HYBIRD
Manual

Alessandro Leonardi

July 26, 2022

Contents

Chapter 1

Introduction

HYBIRD is a numerical code that combines a Lattice-Boltzmann Method (LBM) solver for fluid dynamics with a Discrete Element Method (DEM) solver for the solution of the dynamics of a particle ensemble. Both DEM and LBM are parallelized using OPENMP loops, and therefore optimized for a small number of processors, sharing memory.

This document describes the user interface, a bit of the theoretical background and guides the user through a set of tutorials. The content is organized as follows. In chapter 2 the configuration and input files are described, while section 3 outlines the output files and gives a brief introduction to their use. A set of tutorials is provided in chapter 4, guiding the user MISSING. Finally, a theoretical outline for DEM is provided in chapter 5.

1.1 Fluid dynamics with LBM

The Lattice-Boltzmann Method, commonly abbreviated as LBM, is a recently developed tool for the simulation of fluid systems. Over the last two decades it has gained increased popularity, because of its superior performance on parallel architectures [1], which makes LBM one of the most promising tool for the future development of Computational Fluid Dynamic (CFD). LBM is generally successful in handling intricate boundary conditions, and is an optimal tool for the coupling of fluid and particles. In this section, a basic version of LBM for fluid dynamics is described, together with the adjustments and extensions necessary for the simulation of debris flows. Further information, and a complete overview of the method can be found in Ref. [2], while a practical implementation guide is provided in Ref. [3].

1.2 Granular dynamics with DEM

The DEM solver has been developed by Alessandro Leonardi. While LBM solves the Navier-Stokes equation on an Eulerian, fixed grid, DEM follows a Lagrangian approach. A group of spherical particles is treated as a mobile set of points, which can navigate through the fluid domain. The two solvers are coupled by the transmission of hydrodynamic interactions between fluid and particles, via an Immersed Boundary Method (IBM). The discretization in time is explicit, and a Gear 5th order predictor-corrector scheme is used. The contact mechanics is based on a soft-contact approach. Two spheres are allowed to overlap slightly, and a repulsive force is applied to both element. Two models are available for the resolution of the contact: a Linear model and a more accurate but less intuitive Hertzian model. On the tangential direction, frictional forces arise whenever two surfaces are in contact. The code is described in detail in chapter 5.1.

Chapter 2

Pre-processing

2.1 Compilation

The code can be compiled from scratch directly in the terminal by positioning into the main folder (where the file `makefile` is located) and typing:

```
make clean
make -j 1
```

The number "1" can be substituted by the number of processors to be used by the compilation. The recommended number is 4. A larger one might give conflicts, which anyway can be resolved by waiting for the compiler to stop and calling the `make` command again (compilation restarts from where it had stopped).

The compiler will then produce a subfolder called "dist", in the same folder where the `makefile` is located. Inside it, there will be some additional subfolders, which contain the main executable, which is called `hybird.exe`.

2.2 Running the code locally

```
$ ./hybird.exe -c incline_flow.cfg -d results -n time
```

Figure 2.1: A sample command line

On a local computer, the executable can be run by preparing a series of files and folders (see the tutorial section, Sec. 4), and by executing the following command on the terminal:

```
OMP_NUM_THREADS={n_p} ./hybird.exe -c path/to/config/file.cfg -d path/to/storage/
    folder -n simulation_name {other options}
```

The number that substitutes `n_p` sets the number of processor that run the simulations. The options are now described:

`-c path/to/config/file.cfg` This is a compulsory option. Specifies the location of the configuration file, which can be of any extension (.cfg is used throughout this manual).

`-d path/to/storage/folder` Specifies the location of the directory where the results should be created. The directory needs to exist before the code starts running. Note that every simulation run will then create a subfolder within this directory.

`-n simulation_name` Specifies the name of the simulation. Eventually, this is the name of the subfolder created by the simulation. If the name is specified as `-n time`, then the subfolder will be named as a combination of the date and time of when the simulation starts.

other options Any parameter set in the configuration file can be overwritten using the following patter: `-{config_parameter} {value}` as an additional option in the command line. Particularly useful when a large set of simulations is run, because it cancels the need to have multiple configuration files.

