time step size to control the correction pressure or the divergence of velocity

The virtual time step size helps to control the correction pressure.
We choose

$$\Delta t_{virt} = \min\left(\Delta t, A_{virt}\frac{\rho h^2}{\hat{\eta}}\right)$$

The term A_{virt} is represented by the input parameter COEFF_dt_virt .

The actually used value of Δt_{virt} can be retrieved by the variable %ind_dt_virt% .

A universal number is $A_{virt} = 1$.

In case of vp- , one can try to make A_{virt} as small as possible, in order to force $\nabla^T \boldsymbol{v}$ as close to the target value as possible.

In case of v-- together with small local Re numbers, choosing $A_{virt} > 1$ is favorable in order to avoid oscillations of the correction pressure.

DynamicPressureAlgorithm

List of members:

ClassicalDPA	compute the dynamic(consistent) pressure as a (postprocessing) result to the current velocity field
RegularizeDPA	regularize the computation of dynamic pressure in order to reduce fluctuations
AlternativeDPA	compute the consistent pressure as a (postprocessing) result to the current velocity field

---

AlternativeDPA

compute the consistent pressure as a (postprocessing) result to the current velocity field

Important remark. This algorithm is obsolete as it is contained in the ClassicalDPA by using the option

```
FLIQUID_ConsistentPressure_Version = 1127
```

==========

This algorithm is invoked if the first digit of the variable FLIQUID_ConsistentPressure_Version is put to 2.

Short derivation/motivation:

Let us again consider the equation of momentum

$$\dot{\mathbf{v}} + \frac{1}{\rho}\nabla p = \frac{1}{\rho}\left(\nabla^T \mathbf{S}_s\right)^T + \frac{1}{\rho}\left(\nabla^T \mathbf{S}_v\left(\mathbf{v}\right)\right)^T + \mathbf{g} - \beta\cdot\mathbf{v}$$

and isolate for the target dynamic pressure gradient

$$\nabla p_{dyn}^{\text{target}} = \left(\nabla^T \mathbf{S}_v\left(\mathbf{v}\right)\right)^T - \rho\beta\cdot\mathbf{v} - \rho\dot{\mathbf{v}}$$

For simplicity, we now omit the suffix >>>dyn<< <. between="" href="MESHFREE.html" two=""> MESHFREE points i and j we can compute the intermediate pressure value by

$$\bar{p}_{ij} = p_i + \frac{\rho_j}{\rho_i + \rho_j} \cdot \left(\mathbf{x}_j - \mathbf{x}_i\right)^T \cdot \nabla p_i^{\text{target}} = p_j + \frac{\rho_i}{\rho_i + \rho_j} \cdot \left(\mathbf{x}_i - \mathbf{x}_j\right)^T \cdot \nabla p_j^{\text{target}}$$

So, we can write

$$\frac{\rho_i}{\rho_i + \rho_j} \cdot \left(\mathbf{x}_j - \mathbf{x}_i\right)^T \cdot \nabla p_j^{\text{target}} + \frac{\rho_j}{\rho_i + \rho_j} \cdot \left(\mathbf{x}_j - \mathbf{x}_i\right)^T \cdot \nabla p_i^{\text{target}} = p_j - p_i$$

or even better, in order to have full symmetry,

$$\frac{1}{2}\left(\mathbf{x}_j - \mathbf{x}_i\right)^T \cdot \frac{1}{\rho_j} \nabla p_j^{\text{target}} + \frac{1}{2}\left(\mathbf{x}_j - \mathbf{x}_i\right)^T \cdot \frac{1}{\rho_i} \nabla p_i^{\text{target}} = \frac{1}{2} \frac{\rho_i + \rho_j}{\rho_i \rho_j} \left(p_j - p_i\right)$$

For each MESHFREE point i we have as many equations of this type as there are neighbour points, which means we have an
overdetermined system. However, we could require

$$\sum_{j=1}^{N(i)} W_{ij} \left(\frac{1}{2}\left(\mathbf{x}_j - \mathbf{x}_i\right)^T \cdot \frac{1}{\rho_j} \nabla p_j^{\text{target}} + \frac{1}{2}\left(\mathbf{x}_j - \mathbf{x}_i\right)^T \cdot \frac{1}{\rho_i} \nabla p_i^{\text{target}} \right) = \sum_{j=1}^{N(i)} W_{ij} \left(\frac{1}{2} \frac{\rho_i + \rho_j}{\rho_i \rho_j} \left(p_j - p_i\right) \right)$$

This set forms a linear system of equations for the unknowns p_i .

So far, only choosing $W_{ij} = c_{ij}^{\Delta}$ provided very stable results. This special chices of the weight function, by the way, provides a nice similarity to the classical ansatz, if we
also remember, that

$$c_{ij}^x = \frac{1}{2}\left(x_j - x_i\right) \cdot c_{ij}^{\Delta},\ c_{ij}^y = \frac{1}{2}\left(y_j - y_i\right) \cdot c_{ij}^{\Delta},\ c_{ij}^z = \frac{1}{2}\left(z_j - z_i\right) \cdot c_{ij}^{\Delta}$$

ClassicalDPA

compute the dynamic(consistent) pressure as a (postprocessing) result to the current velocity field

This algorithm is invoked if the first digit of the variable FLIQUID_ConsistentPressure_Version is put to 1.

According to DynamicPressure , the precise model for the dynamic pressure is

$$\frac{\left(\nabla^T \mathbf{v}\right)_{dyn}^{n+1} - \left(\nabla^T \mathbf{v}\right)^n}{\Delta t} + \nabla^T \left(\frac{1}{\rho} \nabla \left(p_{dyn}^{n+1} \right) \right) = \Psi\left(\mathbf{v}^{n+1}\right) - \Theta_{dyn}\left(\mathbf{v}^{n+1}\right) - \Phi\left(\mathbf{v}^{n+1}\right)$$

Which, in another way, is

$$\frac{\left(\overline{\nabla^T \mathbf{v}}\right)_{dyn}^{n+1} - \frac{1}{\Delta t}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p}\left(p_{dyn}^{n+1} - p_{dyn}^n\right)\right) - \left(\nabla^T \mathbf{v}\right)^n}{\Delta t} + \nabla^T \left(\frac{1}{\rho} \nabla \left(p_{dyn}^{n+1} \right) \right) = \Psi\left(\mathbf{v}^{n+1}\right) - \Theta_{dyn}\left(\mathbf{v}^{n+1}\right) - \Phi\left(\mathbf{v}^{n+1}\right)$$

Reorganization (step by step) yields

$$\frac{1}{\Delta t}\left(\left(\overline{\nabla^T \mathbf{v}}\right)_{dyn}^{n+1} - \left(\nabla^T \mathbf{v}\right)^n\right) - \frac{1}{\Delta t^2}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p}\left(p_{dyn}^{n+1} - p_{dyn}^n\right)\right) + \nabla^T \left(\frac{1}{\rho} \nabla \left(p_{dyn}^{n+1} \right) \right) = \Psi\left(\mathbf{v}^{n+1}\right) - \Theta_{dyn}\left(\mathbf{v}^{n+1}\right) - \Phi\left(\mathbf{v}^{n+1}\right)$$

And finally

$$\begin{aligned} -\frac{1}{\Delta t^2}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p} p_{dyn}^{n+1}\right) + \nabla^T \left(\frac{1}{\rho} \nabla \left(p_{dyn}^{n+1} \right) \right) = & -\frac{1}{\Delta t}\left(\left(\overline{\nabla^T \mathbf{v}}\right)_{dyn}^{n+1} - \left(\nabla^T \mathbf{v}\right)^n\right) \\ & -\frac{1}{\Delta t^2}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p} p_{dyn}^n\right) \\ & +\Psi\left(\mathbf{v}^{n+1}\right) - \Theta_{dyn}\left(\mathbf{v}^{n+1}\right) - \Phi\left(\mathbf{v}^{n+1}\right) \end{aligned}$$

The numerical discretization of this PDE is

$$\sum_{j=1}^{N(i)} \left(c_{ij}^{\nabla^T \frac{1}{\rho} \nabla} - \delta_{ij} \frac{1}{\Delta t^2} \frac{1}{\rho} \frac{\partial \rho}{\partial p} \right) p_j = \sum_{j=1}^{N(i)} W_{ij} p_j = Q_i$$

with i running over all MESHFREE point indices, W_{ij} being the matrix indices and Q_i the right hand side vector of the global, sparse linear system.

List of members:	
ComputationOfPHI	how to numericall compute the source term that goes with intertial forces
ComputationOfPSI	how to numerically compute the source term that goes with the viscous forces
ComputationOfTHETA	how to numerically compute the source term due to the Darcy forces

ComputationOfPHI

how to numericall compute the source term that goes with intertial forces

From DerivePoissonEquationForPressure we have a formulation for $\Phi(\mathbf{v})$, which is

$$\Phi(\mathbf{v}) = \nabla^T \left(\frac{d\mathbf{v}}{dt} \right) - \frac{d}{dt} \left(\nabla^T \mathbf{v} \right)$$

Numerically, we have several choices to compute this term, they all might differe depending on the approximatio nquality of the differential operators.

- Variant 1: The formal way from the equation above yields
$$\Phi\left(\mathbf{v}\right) = \left(u_x u_x + v_x u_y + w_x u_z\right) + \left(u_y v_x + v_y v_y + w_y v_z\right) + \left(u_z w_x + v_z w_y + w_z w_z\right)$$
- Variant 2: We try to isolate the divergence of velocity by
$$\Phi\left(\mathbf{v}\right) = 2(v_x u_y - u_x v_y) + 2(w_x u_z - u_x w_z) + 2(w_y v_z - v_y w_z) + \left(\nabla^T \mathbf{v}\right)^2$$
- Variant 3: take the divergence of the stationary part of the substantial velocity
$$\Phi\left(\mathbf{v}\right) = \nabla^T \left(u\mathbf{v}_x + v\mathbf{v}_y + w\mathbf{v}_z\right) - u(\nabla^T \mathbf{v})_x - v(\nabla^T \mathbf{v})_y - w(\nabla^T \mathbf{v})_z$$
- Variant 4: take the divergence of the true substantial derivative of the velocity (use $\frac{d\mathbf{v}}{dt}$ as computed due to FLIQUID_ConsistentPressure_Version)
$$\Phi\left(\mathbf{v}\right) \approx \nabla^T \left(\frac{d\mathbf{v}}{dt} \right) - \frac{\nabla^T \mathbf{v}^{n+1} - \nabla^T \mathbf{v}^n}{\Delta t}$$

Especially variants 2 and 3 explicitly contain terms with the divergence of velocity $\nabla^T \mathbf{v}$. Even for incompressible flows, the numerical evaluation of this term will not entirely be zero. In order to neglect the divergence anyways, use the option FLIQUID_ConsistentPressure_UseDivV . The hope is to gain smoothness of the pressure solutiuon.

Another degree of freedom is given by the fact, that we can express the differential operators fpr x-, y-, z- derivatrives by the two options

Classical :
which is the typical way of gradient/derivative approximation.
Derived :
here, the laplace together with the distance terms is used, the order of approximation is usually one less than the approximation order of .

If variants 1 ... 4 are computed with the **classical** derivative operators, we have 4 more variants:

- variant 5: same as variante 1 computed with the **derived** operators
- variant 6: same as variante 2 computed with the **derived** operators

- variant 7: same as variante 3 computed with the **derived** operators
- variant 8: same as variante 4 computed with the **derived** operators

---

ComputationOfPSI

how to numerically compute the source term that goes with the viscous forces

From DerivePoissonEquationForPressure we have a formulation for $\Phi(\mathbf{v})$, which is

$$\Psi(\mathbf{v}) \equiv \nabla^T \left(\frac{1}{\rho} \nabla \mathbf{S}_v(\mathbf{v}) \right)$$

In the same fashion as in ComputationOfPHI , we have the **classical** and the **derived** differential operators for the divergence operation needed for Ψ .