2.3 Running the code on the HPC@POLITO cluster

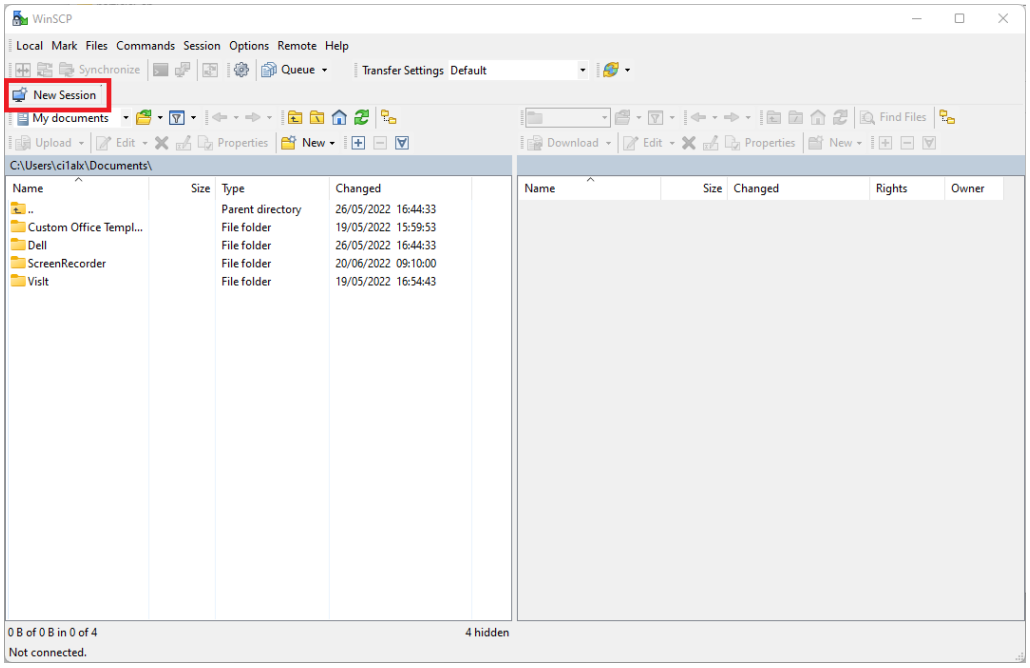

Figure 2.2: The interface of WinSCP

On the cluster, the procedure is slightly different. First, to make things easier for who is not used to work remotely with a terminal, it is recommended to download the software WinSCP. This software creates a graphical interface to manage files and folder on the cluster. When downloaded, open it and select "New Session". Fill in the form with your login data, selecting "legionlogin.polito.it" as the Host name, and fill in your cluster username and password. You can also save these credentials so that you do not have to type them again.

Once logged in, You will be able to see the files on your folder in the cluster, and to prepare the simulation environment there. The first thing is to copy the source code of `HYBIRD` in the cluster, and compile the executable there. **This is very important!** The cluster will be able to run an executable only if it has been compiled there. An executable compiled on a local PC and copied to the cluster will not run.

In the cluster, it is not a good idea to sun the code issuing a command. Rather, it is necessary to prepare an additional file, a "job file", which can be created with any text editor. The extension is `.sh`, e.g. `myjobfile1.sh`. The job file contains the information necessary to run the code, in addition to the command line itself. It contains several options, and is needed for the cluster to manage efficiently its use by multiple users. An example job file is provided in the manual folder.

In the following, the options are described. All options need to be preceded by the keyword `#SBATCH`. Note that a header is present too: `#!binbash`

`--job-name` This is the job name that will appear in the cluster queue and will be useful to identify the simulation. It should be meaningful but also short, a maximum of 6 characters is recommended. it is a good idea to use the same name also for the results folder. In the example in Fig. 2.4, this name is `d2`. Note how this name appears overall four times in the job file. This is recommended to have a consistent labelling of the simulation.

`--mail-type` "ALL" will make so that every update on the simulation status will be emailed to the user. More options are available in the cluster manual.