Therefore, numerically, we have the two choices

- Variant 1: compute $\Psi(\mathbf{v}) = \nabla^T(...)$ with the **classical** differential operators
- Variant 2: compute $\Psi(\mathbf{v}) = \nabla^T(...)$ with the **derived** differential operators

---

ComputationOfTHETA

how to numerically compute the source term due to the Darcy forces

We see in DerivePoissonEquationForPressure , HydrostaticPressure , and DynamicPressure that the source term due to the Darcy-forces is subdivided into its hydrostatic and dynamic parts.

For this we provide 4 different variants, which are meant for experimenting. The Darcy-term has the property, that it might produce huge acceleration forces at very
low velocities, as β , the Darcy constant $\tilde{\beta}$ divided by ρ , becomes big (which is naturally possible).
So, the numerics is very sensitive in these cases, and a final general stability condition could not yet be determined.

- Variant 1: the original and numerically most natural version

$$\Theta_{hyd}^{n+1} = \nabla^T(\beta \cdot \mathbf{v}_\beta)$$

$$\Theta_{dyn}^{n+1} = \nabla^T\left(\beta \cdot \mathbf{v}^{n+1}\right)$$

- Variant 2: bring the Darcy-contributions mainly to the hydrostatic part of the pressure

$$\Theta_{hyd}^{n+1} = \nabla^T(\beta \cdot (\mathbf{v}_\beta - \mathbf{v}^n))$$

$$\Theta_{dyn}^{n+1} = \nabla^T\left(\beta \cdot (\mathbf{v}^{n+1} - \mathbf{v}^n)\right) \approx 0$$

- Variant 3: bring the Darcy-contributions mainly to the dynamic part of the pressure

$$\Theta_{hyd}^{n+1} = 0$$

$$\Theta_{dyn}^{n+1} = \nabla^T\left(\beta \cdot (\mathbf{v}^{n+1} - \mathbf{v}_\beta)\right)$$

- Variant 4: assume perfect adaption of the fluid velocity to the Darcy basis velocity, i.e. assume $\left(\mathbf{v}^{n+1} - \mathbf{v}_\beta\right)) = 0$:

$$\Theta_{hyd}^{n+1} = 0$$

$$\Theta_{dyn}^{n+1} = 0$$

The definition of the DarcyVersion is done based on %ind_DarcyVersion% .

RegularizeDPA

regularize the computation of dynamic pressure in order to reduce fluctuations

The ClassicalDPA as well as the AlternativeDPA lead to tremendous pressure fluctuations over time, that can be observed especially for simulations in long channels, where only at one end there is a Dirichlet-condition, whereas as all the other walls are modelled by Neumann-type conditions.
The problem here, most probably, comes from the fact, that the pressure Poisson equation stems from the equation of momentum where we have a formulation on the gradient of pressure (overdetermined system, containing 3*N equations for the N unknown pressure values). By application of the divergence operator, see DerivePoissonEquationForPressure , we obtain N equations for N unknown pressure values, but we most probably loose information. This information loss might be the reason for the pressure fluctuations.

In general, the numerical discretization of the Pressure Poisson equation is given by

$$\sum_{j=1}^{N(i)} W_{ij} \left(\frac{1}{2} \frac{\rho_i + \rho_j}{\rho_i \rho_j} \left(p_j - p_i \right) \right) = Q_i$$

Such type of equation also arises, if we do not apply the divergence operator to the equation of momentum, instead we apply an arbitrary,
locally choosen vector $\mathbf{q}_i$ to the equation of momentum, i.e.

$$\frac{1}{\rho_i} \mathbf{q}_i^T \nabla p_i = \frac{1}{\rho_i} \mathbf{q}_i^T \left(\nabla^T \mathbf{S}_v \left(\mathbf{v} \right) \right)_i^T - \beta \cdot \mathbf{q}_i^T \mathbf{v}_i - \mathbf{q}_i^T \dot{\mathbf{v}}_i = Q_i^{reg}$$

Discretize this equation in the FPM sense, i.e.

$$\sum_{j=1}^{N(i)} \frac{1}{\rho_i} \left(q_i^x \cdot c_{ij}^x + q_i^y \cdot c_{ij}^y + q_i^z \cdot c_{ij}^z \right) \cdot p_j = Q_i^{reg}$$

In fact, both equations aim to give an answer to the pressure. So, we could just add the enhancement, such that the final, regularized linear system of equations is

$$\sum_{j=1}^{N(i)} W_{ij} \left(\frac{1}{2} \frac{\rho_i + \rho_j}{\rho_i \rho_j} \left(p_j - p_i \right) \right) + \sum_{j=1}^{N(i)} \frac{1}{\rho_i} \left(q_i^x \cdot c_{ij}^x + q_i^y \cdot c_{ij}^y + q_i^z \cdot c_{ij}^z \right) \cdot p_j = Q_i + Q_i^{reg}$$

This regularization provides additional information on the slope / gradient of the pressure, whereas the original Poisson equation provides only information about the curvature of the pressure.

The length of $\mathbf{q}_i$ determines the weight of the regularization.
The observation so far is, that best result are obtained by $\|\mathbf{q}_i\| < 0.1$
Useful choices of the direction of $\mathbf{q}_i$ could be the velocity, the direction of the pressure gradient itself, or the directions perpendicular to
the pressure gradient. We have implemented a collection in the sense

$$\mathbf{q}_i = c_m \cdot \frac{\mathbf{v}_i}{\|\mathbf{v}_i\|} + c_n \cdot \frac{\nabla p_i}{\|\nabla p_i\|} + c_a \cdot \frac{\mathbf{a}_i}{\|\mathbf{a}_i\|} + c_b \cdot \frac{\mathbf{b}_i}{\|\mathbf{b}_i\|}$$

where the last two vectors $\mathbf{a}_i, \mathbf{b}_i$ are perpendicular to the pressure gradient ∇p_i .

The choice of $\mathbf{q}_i$ can be controlled by the user:

- c_m -> adjust FLIQUID_ConsistentPressure_CoeffMM
- c_n -> adjust FLIQUID_ConsistentPressure_CoeffNN
- c_a -> adjust FLIQUID_ConsistentPressure_CoeffTT (in fact, from a and b, MESHFREE computes a random vector perpendicular to the gradient of pressure)

- c_b -> adjust FLIQUID_ConsistentPressure_CoeffTT

HydrostaticPressureAlgorithm

The hydrostatic pressure algortihm solves the poisson equation derived in DerivePoissonEquationForPressure .

The numerical representation of the equation is

$$-\frac{1}{\Delta t^2}\frac{1}{\rho}\frac{\partial \rho}{\partial p}\cdot \tilde{p}^{n+1}_{hyd,i} + \sum_{j=1}^{N} c_{ij}^{\nabla^T(\frac{1}{\rho}\nabla)}\cdot \tilde{p}^{n+1}_{hyd,j} = -\frac{1}{\Delta t^2}\frac{1}{\rho}\frac{\partial \rho}{\partial p} p^{n}_{hyd,i} + \nabla^T\left(\frac{1}{\rho}\nabla \mathbf{S}^n_s\right)_i + \nabla^T \mathbf{g}_i$$

KepsilonAlgorithm

turbulence modelling using the k-epsilon model

The K-epsilon turbulence model is one of the common models for including turbulence into a CFD simulation.
In MESHFREE , it can be incorporated for chambers of LIQUID and GASDYN type in the KindOfProblem definition.
For all boundary elements also boundary conditions BC_eps and BC_k must be defined.
Also positive initial values for k and epsilon must be chosen, e.g.

```
INITDATA ( $MAT$ ,%ind_k%) = 0.0001
INITDATA ( $MAT$ ,%ind_eps%) = 0.1
```

More Information

See DOCUMATH_NumericalIntegrationOfTurbulence.pdf for a detailed discussion of how MESHFREE incorporates the k-epsilon turbulence model.
For some specific derivation of the heat source triggered by turbulence, see DOCUMATH_DerivationOfEnergyEquationWithTurbulence.pdf .
Relevant Indices

- %ind_k%
- %ind_eps%
- %ind_NUE_turb%
- %ind_ETA_eff%

PreparationAlgorithm

This subroutine provides all information about the auxiliary points of point pairs close to the boundary.
This is necessary for determining function values at the auxiliary points which are needed
for reconstruction techniques (MUSCL) in the Eulerian framework.
Therefore this subroutine detects all points close to the boundary and checks if the auxiliary points are
out of domain by computing the distance to boundary.

StressTensorAlgorithm

update the solid stress tensor towards the next time level

We assume, that during the time step, the values of velocity and shear modulus do not change.
In fact, we numerically integrate

$$\frac{d\boldsymbol{S}_s}{dt} = \mu \cdot \left(\nabla \boldsymbol{v}^T + \left(\nabla \boldsymbol{v}^T \right)^T \right) + \boldsymbol{K}\boldsymbol{S}_s - \boldsymbol{S}_s\boldsymbol{K}$$

where $\boldsymbol{K}$ is an antisymmetric rotation matrix.
For isotropic materials, i.e. for scalar values of mue, we find an analytical solution to this problem.
See DOCUMATH_StressTensor_TimeIntegration.pdf for a detailed discussion of
the stress tensor integration.

Remark for applications with yield stress (see mue ,):
Here, we have the numerical difficulty, that at reaching the yield stress, the effective
shear modulus decreases considerably. That means, to possible inner stress gradients (leading to acceleration),
there is only small material resistance. Numerically, we would like to avoid sudden velocity increase/jumps
due to this fact.
A small stabilty analysis of the VelocityAlgorithm , considering only the terms of $\hat{\boldsymbol{g}}$ and the
effective viscous stresses governed by $\hat{\eta}$, yields the following: if requiring only small changes of velocity, then
we find

$$\hat{\eta} = O\left(\|\boldsymbol{S}_s\| \frac{h}{\|\boldsymbol{v}\|} \right)$$

So, it is maybe a good idea to require the numerics to always provide

$$\hat{\eta} = C_{\hat{\eta}} \cdot \|\boldsymbol{S}_s\| \frac{h}{\|\boldsymbol{v}\|}$$

with $C_{\hat{\eta}}$ possibly small, but big enough to keep stability.
If the effective viscosity is only made up by the shear modulus, we have a resulting condition

$$\Delta t \cdot \mu = C_\mu \cdot \|\boldsymbol{S}_s\| \frac{h}{\|\boldsymbol{v}\|}$$

In other words:

$$\mu = C_\mu \cdot \|\boldsymbol{S}_s\| \frac{h}{\|\boldsymbol{v}\|} \frac{1}{\Delta t}$$

As a simplifying approach, we can assume that $\frac{h}{\|\boldsymbol{v}\|}$ and Δt have the same/comparable size.
Then, the equation above simplifies to

$$\mu = C_\mu \cdot \|\boldsymbol{S}_s\|$$

There is ongoing research on this topic.

TemperatureAlgorithm

This algorithm solves the conservation of energy stated in EquationsToSolve .

$$T_i^{n+1} - \frac{\Delta t}{\rho \cdot c_v} \cdot \sum_{j=1}^{N} c_{ij}^{\nabla^T(\lambda \cdot \nabla)} \cdot T_j^{n+1} = T_i^n + \frac{\Delta t}{\rho \cdot c_v} \cdot q$$

This algorithm solves the conservation of energy stated in EquationsToSolve .
Version 2 provides especially Eulerian framework, using intermediate time steps and second order time integration.

$$T_i^{n+1} - \frac{\Delta t}{\rho \cdot c_v} \cdot \sum_{j=1}^{N} c_{ij}^{\nabla^T(\lambda \cdot \nabla)} \cdot T_j^{n+1} = T_i^n + \frac{\Delta t}{\rho \cdot c_v} \cdot q$$

TimeIntegrationAlgorithm

This algorithm computes the right hand side for the linearized system,
where the transport term is approximated by using the MUSCL-reconstruction.

# #

This is only for the computation of velocity and pressure using solve_V_2.

This algorithm solves the semi-discrete ODE -system

$$\dot{\mathbf{U}}(t) = \mathbf{F}(t, \mathbf{U}(t))$$

with the implicit SDIRK2 method which is of order 2.
The ODE -system comes from the spatial discretization of scalar entities like temperature, k and epsilon(turbulence model), etc.

This algorithm solves the semi-discrete ODE -system

$$\dot{\mathbf{U}}(t) = \mathbf{F}(t, \mathbf{U}(t))$$

with the implicit Euler method which is of order 1.
The ODE -system comes from the spatial discretization of scalar entities like temperature, k and epsilon(turbulence model), etc.

This algorithm computes the transport operator stencils for the linearized system,
where the transport term is approximated by using the MUSCL-reconstruction.
Please note that the stencils are independent of reconstructions.
Due to the linearization the reconstructions are exclusively on the right hand side.

# #

This algorithm computes the right hand side for the linearized system,
where the transport term is approximated by using the MUSCL-reconstruction.

# #

This is only for scalar entities like temperature, k and epsilon(turbulence model), etc.

VelocityAlgorithm

List of members:

ALE	ALE
Lagrange	Lagrange

ALE

This algorithm solves the conservation of momentum stated in EquationsToSolve .
Version 2 provides especially Eulerian framework, using intermediate time steps and second order time integration.

$$\tilde{\mathbf{v}}_i^{n+1} - \frac{\Delta t}{\rho} \sum_j c_{ij}^{\nabla^T(\hat{\eta}\nabla)} \cdot \tilde{\mathbf{v}}_j^{n+1} + \Delta t \cdot \beta \cdot \tilde{\mathbf{v}}_i^{n+1} = \mathbf{v}_i^n - \frac{\Delta t}{\rho} \left(\nabla \tilde{p}_{hyd}^{n+1}\right)_i - \frac{\Delta t}{\rho} \left(\nabla \tilde{p}_{dyn}^n\right)_i + \Delta t \cdot \hat{\mathbf{g}}_i^n$$

List of members:

SDIRK2	SDIRK2
ImplicitEuler	ImplicitEuler

MESHFREE · Solvers · Numerics · LIQUID · Algorithms · VelocityAlgorithm · ALE · ImplicitEuler

ImplicitEuler

This algorithm solves the semi-discrete ODE -system

$$\dot{\mathbf{U}}(t) = \mathbf{F}(t, \mathbf{U}(t))$$

with the implicit Euler method which is of order 1:

$$\mathbf{U}^{k+1} = \mathbf{U}^k + \Delta t\, \mathbf{F}\left(t^k + \Delta t,\, \mathbf{U}^{k+1}\right)$$

The ODE -system comes from the spatial discretization of velocity and pressure in solve_V_2.
Both vp- and v-- can be solved.

The time integration scheme can be controlled by time_integration_impl and resp. for the velocity by time_integration_impl_solve_v .

MESHFREE · Solvers · Numerics · LIQUID · Algorithms · VelocityAlgorithm · ALE · SDIRK2

SDIRK2

This algorithm solves the semi-discrete ODE -system

$$\dot{\mathbf{U}}(t) = \mathbf{F}(t, \mathbf{U}(t))$$

with the implicit SDIRK2 method which is of order 2:

$$\boldsymbol{\eta}_1 = \mathbf{U}^k + \Delta t\, \alpha\, \mathbf{F}\left(t^k + \alpha\, \Delta t,\, \boldsymbol{\eta}_1\right),$$

$$\mathbf{U}^{k+1} = \mathbf{U}^k + \Delta t \left((1-\alpha)\, \mathbf{F}\left(t^k + \alpha\, \Delta t,\, \boldsymbol{\eta}_1\right) + \alpha\, \mathbf{F}\left(t^k + \Delta t,\, \mathbf{U}^{k+1}\right)\right)$$

$$\alpha = 1 - \frac{\sqrt{2}}{2}$$

The ODE -system comes from the spatial discretization of velocity and pressure in solve_V_2.
Both vp- and v-- can be solved.

The time integration scheme can be controlled by time_integration_impl and resp. for the velocity by time_integration_impl_solve_v .

MESHFREE · Solvers · Numerics · LIQUID · Algorithms · VelocityAlgorithm · Lagrange

Lagrange

This algorithm solves the conservation of momentum stated in EquationsToSolve .

$$\tilde{\mathbf{v}}_i^{n+1} - \frac{\Delta t}{\rho} \sum_j c_{ij}^{\nabla^T(\hat{\eta}\nabla)} \cdot \tilde{\mathbf{v}}_j^{n+1} + \Delta t \cdot \beta \cdot \tilde{\mathbf{v}}_i^{n+1} = \mathbf{v}_i^n - \frac{\Delta t}{\rho} \left(\nabla \tilde{p}_{hyd}^{n+1}\right)_i - \frac{\Delta t}{\rho} \left(\nabla \tilde{p}_{dyn}^{n}\right)_i + \Delta t \cdot \hat{\mathbf{g}}_i^n$$

VolumeCorrection

Volume Correction Algorithms in MESHFREE

Motivation

The MESHFREE method is not inherently mass and volume conservative, (only approximatively) hence there are extra measures taken to
improve the conservative properties. There are currently two approaches for volume correction available:

- global volume correction, here referred to as classical, cv Parameter RepresentativeMass_iData = 0
- localised volume correction by RepresentativeMassAlgorithm , RepresentativeMass_iData not equal 0

Pointcloud Motion

Currently, these are available depending on the MotionOfPointcloud :

Pointcloud Motion	classical	RepresentativeMassAlgorithm
LAGRANGE	X	X
EULER	X	
EULERIMPL	X	

Parameter

To control the strength of the volume correction, several parameters are available:

cv Parameter	classical	RepresentativeMassAlgorithm
VOLUME_correction	X	
VOLUME_correction_FreeSurface	X	X
VOLUME_correction_local		X

Currently, the use of any combination of VOLUME_correction, VOLUME_correction_FreeSurface, VOLUME_correction_local may yield to instabilities, so we recommend using only one of these. Stabilizing this is an open topic of research.

List of members:

RepresentativeMassAlgorithm	How to distribute prepresentative masses to MESHFREE points

RepresentativeMassAlgorithm

How to distribute prepresentative masses to MESHFREE points

Motivation

Inherently, MESHFREE points do not carry mass as they are information carriers only. That gives a lot of freedom regarding adaptive refinement and much more.
That also means that MESHFREE is not inherently mass conservative, but we provide strategies for conserving mass that act locally.

Basic Idea

Within MESHFREE , mass can only be produced at inflow boundaries or DropletSource items. Mass can only be reduced at outflow boundary elements or by EVENT statements.
Besides that, mass cannot be generated. That means that the total mass can be determined analytically by measuring the mass being brought into and out of the system.
The idea is to distribute mass packages among the MESHFREE points. The sum of all masses shall represent the analytical mass to be in the system.
Mass can be re-distributed among points, which will become necessary in regions of local refinement/coarsening.

Parameters

The RepresentativeMassAlgorithm is triggered by common_variables parameter RepresentativeMass_iData .

Algorithm

First, please see DefinitionRepresentativeMass (%ind_mi_rep%) and DefinitionRepresentativeDensity (%ind_r_rep%).
The algorithm is sketched here:

- algorithm DeletedOrInactivePoints is launched after point deletion in order to project back the representative masses of the vanishing points onto the active part of the boundary

After the end of point cloud organization

- launch DeletedOrInactivePoints for all recently deactivated boundary points onto the remaining active part of boundary
- launch NewPoints for all new points or newly activated boundary points
- launch FlowPenetrationBoundaries in order to update representative mass at inflow, outflow, or permeable walls
- iterate Smoothing for a given number of iteration loops (to be defined in RepresentativeMass_iData)
- finally, launch again DeletedOrInactivePoints in order to remove negative masses in active points, that might have occured during the previous steps

Relevant Indices

%ind_mi_rep% , %ind_r_rep% , %ind_Vi% , %ind_BNDfree_defect% , %ind_cluster%

List of members:

DefinitionRepresentativeMass	define the representative mass
DefinitionRepresentativeDensity	define the representative density
Smoothing	smooth the representative masses in order to obtain smooth representative density
NewPoints	newly created points acquiring mass from existing points
DeletedOrInactivePoints	deleted or inactive points giving away their mass to active points
FlowPenetrationBoundaries	adapt mass of flow-penetrated boundaries

DefinitionRepresentativeDensity

define the representative density

The representative density is computed from the representative masses and the appropriate volumina of the MESHFREE points (given by %ind_Vi%).

Version 1:

$$\widehat{\rho}_i = \frac{\sum_j W_{ij} \cdot \widehat{m}_j}{\sum_j W_{ij} \cdot \widehat{V}_j}$$

where the weight kernel is defined as

$$W_{ij} = \exp\left(-\alpha_W \cdot r_{ij}^2\right) / \widehat{V}_j^{\beta_W} \quad \text{with } r_{ij} = \frac{\left\|\mathbf{x}_j - \mathbf{x}_i\right\|}{\frac{1}{2}\left(h_j + h_i\right)}$$

And $\alpha_W > 0$, $\beta_W = 0, 1$ to be given by the user.
Remark:

- $\beta = 0$: average mass divided by average volume $\widehat{\rho}_i = \frac{\sum_j \exp\left(-\alpha_W \cdot r_{ij}^2\right) \cdot \widehat{m}_j / \sum_j \exp\left(-\alpha_W \cdot r_{ij}^2\right)}{\sum_j \exp\left(-\alpha_W \cdot r_{ij}^2\right) \cdot \widehat{V}_j / \sum_j \exp\left(-\alpha_W \cdot r_{ij}^2\right)}$ (the default!!!)
- $\beta = 1$: average local representative density, i.e. $\widehat{\rho}_i = \frac{\sum_j \exp\left(-\alpha_W \cdot r_{ij}^2\right) \cdot \frac{\widehat{m}_j}{\widehat{V}_j}}{\sum_j \exp\left(-\alpha_W \cdot r_{ij}^2\right)}$

Version 2:
Determine the representative density clusterwise by

$$\widehat{\rho}_i = \frac{\sum_{j \in J_{cluster}} \widehat{m}_j}{\sum_{j \in J_{cluster}} \widehat{V}_j}$$

which leads to less local fluctuations of the representative density.

MESHFREE · Solvers · Numerics · LIQUID · Algorithms · VolumeCorrection · RepresentativeMassAlgorithm · DefinitionRepresentativeMass

DefinitionRepresentativeMass

define the representative mass

Representative mass of a MESHFREE point with index i is denoted by

$\widehat{m}_i$

MESHFREE · Solvers · Numerics · LIQUID · Algorithms · VolumeCorrection · RepresentativeMassAlgorithm · DeletedOrInactivePoints

DeletedOrInactivePoints

deleted or inactive points giving away their mass to active points

Also here, the Smoothing algorithm is appropriate by setting $\widehat{V}_i = 0$, has we find the mass exchange by

$$\zeta_i = \frac{\gamma_i \cdot \sum_j W_{ij} \cdot \widehat{m}_j - \widehat{m}_i}{\sum_j K_{ij} \widehat{m}_j + \gamma_i \cdot \sum_j W_{ij} \cdot K_{ij} \widehat{m}_j} = \frac{-\widehat{m}_i}{\sum_j K_{ij} \widehat{m}_j}$$

We additionally have to set $W_{ij} = K_{ij} = 0$ if j is an index of a disappearing point.
Given this, it suffices to run just one iteration for convergence.

FlowPenetrationBoundaries

adapt mass of flow-penetrated boundaries

Boundary points of a boundary penetrated by flow, have to adapt their representative mass simply by:

$$\frac{d\widehat{m}_i}{dt} = \rho_i \left(\mathbf{n}_i^T \cdot \mathbf{v}_i\right) A_i$$

where A_i is the area occupied by the boundary point to be found in %ind_dA% .

NewPoints

newly created points acquiring mass from existing points

Here, we can also use the Smoothing algorithm, with additionally setting $\widehat{m}_i = 0$, thus solving the mass exchange by

$$\zeta_i = \frac{\sum\limits_j W_{ij} \cdot \widehat{m}_j}{\sum\limits_j K_{ij}\widehat{m}_j/\gamma_i \; + \sum\limits_j W_{ij} \cdot K_{ij}\widehat{m}_j}$$

We additionally have to set $W_{ij} = K_{ij} = 0$ if j is an index of a new point.
Given this, it suffices to run just one iteration for convergence.

Smoothing

smooth the representative masses in order to obtain smooth representative density

Given the definition of the representative density (DefinitionRepresentativeDensity), we search for the eigenfunction

$$\widehat{\rho}_i = \frac{\widehat{m}_i}{\widehat{V}_i}$$

That is try to adapt the representative masses by $\Delta\widehat{m}_i$ such that

$$\widehat{\rho}_i^{smooth} = \frac{\sum\limits_j W_{ij} \cdot \left(\widehat{m}_j - \Delta\widehat{m}_j\right)}{\sum\limits_j W_{ij} \cdot \widehat{V}_j} = \frac{\widehat{m}_j + \sum\limits_j \Delta\widehat{m}_j}{\widehat{V}_i}$$

with the requirement that

$$\sum_i \Delta\widehat{m}_i = 0$$

and the optimality constraint

$$\sum_i \left|\Delta\widehat{m}_i\right| = \min$$

that means the sum of all particular mass changes on the MESHFREE points has to be zero in order not to generate or dissolve mass by smoothing, and that the mass adaption changes are possibly small.

This is a big optimization problem of the N unknowns $\Delta\widehat{m}_i \; i = 1...N$. The solution would be very costly, so we propose an iteration procedure:

Version 1:
The equation above can be solved pointwise by defining mass packages $\Delta\widehat{m}_{ij}$ that go over from point j to point i :

$$\widehat{\rho}_i^{smooth} = \frac{\sum_j W_{ij} \cdot \left(\widehat{m}_j - \Delta\widehat{m}_{ij}\right)}{\sum_j W_{ij} \cdot \widehat{V}_j} = \frac{\widehat{m}_i + \sum_j \Delta\widehat{m}_{ij}}{\widehat{V}_i}$$

together with the ansatz

$$\Delta\widehat{m}_{ij} = \zeta_i K_{ij} \cdot \widehat{m}_j \text{ with } K_{ij} = \exp\left(-\alpha_K \cdot r_{ij}^2\right) / \widehat{V}_j^{\beta_K}$$

As mentioned above, $\Delta\widehat{m}_{ij}$ is the little mass portion, given away from point j to point i .

We find the unknown ζ_i by

$$\widehat{\rho}_i^{smooth} = \frac{\sum_j W_{ij} \cdot \left(\widehat{m}_j - \zeta_i K_{ij} \widehat{m}_j\right)}{\sum_j W_{ij} \cdot \widehat{V}_j} = \frac{\widehat{m}_i + \sum_j \zeta_i K_{ij} \widehat{m}_j}{\widehat{V}_i} \;\Rightarrow$$

$$\gamma_i \cdot \sum_j W_{ij} \cdot \left(\widehat{m}_j - \zeta_i K_{ij} \widehat{m}_j\right) = \left(\widehat{m}_i + \sum_j \zeta_i K_{ij} \widehat{m}_j\right) \;\Rightarrow$$

$$\gamma_i \cdot \sum_j W_{ij} \cdot \widehat{m}_j - \widehat{m}_i = \zeta_i \sum_j K_{ij} \widehat{m}_j + \zeta_i \gamma_i \cdot \sum_j W_{ij} \cdot K_{ij} \widehat{m}_j \;\Rightarrow$$

$$\zeta_i = \frac{\gamma_i \cdot \sum_j W_{ij} \cdot \widehat{m}_j - \widehat{m}_i}{\sum_j K_{ij} \widehat{m}_j + \gamma_i \cdot \sum_j W_{ij} \cdot K_{ij} \widehat{m}_j} \;\Rightarrow$$

$$\zeta_i = \frac{\sum_j W_{ij} \cdot \widehat{m}_j - \widehat{m}_i / \gamma_i}{\sum_j K_{ij} \widehat{m}_j / \gamma_i + \sum_j W_{ij} \cdot K_{ij} \widehat{m}_j}$$

Above, we use the abbreviation $\gamma_i \equiv \frac{\widehat{V}_i}{\sum_j W_{ij} \cdot \widehat{V}_j}$ as the ratio of point volume and smoothed point volume.
From the pointwise corrections we find the global correction

$$\Delta\widehat{m}_i = \sum_j \Delta\widehat{m}_{ij} - \Delta\widehat{m}_{ji}$$

In this sense, the global sum of all mass changes is zero, which guarantees mass conservation.

Version 2:
We require

$$\rho_i = \frac{\sum_j W_{ij} \cdot \left(\widehat{m}_j - \Delta\widehat{m}_{ij}\right)}{\sum_j W_{ij} \cdot \widehat{V}_j}$$

together with the ansatz

$$\Delta\widehat{m}_{ij} = \zeta_i K_{ij} \cdot \widehat{m}_j \text{ with } K_{ij} = \exp\left(-\alpha_K \cdot r_{ij}^2\right) / \widehat{V}_j^{\beta_K}$$

Thus, we have

$$\zeta_i = \frac{\sum_j W_{ij} \widehat{m}_j - \sum_j W_{ij} \widehat{V}_j \rho_i}{\sum_j W_{ij} K_{ij} \widehat{m}_j}$$

Version 3:
We require

$$\widehat{m}_i + \sum_j \Delta\widehat{m}_{ij} = \rho_i \widehat{V}_i$$

again with the ansatz

$$\Delta\widehat{m}_{ij} = \zeta_i K_{ij} \cdot \widehat{m}_j \text{ with } K_{ij} = \exp\left(-\alpha_K \cdot r_{ij}^2\right) / \widehat{V}_j^{\beta_K}$$

and hence it follows

$$\zeta_i = \frac{\rho_i \widehat{V}_i - \widehat{m}_i}{\sum_j K_{ij} \widehat{m}_j}$$

Version 4:
We try to locally exchange masses in order to equalize the current representative density. I.e. we locally exchange the masses $\widehat{m}_{ij}$ such that the averaged representative density is achieved. That is

$$\frac{\sum_{j\neq i} \left(\widehat{m}_j - \widehat{m}_{ij}\right) W_{ij} + \left(\widehat{m}_i + \sum_{j\neq i} \widehat{m}_{ij}\right) W_{ii}}{\sum_j \widehat{V}_j W_{ij}} = \frac{\sum_j \widehat{\rho}_j W_{ij}}{\sum_j W_{ij}}$$

which means

$$\frac{\sum_{j\neq i} \left(\widehat{m}_j - \zeta_i K_{ij} \widehat{m}_j\right) W_{ij} + \left(\widehat{m}_i + \sum_{j\neq i} \zeta_i K_{ij} \widehat{m}_j\right) W_{ii}}{\sum_j \widehat{V}_j W_{ij}} = \frac{\sum_j \widehat{\rho}_j W_{ij}}{\sum_j W_{ij}}$$

and leads to

$$\zeta_i = \left(\frac{\sum_j \widehat{\rho}_j W_{ij}}{\sum_j W_{ij}} \frac{\sum_j \widehat{V}_j W_{ij}}{\sum_j W_{ij}} - \frac{\sum_j \widehat{m}_j W_{ij}}{\sum_j W_{ij}} \right) \frac{\sum_j W_{ij}}{\sum_j K_{ij} \widehat{m}_j \left(W_{ii} - W_{ij}\right)}$$

In general: the most stable behavior is produced with version 3. Versions 1, 2, and 4 often run into strange fixed points (i.e. eigenfunctions) of mass distribution -> to be further investigated .
Choose the version to be employed by RepresentativeMass_iData !

EquationsToSolve

differential equations to be solved by MESHFREE

Conservation of mass:

$$\dot{\rho} = -\rho \left(\nabla^T \mathbf{v}\right)$$

Conservation of momentum:

$$\dot{\mathbf{v}} + \frac{1}{\rho}\nabla p = \frac{1}{\rho}\left(\nabla^T \mathbf{S}_s\right)^T + \frac{1}{\rho}\left(\nabla^T \mathbf{S}_v\left(\mathbf{v}\right)\right)^T + \mathbf{g} - \beta \cdot \left(\mathbf{v} - \mathbf{v}_\beta\right)$$

Conservation of energy

$$\rho c_v \dot{T} = \nabla^T \left(\lambda \cdot \nabla T\right) + q$$

The variables are

- ∇ => gradient operator
- ρ => density, see %ind_r%
- $\mathbf{v} = (u, v, w)^T$ => velocity, see %ind_v(1)% , %ind_v(2)% , %ind_v(3)%
- p => pressure, see %ind_p% and %ind_p_dyn%
- $\mathbf{g}$ => gravity / body forces, see %ind_g(1)% , %ind_g(2)% , %ind_g(3)%

- T => temperature, see %ind_T%
- c_v => heat capacity, see %ind_CV%
- λ => heat conductivity, see %ind_LAM%
- q => heat sources, given by external heat sources and internal processes (viscous heating), see %ind_diss%
- η_{eff} => effective viscosity, might consist of laminar and tubulent partitions, see %ind_ETA_eff% , %ind_ETA% , %ind_ETA_sm% .
- $\mathbf{S}_v(\mathbf{v}) = \eta_{\text{eff}} \cdot \left(\left(\nabla \mathbf{v}^T\right) + \left(\nabla \mathbf{v}^T\right)^T - \frac{2}{3} \nabla^T \mathbf{v} \cdot \mathbf{I} \right)$ => viscous stress tensor
- $\mathbf{S}_s$ => solid stress tensor, refer to StressTensorAlgorithm , see also %ind_Sxx% , %ind_Sxy% , %ind_Sxz% , %ind_Syy% , %ind_Syz% , %ind_Szz%
- β => The Darcy / Brinkman constant $\beta = \frac{\tilde{\beta}}{\rho}$, see DarcyConstant
- $\mathbf{v}_\beta$ => basis velocity of porous material, see DarcyBasisVelocity

For the solution algorithm it is important to be aware of the following two remarks:

DeriveDivergenceOfVelocity
DerivePoissonEquationForPressure

List of members:

DerivePoissonEquationForPressure	how to compute the pressure from the equation of momentum
DeriveDivergenceOfVelocity	how to compute the divergence of velocity from mass conservation

DeriveDivergenceOfVelocity

how to compute the divergence of velocity from mass conservation

The divergence of the velocity can be computed by considering the equation of mass conservation:

$$\dot{\rho} = -\rho \left(\nabla^T \mathbf{v}\right)$$

Out of this, it follows

$$\begin{aligned} \left(\nabla^T \mathbf{v}\right)^{n+1} &= -\tfrac{d}{dt} \left(\log\left(\rho^{n+1}\right)\right) \\ &\approx -\tfrac{1}{\Delta t} \left(\log\left(\rho^{n+1}\right) - \log\left(\rho^n\right)\right) \end{aligned}$$

For numerical reasons it is preferable to define the intermediate density

$$\rho_{dyn}^n \equiv \rho\left(t^n, p_{hyd}^n + p_{dyn}^n, A_v^{n-1}\right)$$

That means the density as it is given after the computation of the velocity, hydrostatic and dynamic pressure, but before the computation of all additional variables A_v^n .
Hence, we rewrite the formulation of the divergence of velocity by

$$\left(\nabla^T \mathbf{v}\right)^{n+1} \approx -\frac{1}{\Delta t} \left(\log\left(\rho_{dyn}^{n+1}\right) - \log\left(\rho_{dyn}^n\right)\right)$$

Splitting this equation into a hydrostatic and a dynamic part yields

$$\begin{aligned} \left(\nabla^T \mathbf{v}\right)^{n+1} &\approx -\tfrac{1}{\Delta t} \left(\log\left(\rho_{dyn}^{n+1}\right) - \log\left(\rho_{dyn}^n\right)\right) \\ &\approx -\tfrac{1}{\Delta t} \left(\log\left(\rho\left(t^{n+1}, p_{hyd}^n + p_{dyn}^{n+1}, A_v^n\right)\right) + \tfrac{1}{\rho}\tfrac{\partial \rho}{\partial p}\left(p_{hyd}^{n+1} - p_{hyd}^n\right) - \log\left(\rho_{dyn}^n\right)\right) \\ &\approx -\tfrac{1}{\Delta t} \left(\log\left(\rho\left(t^{n+1}, p_{hyd}^n + p_{dyn}^{n+1}, A_v^n\right)\right) - \log\left(\rho_{dyn}^n\right)\right) - \tfrac{1}{\Delta t}\left(\tfrac{1}{\rho}\tfrac{\partial \rho}{\partial p}\left(p_{hyd}^{n+1} - p_{hyd}^n\right)\right) \\ &\approx \left(\nabla^T \mathbf{v}\right)_{dyn}^{n+1} + \left(\nabla^T \mathbf{v}\right)_{hyd}^{n+1} \end{aligned}$$

Thus, the definitions for hydrostatic and dynamic compression rates follow as

$$\left(\nabla^T \mathbf{v}\right)_{hyd}^{n+1} \equiv -\frac{1}{\Delta t}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p}\left(p_{hyd}^{n+1} - p_{hyd}^n\right)\right)$$

and

$$
\begin{aligned}
\left(\nabla^T \mathbf{v}\right)_{dyn}^{n+1} &\equiv -\tfrac{1}{\Delta t}\left(\log\left(\rho\left(t^{n+1}, p_{hyd}^n + p_{dyn}^{n+1}, A_v^n\right)\right) - \log\left(\rho_{dyn}^n\right)\right) \\
&= -\tfrac{1}{\Delta t}\left(\tfrac{1}{\rho}\tfrac{\partial \rho}{\partial p}\left(p_{dyn}^{n+1} - p_{dyn}^n\right) + \log\left(\rho\left(t^{n+1}, p_{hyd}^n + p_{dyn}^n, A_v^n\right)\right) - \log\left(\rho_{dyn}^n\right)\right) \\
&= -\tfrac{1}{\Delta t}\tfrac{1}{\rho}\tfrac{\partial \rho}{\partial p}\left(p_{dyn}^{n+1} - p_{dyn}^n\right) - \tfrac{1}{\Delta t}\left(\log\left(\rho\left(t^{n+1}, p_{hyd}^n + p_{dyn}^n, A_v^n\right)\right) - \log\left(\rho_{dyn}^n\right)\right) \\
&= -\tfrac{1}{\Delta t}\tfrac{1}{\rho}\tfrac{\partial \rho}{\partial p}\left(p_{dyn}^{n+1} - p_{dyn}^n\right) + \overline{\left(\nabla^T \mathbf{v}\right)}_{dyn}^{n+1}
\end{aligned}
$$

where we have

$$
\overline{\left(\nabla^T \mathbf{v}\right)}_{dyn}^{n+1} \equiv -\frac{1}{\Delta t}\left(\log\left(\rho\left(t^{n+1}, p_{hyd}^n + p_{dyn}^n, A_v^n\right)\right) - \log\left(\rho_{dyn}^n\right)\right)
$$

Remark: the term $\overline{\left(\nabla^T \mathbf{v}\right)}_{dyn}^{n+1}$ represents the compression/expansion of the material that is independent of the pressure, i.e. compression due to time, reaction kinetics, temperature change etc.

DerivePoissonEquationForPressure

how to compute the pressure from the equation of momentum

The Poisson equation for the pressure can be derived by application of the divergence-operator to the equation of momentum:

$$
\nabla^T \dot{\mathbf{v}} + \nabla^T\left(\frac{1}{\rho}\nabla\left(p\right)\right) = \nabla^T\left(\frac{1}{\rho}\nabla \mathbf{S}_s\right) + \nabla^T\left(\frac{1}{\rho}\nabla \mathbf{S}_v\left(\mathbf{v}\right)\right) + \nabla^T \mathbf{g} - \nabla^T\left(\beta \cdot \left(\mathbf{v} - \mathbf{v}_\beta\right)\right)
$$

That gives

$$
\frac{d}{dt}\left(\nabla^T \mathbf{v}\right) + \Phi\left(\mathbf{v}\right) + \nabla^T\left(\frac{1}{\rho}\nabla\left(p\right)\right) = \nabla^T\left(\frac{1}{\rho}\nabla \mathbf{S}_s\right) + \nabla^T\left(\frac{1}{\rho}\nabla \mathbf{S}_v\left(\mathbf{v}\right)\right) + \nabla^T \mathbf{g} - \nabla^T\left(\beta \cdot \left(\mathbf{v} - \mathbf{v}_\beta\right)\right)
$$

using the definitions

$\mathbf{v} = \left(u, v, w\right)^T$

$\Phi\left(\mathbf{v}\right) \equiv \left(\nabla u\right)^T \cdot \frac{\partial \mathbf{v}}{\partial x} + \left(\nabla v\right)^T \cdot \frac{\partial \mathbf{v}}{\partial y} + \left(\nabla w\right)^T \cdot \frac{\partial \mathbf{v}}{\partial z}$

More simplifications can achieved by defining

$\Psi\left(\mathbf{v}\right) \equiv \nabla^T\left(\frac{1}{\rho}\nabla \mathbf{S}_v\left(\mathbf{v}\right)\right)$

$\Pi\left(\mathbf{v}\right) \equiv \nabla^T\left(\frac{1}{\rho}\nabla \mathbf{S}_s\right)$

$\Theta\left(\mathbf{v}\right) \equiv \nabla^T\left(\beta \cdot \left(\mathbf{v} - \mathbf{v}_\beta\right)\right)$

That means

$$
\frac{d}{dt}\left(\nabla^T \mathbf{v}\right) + \nabla^T\left(\frac{1}{\rho}\nabla\left(p\right)\right) = \Pi + \nabla^T \mathbf{g} + \Psi\left(\mathbf{v}\right) - \Theta\left(\mathbf{v}\right) - \Phi\left(\mathbf{v}\right)
$$

In a numerical sense, this is

$$
\frac{\left(\nabla^T \mathbf{v}\right)^{n+1} - \left(\nabla^T \mathbf{v}\right)^n}{\Delta t} + \nabla^T\left(\frac{1}{\rho}\nabla\left(p^{n+1}\right)\right) = \Pi^{n+1} + \nabla^T \mathbf{g} + \Psi\left(\mathbf{v}^{n+1}\right) - \Theta\left(\mathbf{v}^{n+1}\right) - \Phi\left(\mathbf{v}^{n+1}\right)
$$

Splitting this equation into a hydrostatic and a dynamic part yields

$$
\frac{\left(\nabla^T \mathbf{v}\right)_{hyd}^{n+1} + \left(\nabla^T \mathbf{v}\right)_{dyn}^{n+1} - \left(\nabla^T \mathbf{v}\right)^n}{\Delta t} + \nabla^T\left(\frac{1}{\rho}\nabla\left(p_{hyd}^{n+1} + p_{dyn}^{n+1}\right)\right) = \Pi^{n+1} + \nabla^T \mathbf{g} + \Psi\left(\mathbf{v}^{n+1}\right) - \Theta\left(\mathbf{v}^{n+1}\right) - \Phi\left(\mathbf{v}^{n+1}\right)
$$

The different parts of pressure are more precisely described HydrostaticPressure and DynamicPressure .

The splitting of $\Theta\left(\mathbf{v}^{n+1}\right)$ into hydrostatic and dynamic parts is explained in ComputationOfTHETA .

List of members:

DynamicPressure	dynamic pressure derived from momentum equation
HydrostaticPressure	hydrostatic pressure derived from momentum equation

DynamicPressure

dynamic pressure derived from momentum equation

This pressure only occurs (different from zero) if the fluid is in motion. Therefore it represents the dynamic forces or compression forces.
Its basic equation stems from the considerations in DerivePoissonEquationForPressure and is given by

$$\frac{\left(\nabla^T \mathbf{v}\right)_{dyn}^{n+1} - \left(\nabla^T \mathbf{v}\right)^n}{\Delta t} + \nabla^T \left(\frac{1}{\rho} \nabla \left(p_{dyn}^{n+1}\right)\right) = \Psi\left(\mathbf{v}^{n+1}\right) - \Theta_{dyn}\left(\mathbf{v}^{n+1}\right) - \Phi\left(\mathbf{v}^{n+1}\right)$$

We take into account equation (1.10), hence we obtain

$$\begin{aligned}\frac{-\frac{1}{\Delta t}\frac{1}{\rho}\frac{\partial \rho}{\partial p}\left(p_{dyn}^{n+1} - p_{dyn}^{n}\right) + \overline{\left(\nabla^T \mathbf{v}\right)}_{dyn}^{n+1} - \left(\nabla^T \mathbf{v}\right)^n}{\Delta t} + \nabla^T \left(\frac{1}{\rho} \nabla \left(p_{dyn}^{n+1}\right)\right) =& \Psi\left(\mathbf{v}^{n+1}\right) \\ & - \Theta_{dyn}\left(\mathbf{v}^{n+1}\right) - \Phi\left(\mathbf{v}^{n+1}\right)\end{aligned}$$

and after sorting terms, the final representation of the dynamic pressure is

$$\begin{aligned}-\frac{1}{\Delta t^2}\frac{1}{\rho}\frac{\partial \rho}{\partial p}\left(p_{dyn}^{n+1} - p_{dyn}^{n}\right) + \nabla^T \left(\frac{1}{\rho} \nabla \left(p_{dyn}^{n+1}\right)\right) =& \frac{1}{\Delta t}\left(\nabla^T \mathbf{v}\right)^n - \frac{1}{\Delta t}\overline{\left(\nabla^T \mathbf{v}\right)}_{dyn}^{n+1} \\ & + \Psi\left(\mathbf{v}^{n+1}\right) - \Theta_{dyn}\left(\mathbf{v}^{n+1}\right) - \Phi\left(\mathbf{v}^{n+1}\right)\end{aligned}$$

The numerical way of solving this (extremely non-trivial) equation is found in DynamicPressureAlgorithm .

HydrostaticPressure

hydrostatic pressure derived from momentum equation

The basic equation for the hydrostatic pressure is:

$$\frac{\left(\nabla^T \mathbf{v}\right)_{hyd}^{n+1}}{\Delta t} + \nabla^T \left(\frac{1}{\rho} \nabla \left(p_{hyd}^{n+1}\right)\right) = \Pi^{n+1} + \nabla^T \mathbf{g} + \Theta_{hyd}^{n+1}$$

This pressure might be different from zero even if there is no motion of the fluid.
It might be due to gravity (depth pressure), internal forces (elasticity), etc.

We represent the compression part $\nabla^T \mathbf{v}$ by the expressions found in DeriveDivergenceOfVelocity :

$$\begin{aligned}\frac{\left(\nabla^T \mathbf{v}\right)_{hyd}^{n+1}}{\Delta t} + \nabla^T \left(\frac{1}{\rho} \nabla \left(p_{hyd}^{n+1}\right)\right) &= \Pi^{n+1} + \nabla^T \mathbf{g} + \Theta_{hyd}^{n+1} \\ -\frac{1}{\Delta t}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p}\left(p_{hyd}^{n+1} - p_{hyd}^{n}\right)\right) + \nabla^T \left(\frac{1}{\rho} \nabla \left(p_{hyd}^{n+1}\right)\right) &= \Pi^{n+1} + \nabla^T \mathbf{g} + \Theta_{hyd}^{n+1} \\ -\frac{1}{\Delta t}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p} p_{hyd}^{n+1}\right) + \nabla^T \left(\frac{1}{\rho} \nabla p_{hyd}^{n+1}\right) &= -\frac{1}{\Delta t}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p} p_{hyd}^{n}\right) + \Pi^{n+1} + \nabla^T \mathbf{g} + \Theta_{hyd}^{n+1}\end{aligned}$$

the final equation is

$$-\frac{1}{\Delta t}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p} p_{hyd}^{n+1}\right) + \nabla^T \left(\frac{1}{\rho} \nabla p_{hyd}^{n+1}\right) = -\frac{1}{\Delta t}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p} p_{hyd}^{n}\right) + \Pi^{n+1} + \nabla^T \mathbf{g} + \Theta_{hyd}^{n+1}$$

and its numerical discretization is found in HydrostaticPressureAlgorithm .

Scheme

List of members:

v--	segregated, directly incompressible solver
vp-	directly incompressible, implicit solver with penalty formulation

v--

segregated, directly incompressible solver

We describe the numerical scheme for incompressible / weakly compressible.
A document containing the scheme is found in DOCUMATH_GeneralNumericalScheme.pdf .

The timestep starts with an (explicit!) movement of the MESHFREE points.

$$\boldsymbol{x}_i^{n+1} = \boldsymbol{x}_i^n + \Delta t \cdot \boldsymbol{v}_i^n$$

The new positions of time level $n+1$ are found in %ind_x(1)% , %ind_x(2)% , %ind_x(3)% .
The old positions of time level n are kept in %ind_x0(1)% , %ind_x0(2)% , %ind_x0(3)% .

Compute all necessary material data. Especially see %ind_r% , %ind_ETA% , %ind_LAM% , %ind_MUE% , %ind_betaDarcy% , %ind_v0Darcy(1)% ... %ind_v0Darcy(3)% , %ind_SIG% , ...

$$\begin{aligned}
\rho &= \rho\left(t^{n+1}, p_{hyd}^n + p_{dyn}^n, T^n, A_v^n\right) \\
\eta &= \eta\left(t^{n+1}, p_{hyd}^n + p_{dyn}^n, T^n, A_v^n\right) \\
\lambda &= \lambda\left(t^{n+1}, p_{hyd}^n + p_{dyn}^n, T^n, A_v^n\right) \\
\mu &= \mu\left(t^{n+1}, p_{hyd}^n + p_{dyn}^n, T^n, A_v^n\right) \\
k_D &= k_D\left(t^{n+1}, p_{hyd}^n + p_{dyn}^n, T^n, A_v^n\right) \\
&\ldots
\end{aligned}$$

Also, compute derived data, for example:
the compressibility, see %ind_R_P% , also %ind_DiagPcorr% .

$$\frac{\partial \rho}{\partial p}$$

Compute the effective viscosity, see %ind_ETA_sm% and VelocityAlgorithm .

$$\hat{\eta} = \eta + C \cdot \Delta t \cdot \mu + c_\mu \cdot \frac{k^2}{\varepsilon}$$

Compute the effective body forces, see %ind_g(1)% ... %ind_g(3)%

$$\hat{\mathbf{g}}^n = \mathbf{g}^{n+1} + \frac{1}{\rho}\nabla \mathbf{S}_s^n$$

Solve the hydrostatic pressure. See HydrostaticPressureAlgorithm . See also LIQUID.%ind_p% .

$$-\frac{1}{\Delta t^2}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p}\tilde{p}_{hyd}^{n+1}\right) + \nabla^T\left(\frac{1}{\rho}\nabla \tilde{p}_{hyd}^{n+1}\right) = -\frac{1}{\Delta t^2}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p} p_{hyd}^n\right) + \nabla^T \hat{\mathbf{g}}^n$$

Solve the temperature. See TemperatureAlgorithm . See %ind_T% .

$$\rho \cdot c_v \frac{T^{n+1} - T^n}{\Delta t} = q + \nabla^T \left(\lambda \cdot \nabla T^{n+1} \right)$$

Set up the preliminary dynamic pressure for the momentum equation.

$$\tilde{p}_{dyn}^n = \mathcal{C} \cdot p_{dyn}^n$$

The preliminary value is stored in %ind_p_dyn% . The original value of the dynamic pressure at time level n is stored in %ind_p_dyn_0% .
Remember, that the parameter $\mathcal{C}$ is set by the input-file-parameter damping_p_corr .

Compute the nominal divergence of velocity, needed for the desired divergence of velocity in CorrectionPressureAlgorithm , see especially DesiredAndNominalDivergenceOfVelocity . Temporarily saved in %ind_div_bar% .

$$\left(\overline{\nabla^T \mathbf{v}} \right)_c^{n+1} \equiv -\frac{1}{\Delta t} \left(\log \left(\rho \left(t^{n+1}, p_{hyd}^{n+1} + \tilde{p}_{dyn}^n, T^{n+1}, A_v^n \right) \right) - \log \left(\rho_{p_{dyn}}^n \right) \right)$$

Solve the velocity. See VelocityAlgorithm . See %ind_v(1)% ... %ind_v(3)% as well as %ind_v_tild(1)% ... %ind_v_tild(3)% .

$$\frac{\tilde{\mathbf{v}}^{n+1} - \mathbf{v}^n}{\Delta t} + \frac{1}{\rho} \nabla \tilde{p}_{hyd}^{n+1} + \frac{1}{\rho} \nabla \tilde{p}_{dyn}^n = \frac{1}{\rho} \nabla \mathbf{S}_v \left(\tilde{\mathbf{v}}^{n+1} \right) + \hat{\mathbf{g}}^n - \beta \cdot \tilde{\mathbf{v}}^{n+1}$$

Compute correction pressure. See %ind_c% .

$$-\frac{1}{\rho} \frac{\partial \rho}{\partial p} \frac{1}{\Delta t} c + \nabla^T \left(\frac{\Delta t_{virt}}{\left(1 + \Delta t_\beta \cdot \beta \right)} \frac{1}{\rho} \nabla c \right) - \nabla^T \tilde{\mathbf{v}}^{n+1} = - \left(\overline{\nabla^T \mathbf{v}} \right)_c^{n+1}$$

Correct the velocity with the help of the correction pressure. Result in %ind_v(1)% ... %ind_v(3)%

$$\mathbf{v}^{n+1} = \tilde{\mathbf{v}}^{n+1} - \frac{\Delta t_{virt}}{\left(1 + \Delta t_\beta \cdot \beta \right)} \frac{1}{\rho} \nabla c$$

Update the dynamic pressure. See %ind_p_dyn%

$$\tilde{p}_{dyn}^{n+1} = \tilde{p}_{dyn}^n + c$$

Compute the new density as a backup for the next time step. See %ind_r_c% .

$$\rho_c^{n+1} = \rho \left(t^{n+1}, p_{hyd}^{n+1} + \tilde{p}_{dyn}^{n+1}, T^{n+1}, A_v^n \right)$$

Compute the stress tensor at time level $n+1$ by the stress tensor algorithm, i.e.

$$\boldsymbol{S}^{n+1} = f \left(\Delta t, \boldsymbol{S}^n, \boldsymbol{v}^n \right)$$

see the StressTensorAlgorithm .

Update turbulence values for k-epsilon. See %ind_k% and %ind_eps% .

$$k^{n+1} = k^n + ...$$

$$\varepsilon^{n+1} = \varepsilon^n + ...$$

Recompute the resulting body forces. See %ind_g(1)% ... %ind_g(3)% .

$$\hat{\mathbf{g}}^{n+1} = \frac{1}{\rho} \nabla \mathbf{S}_s^{n+1} + \mathbf{g}$$

Recompute, if needed, the hydrostatic pressure. See LIQUID.%ind_p% .

$$-\frac{1}{\Delta t^2} \left(\frac{1}{\rho} \frac{\partial \rho}{\partial p} p_{hyd}^{n+1} \right) + \nabla^T \left(\frac{1}{\rho} \nabla p_{hyd}^{n+1} \right) = -\frac{1}{\Delta t^2} \left(\frac{1}{\rho} \frac{\partial \rho}{\partial p} p_{hyd}^n \right) + \nabla^T \left(\hat{\mathbf{g}}^{n+1} \right)$$

Nominal divergence of velocity, motivated by dynamic pressure. Temporarily resulting in %ind_div_bar% .

$$\left(\overline{\nabla^T \mathbf{v}} \right)_{dyn}^{n+1} \equiv -\frac{1}{\Delta t} \left(\log \left(\rho \left(..., p_{hyd}^n + p_{dyn}^n, ... \right) \right) - \log \left(\rho_{p_{dyn}}^n \right) \right)$$

out of the velocity field, compute the consistent dynamic pressure. See %ind_p_dyn% .

$$-\frac{1}{\Delta t^2}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p}p_{dyn}^{n+1}\right)+\nabla^T\left(\frac{1}{\rho}\nabla\left(p_{dyn}^{n+1}\right)\right)=-\frac{1}{\Delta t}\left(\left(\overline{\nabla^T\mathbf{v}}\right)_{dyn}^{n+1}-\left(\nabla^T\mathbf{v}\right)^n\right)$$
$$-\frac{1}{\Delta t^2}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p}p_{dyn}^{n}\right)$$
$$+\Psi\left(\mathbf{v}^{n+1}\right)-\Theta\left(\mathbf{v}^{n+1}\right)-\Phi\left(\mathbf{v}^{n+1}\right)$$

backup the density after computing the dynamic pressure

$$\rho_{p_{dyn}}^{n+1}=\rho\left(t^{n+1},p_{hyd}^{n+1}+p_{dyn}^{n+1},T^{n+1},A_v^n\right)$$

compute the target divergence of velocity as a backup for the next time cycle. Resulting both in %ind_div_bar_0% and %ind_div_bar% .

$$\left(\nabla^T\mathbf{v}\right)_{target}^{n+1}=-\frac{1}{\Delta t}\left(\log\left(\rho_{p_{dyn}}^{n+1}\right)-\log\left(\rho_{p_{dyn}}^{n}\right)\right)$$

Compute the rediduals for the velocity. See %ind_v_residual(1)% ... %ind_v_residual(3)% .

Compute the rediduals for the density. See %ind_r_residual%
integrate all additional variables defined by the CODI commands. See also %ind_addvar% ...

$$A_v^{n+1}=F\left(A_v^n,t^{n+1},\mathrm{v}^{n+1},T^{n+1},p_{hyd}^{n+1},p_{dyn}^{n+1},k^{n+1},\varepsilon^{n+1}\right)$$

MESHFREE · Solvers · Numerics · LIQUID · Scheme · vp-

vp-

directly incompressible, implicit solver with penalty formulation

We describe the numerical scheme for incompressible / weakly compressible.
A document containing the scheme is found in DOCUMATH_GeneralNumericalScheme.pdf .

The timestep starts with an (explicit!) movement of the MESHFREE points.

$$\boldsymbol{x}_i^{n+1}=\boldsymbol{x}_i^n+\Delta t\cdot\boldsymbol{v}_i^n$$

The new positions of time level $n+1$ are found in %ind_x(1)% , %ind_x(2)% , %ind_x(3)% .
The old positions of time level n are kept in %ind_x0(1)% , %ind_x0(2)% , %ind_x0(3)% .

Compute all necessary material data. Especially see %ind_r% , %ind_ETA% , %ind_LAM% , %ind_MUE% , %ind_betaDarcy% , %ind_v0Darcy(1)% ... %ind_v0Darcy(3)% , %ind_SIG% , ...

$$\begin{aligned}\rho &= \rho\left(t^{n+1},p_{hyd}^n+p_{dyn}^n,T^n,A_v^n\right)\\ \eta &= \eta\left(t^{n+1},p_{hyd}^n+p_{dyn}^n,T^n,A_v^n\right)\\ \lambda &= \lambda\left(t^{n+1},p_{hyd}^n+p_{dyn}^n,T^n,A_v^n\right)\\ \mu &= \mu\left(t^{n+1},p_{hyd}^n+p_{dyn}^n,T^n,A_v^n\right)\\ k_D &= k_D\left(t^{n+1},p_{hyd}^n+p_{dyn}^n,T^n,A_v^n\right)\\ &\ldots\end{aligned}$$

Also, compute derived data, for example:
the compressibility, see %ind_R_P% , also %ind_DiagPcorr% .

$$\frac{\partial\rho}{\partial p}$$

Compute the effective viscosity, see %ind_ETA_sm% and VelocityAlgorithm .

$$\hat{\eta} = \eta + C \cdot \Delta t \cdot \mu + c_\mu \cdot \frac{k^2}{\varepsilon}$$

Compute the effective body forces, see %ind_g(1)% ... %ind_g(3)%

$$\hat{\mathbf{g}}^n = \mathbf{g}^{n+1} + \frac{1}{\rho}\nabla \mathbf{S}_s^n$$

Solve the hydrostatic pressure. See HydrostaticPressureAlgorithm . See also LIQUID.%ind_p% .

$$-\frac{1}{\Delta t^2}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p}\tilde{p}_{hyd}^{n+1}\right) + \nabla^T\left(\frac{1}{\rho}\nabla \tilde{p}_{hyd}^{n+1}\right) = -\frac{1}{\Delta t^2}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p} p_{hyd}^{n}\right) + \nabla^T \hat{\mathbf{g}}^n$$

Solve the temperature. See TemperatureAlgorithm . See %ind_T% .

$$\rho \cdot c_v \frac{T^{n+1} - T^n}{\Delta t} = q + \nabla^T\left(\lambda \cdot \nabla T^{n+1}\right)$$

Set up the preliminary dynamic pressure for the momentum equation.

$$\tilde{p}_{dyn}^n = \mathcal{C} \cdot p_{dyn}^n$$

The preliminary value is stored in %ind_p_dyn% . The original value of the dynamic pressure at time level n is stored in %ind_p_dyn_0% .
Remember, that the parameter $\mathcal{C}$ is set by the input-file-parameter damping_p_corr .

Compute the nominal divergence of velocity, needed for the desired divergence of velocity in CorrectionPressureAlgorithm , see especially DesiredAndNominalDivergenceOfVelocity . Temporarily saved in %ind_div_bar% .

$$\left(\overline{\nabla^T \mathbf{v}}\right)_c^{n+1} \equiv -\frac{1}{\Delta t}\left(\log\left(\rho\left(t^{n+1}, p_{hyd}^{n+1} + \tilde{p}_{dyn}^n, T^{n+1}, A_v^n\right)\right) - \log\left(\rho_{p_{dyn}}^n\right)\right)$$

Solve the velocity and the correction pressure in one big system of equations. See %ind_v(1)% ... %ind_v(3)% as well as %ind_v_tild(1)% ... %ind_v_tild(3)% . See %ind_c% (correction pressure).

$$\left\{\begin{array}{c} \frac{\tilde{\mathbf{v}}^{n+1} - \mathbf{v}^n}{\Delta t} + \frac{1}{\rho}\nabla \tilde{p}_{hyd}^{n+1} + \frac{1}{\rho}\nabla \tilde{p}_{dyn}^n + \frac{1}{\rho}\nabla c = \frac{1}{\rho}\nabla \mathbf{S}_v\left(\tilde{\mathbf{v}}^{n+1}\right) + \hat{\mathbf{g}}^n - \beta \cdot \left(\tilde{\mathbf{v}}^{n+1} - \mathbf{v}_\beta\right) \\ -\frac{1}{\rho}\frac{\partial \rho}{\partial p}\frac{1}{\Delta t} c + \nabla^T\left(\frac{\Delta t_{virt}}{\left(1 + \Delta t_\beta \cdot \beta\right)}\frac{1}{\rho}\nabla c\right) - \nabla^T \tilde{\mathbf{v}}^{n+1} = -\left(\overline{\nabla^T \mathbf{v}}\right)_c^{n+1} \end{array}\right\}$$

Update the dynamic pressure. See %ind_p_dyn%

$$\tilde{p}_{dyn}^{n+1} = \tilde{p}_{dyn}^n + c$$

Correct the velocity with the help of the correction pressure if VP0_VelocityCorrection is switched on. Result in %ind_v(1)% ... %ind_v(3)% .

$$\mathbf{v}^{n+1} = \tilde{\mathbf{v}}^{n+1} - \frac{\Delta t_{virt}}{\left(1 + \Delta t_\beta \cdot \beta\right)}\frac{1}{\rho}\nabla c$$

Compute the new density as a backup for the next time step. See %ind_r_c% .

$$\rho_c^{n+1} = \rho\left(t^{n+1}, p_{hyd}^{n+1} + \tilde{p}_{dyn}^{n+1}, T^{n+1}, A_v^n\right)$$

Compute the stress tensor at time level $n+1$ by the stress tensor algorithm, i.e.

$$\boldsymbol{S}^{n+1} = f\left(\Delta t, \boldsymbol{S}^n, \boldsymbol{v}^n\right)$$

see the StressTensorAlgorithm .

Update turbulence values for k-epsilon. See %ind_k% and %ind_eps% .

$$k^{n+1} = k^n + \ldots$$

$$\varepsilon^{n+1} = \varepsilon^n + \ldots$$

Recompute the resulting body forces. See %ind_g(1)% ... %ind_g(3)% .

$$\hat{\mathbf{g}}^{n+1} = \frac{1}{\rho}\nabla \mathbf{S}_s^{n+1} + \mathbf{g}$$

Recompute, if needed, the hydrostatic pressure. See LIQUID.%ind_p% .

$$-\frac{1}{\Delta t^2}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p} p_{hyd}^{n+1}\right) + \nabla^T\left(\frac{1}{\rho}\nabla p_{hyd}^{n+1}\right) = -\frac{1}{\Delta t^2}\left(\frac{1}{\rho}\frac{\partial \rho}{\partial p} p_{hyd}^{n}\right) + \nabla^T\left(\hat{\mathbf{g}}^{n+1}\right)$$

Nominal divergence of velocity, motivated by dynamic pressure. Temporarily resulting in %ind_div_bar% .

$$\left(\overline{\nabla^T \mathbf{v}}\right)_{dyn}^{n+1} \equiv -\frac{1}{\Delta t}\left(\log\left(\rho\left(..., p_{hyd}^n + p_{dyn}^n, ...\right)\right) - \log\left(\rho_{p_{dyn}}^n\right)\right)$$

out of the velocity field, compute the consistent dynamic pressure. See %ind_p_dyn% .

$$\begin{aligned}-\tfrac{1}{\Delta t^2}\left(\tfrac{1}{\rho}\tfrac{\partial \rho}{\partial p} p_{dyn}^{n+1}\right) + \nabla^T\left(\tfrac{1}{\rho}\nabla\left(p_{dyn}^{n+1}\right)\right) =& -\tfrac{1}{\Delta t}\left(\left(\overline{\nabla^T \mathbf{v}}\right)_{dyn}^{n+1} - \left(\nabla^T \mathbf{v}\right)^n\right) \\ & -\tfrac{1}{\Delta t^2}\left(\tfrac{1}{\rho}\tfrac{\partial \rho}{\partial p} p_{dyn}^{n}\right) \\ & +\Psi\left(\mathbf{v}^{n+1}\right) - \Theta\left(\mathbf{v}^{n+1}\right) - \Phi\left(\mathbf{v}^{n+1}\right)\end{aligned}$$

backup the density after computing the dynamic pressure

$$\rho_{p_{dyn}}^{n+1} = \rho\left(t^{n+1}, p_{hyd}^{n+1} + p_{dyn}^{n+1}, T^{n+1}, A_v^n\right)$$

compute the target divergence of velocity as a backup for the next time cycle. Resulting both in %ind_div_bar_0% and %ind_div_bar% .

$$\left(\nabla^T \mathbf{v}\right)_{target}^{n+1} = -\frac{1}{\Delta t}\left(\log\left(\rho_{p_{dyn}}^{n+1}\right) - \log\left(\rho_{p_{dyn}}^{n}\right)\right)$$

Compute the rediduals for the velocity. See %ind_v_residual(1)% ... %ind_v_residual(3)% .

Compute the rediduals for the density. See %ind_r_residual%
integrate all additional variables defined by the CODI commands. See also %ind_addvar% ...

$$A_v^{n+1} = F\left(A_v^n, t^{n+1}, \mathrm{v}^{n+1}, T^{n+1}, p_{hyd}^{n+1}, p_{dyn}^{n+1}, k^{n+1}, \varepsilon^{n+1}\right)$$

7.2.4. SHALLOWWATER

Solver for shallow water equations

To solve a shallow water problem, choose the kind of problem as

```
KOP(1) = SHALLOWWATER LAGRANGE
```

The shallow water phase can be coupled to a 3D liquid phase, say in chamber 2, in one or both directions by one of the following lines

```
KOP(1) = SHALLOWWATER LAGRANGE LPHASE:2
KOP(1) = SHALLOWWATER LAGRANGE COUPLING_3D->2D:2
KOP(1) = SHALLOWWATER LAGRANGE COUPLING_2D->3D:2
```

where the liquid phase might, for example, be defined as

```
KOP(2) = LIQUID IMPLICIT LAGRANGE vp- TURBULENCE:k-epsilon
```

Geometry aliases that can be in contact with both chambers, need to be defined for both separately, see also AliasForGeometryItems .
Example:

```
begin_alias{ }
### 3D Liquid ###
"inflow" = "BC$inflow$ ACTIVE$noinit_always$ IDENT%BND_inflow% MAT$Wasser$ TOUCH%TOUCH_always%
MOVE$NO_MOVE$ LAYER0 CHAMBER2 POSTPROCESS$PPinflow$ "
"bowl" = "BC$wall$ ACTIVE$noinit_always$ IDENT%BND_slip% MAT$Wasser$ TOUCH%TOUCH_liquid%
MOVE$NO_MOVE$ LAYER0 CHAMBER2 "
### Shallow water ###
"bowl" = "BC$swall$ ACTIVE$init_always$ IDENT%BND_slip% MAT$Wasser_SHW$ TOUCH%TOUCH_never%
MOVE$NO_MOVE$ LAYER0 CHAMBER1 POSTPROCESS$BOWL_SHW$ "
end_alias
```

Material parameters and initial conditions also need to be defined for both chambers. Note that for any boundary parts that may come into
contact with the shallow water chamber, a thin initial liquid film needs to be defined via a positive value of %ind_hwf% . Otherwise, the
points of the shallow water chamber will be deleted at the beginning of the simulation and cannot be recreated from the 3D liquid phase.

See also the list of indices at SHALLOWWATER .

MESHFREE · Solvers · Numerics · STANDBY

7.2.5. STANDBY

stanby with data, no numerical algorithm applied on the data otherwise

The STANDBY pointcloud
- does not undergo any point movement,
- will not add or remove points,
- will not apply for any numerical scheme,
- serves uniquely as data source (wind data in a rain droplet simulation, comparison data in convergence studies, etc., see ReadInPointCloud)

MESHFREE · Solvers · Numerics · TRANSPORT

7.2.6. TRANSPORT

List of members:

Algorithms	Algorithms

MESHFREE · Solvers · Numerics · TRANSPORT · Algorithms

Algorithms

List of members:

TimeIntegrationAlgorithm	TimeIntegrationAlgorithm

MESHFREE · Solvers · Numerics · TRANSPORT · Algorithms · TimeIntegrationAlgorithm

TimeIntegrationAlgorithm

This algorithm solves the semi-discrete ODE -system

$$\dot{\mathbf{U}}(t) = \mathbf{F}(t, \mathbf{U}(t))$$

with the implicit Euler method which is of order 1.
This subroutine is only for testing transport algorithms in F_of_t_Y_TRANSPORT.

This algorithm solves the semi-discrete ODE -system

$$\dot{\mathbf{U}}(t) = \mathbf{F}(t, \mathbf{U}(t))$$

with the implicit SDIRK2 method which is of order 2.
This subroutine is only for testing transport algorithms in F_of_t_Y_TRANSPORT.

This algorithm solves the semi-discrete ODE -system

$$\dot{\mathbf{U}}(t) = \mathbf{F}(t, \mathbf{U}(t))$$

with the implicit SDIRK3 method which is of order 3.
This subroutine is only for testing transport algorithms in F_of_t_Y_TRANSPORT.

This algorithm solves the semi-discrete ODE -system

$$\dot{\mathbf{U}}(t) = \mathbf{F}(t, \mathbf{U}(t))$$

with the explicit Euler method which is of order 1.

This algorithm solves the semi-discrete ODE -system

$$\dot{\mathbf{U}}(t) = \mathbf{F}(t, \mathbf{U}(t))$$

with the explicit Heun method (resp. expl. trapezoidal rule) which is of order 2.

This algorithm solves the semi-discrete ODE -system

$$\dot{\mathbf{U}}(t) = \mathbf{F}(t, \mathbf{U}(t))$$

with the classic Runge-Kutta method which is of order 4.

---

8. Download

Download executables, documentation and examples

MESHFREE Executables

- Stable version executables , see also Stable release notes,
- Beta version executables , see also Beta release notes.
- See also InstallationGuide and our Release cycle .

MESHFREE Documentation and Examples

- User documentation as single pdf: MESHFREEdocu.pdf ,

- Complete archive of user documentation (html pages, pdf docs, example setups): zip , tar.gz ,
- Online example setups (all): zip , tar.gz ,
- Tutorial example setups: zip , tar.gz ,
- LetterCases example setups: zip , tar.gz ,
- SpecialCases example setups: zip , tar.gz ,
- Single example setups linked as "COMPREHENSIVE EXAMPLE" throughout the documentation.

FITlm License Manager
- Executables and libraries ,
- User Manual (pdf) .
- See also README.md provided with MESHFREE executables.

---

9. PerformanceOptimization

useful insight into performance optimization

Here, we discuss current development regarding performance optimization of MESHFREE :

- GeometryOperations

List of members:

GeometryOperations	performance optimization concerning geometry operations

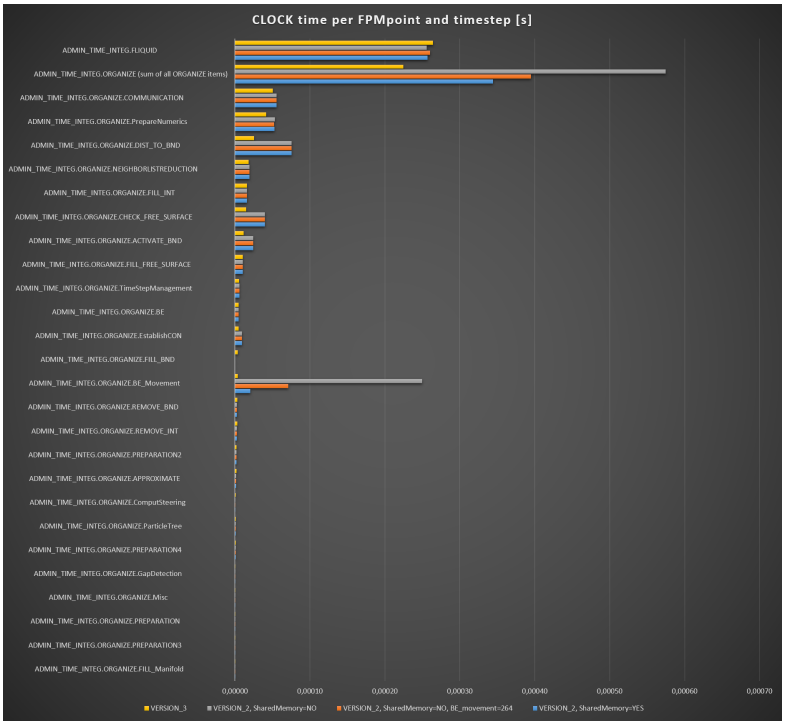
---

9.1. GeometryOperations

performance optimization concerning geometry operations

We list here the recent performance developments. The new algorithms run under version 3, the pervious ones under version 2, see below.

In order to judge on his own, the user is invited to check based on the COMP_TimeCheck functinality, considering the here mentioned stop watches.
Two examples have been carried out inorder to check the performance improvements:
- **Example 1** : simple box in channel case, see Classical with 80000 MESHFREE points on 4 MPI processes. Here, free surfaces, active boundary etc. are nicely distributed among the MPI-processes.
- **Example 2** : sophisticated water crossing of real car geometry, 5.4 Mio MESHFREE points, 192 MPI processes. Here, free surfaces and active boundaries are absolutely NOT load-balanced, such the (see below) the performance is less optimal.



The picture shows the speedup of the performance for the various ORGANIZE-tasks version 3 compared to different options of version 2 for the above mentioned **example 2** .
The time statistics include the MPI-bisection, even though that operation is performed ever 5 timesteps, only.

- **movement of geometry**

  ORGANIZE_USER_update_boundary_particles_Version = 3 # This option allows for shared memory as well as # # performancy boost for static geometries (BE with MOVE -1 are not considered for movement)

  previous standard was 2.

  Version 3 basically cuts down the computation time for geometry movement by one order of magnitude, as the CPU of a SharedMemory -block also
  share the effort for geometry movement. In version 3, we compute the rotation matrix $\mathbf{M}_{rot}^{n+1}$ and the translation vector $\mathbf{b}_{trans}^{n+1}$ such that the movement from the old to the new position of a geometry node is computed by

  $$\mathbf{x}_i^{n+1} = \mathbf{M}_{rot}^{n+1} \cdot \mathbf{x}_i^n + \mathbf{b}_{trans}^{n+1}$$

  For any rigid body movement, the translation and rotation items are unique, so the matrix and vector does not have to be recomputed for any
  geometry point. Thus, please also apply the option %MOVE_InvokeDataCaching% in order to avoid unnecessary recomputation of $\mathbf{M}_{rot}^{n+1}$ and $\mathbf{b}_{trans}^{n+1}$.

  -> **stopwatch** : ADMIN_TIME_INTEG.ORGANIZE.BE_Movement

  -> **SPEEDUP** (version 3 compared to version 2):
  - If MOVE -1 is used, or equally MOVE ($...$) = (%MOVE_none%,...), then these boundary elements are not considered for movement, thus they do not require simulation time. In version 2, even these boundary elements were processed in every time step.
  - If using COMP_SharedMemoryForBE = true, then the workload for boundary element movement is distributed evenly among the shared processes

  Thus, the speedup opportunities are tremendous with version 3.

- **neighbor list production**

```
GEOTREE2_EstablishCON_Version = 3 # After establishing the tree for neighbor search, MESHFREE installs the neighbor lists for each point
# # The way of neighbor list installation has impact on the performance.
```

The previous standard is 2.

REMARK: numbering different until version 18.11.0: 2(old) -> 1(new); 3(old) -> 2(new); 1(old) -> 3(new)

-> **stopwatches** : ADMIN_TIME_INTEG.ORGANIZE.COMMUNICATION.NEIGHLIST + ADMIN_TIME_INTEG.ORGANIZE.EstablishCON

-> **SPEEDUP** : example 1 example 2
. version 3: 1.00E-05 s/p 1.05E-05 s/p
. version 2: 2.30E-05 s/p 2.51E-05 s/p

Also, it might be a good idea to study the performance on the local machine architecture. On the native Fraunhofer-cluster, for example,
optimal values for the tree design were found to be

```
GEOTREE2_FinalBoxSize = 24
GEOTREE2_MaximumBoxSize = 32
GEOTREE2_IntListMargin = 8
```

The previous standard was 4 / 8 / 4

-> **SPEEDUP** : example 1 example 2
. version 3: 0.45E-05 s/p 0.48E-05 s/p
. version 2: 1.10E-05 s/p 0.97E-05 s/p

- **neighbor list reduction**

```
NEIGHBOR_FilterMethod = 3 # after establishing the neighborlist for each point, reduce those neighbors from the list
# # the rays of which pass through the boundary
```

The previous standard was 1 or 2.

-> **stopwatch** : ADMIN_TIME_INTEG.ORGANIZE.NEIGHBORLISTREDUCTION.CC2

-> **SPEEDUP** : example 1 example 2
. version 3: 0.30E-05 s/p 0.45E-05 s/p
. version 2: 0.45E-05 s/p 0.47E-05 s/p

- **neighbor list sorting**
  Neighborlist sorting is necessary in order to select the closest N neighbors, given by the parameter max_N_stencil . For defining the version of neighbor list sorting, have to set the second item in the parameter GEOTREE2_EstablishCON_Version :

```
GEOTREE2_EstablishCON_Version = (3,3) # After the establishing the tree for neighor search, MESHFREE installs the neighbor lists for each point.
# # The way of neighbor list installation has impact on the performance.
```

The previous standard was 2.

REMARK: not active until version 18.11.0

-> **stopwatch** : ADMIN_TIME_INTEG.ORGANIZE.NEIGHBORLISTREDUCTION.ONL

-> **SPEEDUP** : example 1 example 2
. version 3: 0.31E-05 s/p 0.17E-05 s/p
. version 2: 0.75E-05 s/p 0.41E-05 s/p

- **detection of free surface points**

```
ORGANIZE_CheckFreeSurface_Version = 3 #
```

The previous standard and current default is 2.

-> stopwatch: ADMIN_TIME_INTEG.ORGANIZE.CHECK_FREE_SURFACE

-> **SPEEDUP** : example 1 example 2
. version 3: 0.55E-05 s/p 0.14E-04 s/p
. version 2: 1.48E-05 s/p 0.41E-04 s/p

- **activation of boundary points**

```
ORGANIZE_ActivateBNDpoints_Version = 3 #
```

The previous standard and current default is 2.

-> stopwatch: ADMIN_TIME_INTEG.ORGANIZE.ACTIVATE_BND

-> **SPEEDUP** : example 1 example 2
. version 3: 0.62E-05 s/p 0.11E-04 s/p
. version 2: 1.37E-05 s/p 0.23E-04 s/p

- **computation of distance to boundary for all points**

```
ORGANIZE_DistanceToBoundary_Version = 3
```

The previous standard and current default is 2.

-> stopwatch: ADMIN_TIME_INTEG.ORGANIZE.DIST_TO_BND

-> **SPEEDUP** : example 1 example 2
. version 3: 0.15E-05 s/p 0.12E-04 s/p
. version 2: 0.75E-05 s/p 0.31E-04 s/p

10. Support

How to contact the Support Team

Support Team

Our support team is available via

- Email: support@meshfree.eu ,
- Phone: +49 (0) 631 316 00-1361.

Tickets can be in English or German. Please refer to the tips below.

Tips

To speed up the process of debugging and to avoid a lot of call backs, please give us as much information about your problem as possible up front. Besides an accurate description of your observed problem(s), including screenshots where suitable, a complete **minimal working example** showcasing the issue is most helpful. If you cannot provide a minimal working example, please upload the complete setup (USER_common_variables.dat, common_variables.dat, geometry and any other necessary files).

The following questions should be answered as accurately as possible:

- Which version of MESHFREE has been used? What is the name of the executable?
- Were other versions also tested? If so, which versions did work and which did not work?
- What is the error message? It would be ideal to provide the entire stdout and stderr output for at least the last time step.
- What is written in the warnings file?
- How many MPI processes have been used overall? On how many nodes?
- How many openMP threads have been used?
- For how long has the simulation been running before the error occured?
- Can you send us the complete setup via the FileDrop below?
  Or at the very least the USER_common_variables.dat file?

For files that are too large for an email, you can use our Support Ticket File Drop . We recommend to upload data as a single (encrypted) archive. Please ensure to **add the ticket number to the name of each file** , for example Files_Ticket12345.zip.

Tickets can be in English or German.

---

11. Releases

Information on the MESHFREE releases

Release Cycle (Plan)

- New beta version (format betaYYYY.MM.V) released by default every 2 months (01, 03, 05, 07, 09, 11).
- Additional beta versions have to be negotiated.
- New stable version released twice a year (format RYYYYa, RYYYYb; candidates start with betaYYYY.01.0 and betaYYYY.07.0).

You find the corresponding release notes on the pages Beta and Stable .

Download

Executables are available at:

https://svn.itwm.fraunhofer.de/svn/MESHFREEdocu/Executables/MESHFREE/

Mailing list

If you would like to be notified about new versions of MESHFREE , please subscribe to the respective mailing list(s): MESHFREE-Stable and/or MESHFREE-Beta

List of members:

__DeprecatedItems__	Deprecated items to be removed in the near future
Beta	Release notes for the MESHFREE beta executables
Stable	Release notes for the MESHFREE stable executables
TestManagement	Test Management Plan

Releases

Executables