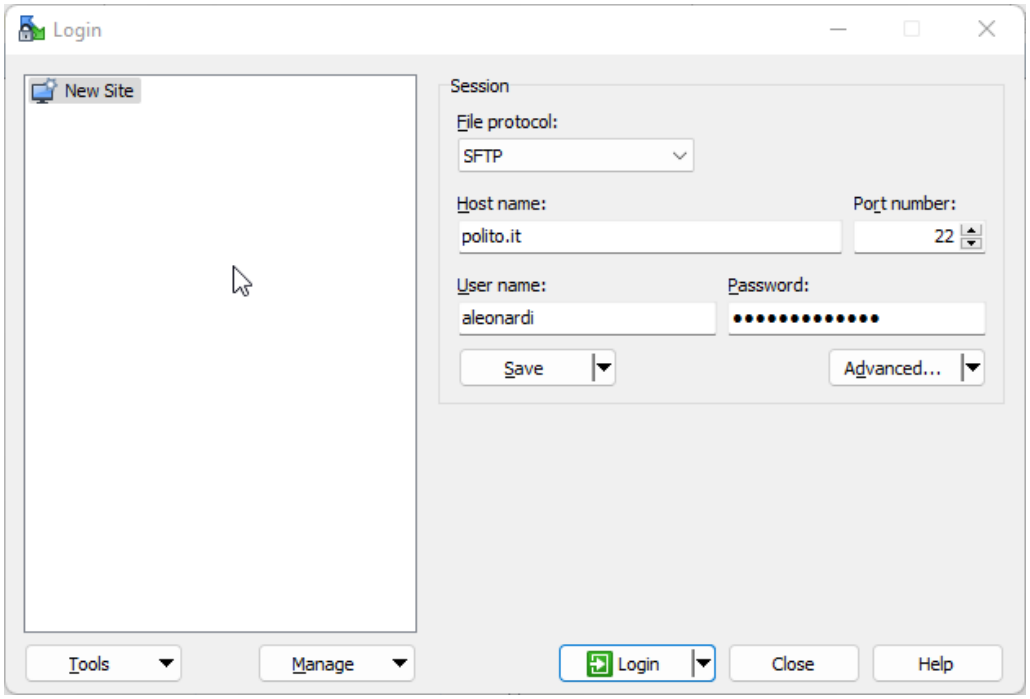

Figure 2.3: The login interface necessary to create a new connection

`--partition` "global" is the correct option.

`--time` This is the maximum real simulation runtime, in hours:minutes:seconds. The maximum allowable by the cluster is 10 days, so a safe option is "240:00:00"

`--nodes` The number of nodes to be employed. `HYBIRD` is optimized for a single node, so "1" is the correct choice.

—`--mail-user` This is the user email, where all updates will be notified.

—`--ntasks-per-node` This is the number of threads. It must be the same number as in the "OMP_NUM_THREAD" option declared below, so care must be placed that these two numbers coincide.

`--mem-per-cpu` The cluster requires to declare in advance the amount of memory required to run the simulation. This memory will be reserved by the queue manager. The amount needs to

```
#!/bin/bash
#SBATCH --job-name=d2
#SBATCH --mail-type=ALL
#SBATCH --partition=global
#SBATCH --time=240:00:00
#SBATCH --nodes=1
#SBATCH --mail-user=alessandro.leonardi.ing@gmail.com
#SBATCH --ntasks-per-node=4
#SBATCH --mem-per-cpu=64M
#SBATCH --output=./results/d2/output.txt
#SBATCH --error=./results/d2/error.txt

export OMP_NUM_THREADS=4

./hybird.exe -c ./erosion2D_particles.cfg -d ./results -n d2
```

Figure 2.4: A sample job file

be estimated with a good level of precisions. Simulation that use only a small part of the reserved memory will be killed, and simulation exceeding it risk slowing down unnecessarily. It is possible to estimate the memory required by running the simulation on a local machine, and check the memory consumption used by `hybird.exe` in the task manager. Note that in the job file the memory is specified *per CPU*. In the example in Fig. 2.4, 64 MB have been reserved per CPU, and 4 threads are used. The amount of memory effectively reserve is therefore $4 \cdot 68 = 252$ MB.

`--output` This is a file where the cluster will save the text output generated by `HYBIRD`. This is the output that, on a local simulation, would be displayed on the terminal.

`--error` This is a file where the cluster will save any error message.

The job file will also need to have a line that sets up the number of threads. This is export `OMP_NUM_THREADS=N` where `N` is the number of threads (4 in the example). Finally, the command line needs to be included in the end, with all relevant options (see previous section). it is not recommended here to use the option `-n time` to name the simulations with a timestamp. Always use a set name. Important: every option in the configuration file that includes a file path should be updated to make sense in the cluster folder. This includes the particle file, even if it is empty.

Once the job file is ready, the simulation folder should have the following elements:

1. The executable `hybird.exe`
2. The results folder
3. A subfolder inside the results folder, named identically to the simulation name declared in the job file (`d2` in the example).
4. The configuration file
5. The particles file(s)

Once this is ready, it is time to submit the job. Open a Cygwin terminal, and use the following command:

```
ssh cluster_username@legionlogin.polito.it
```

Substitute `cluster_username` with your personal username in the cluster, answer "yes" to the security question, and type your password when requested. If `ssh` is not recognized, this means that the `Cygwin` installation did not include it. This can be easily fixed by running the `Cygwin` installer, and choose to include `openssh` to the installation.

Now you are using the terminal, and you are connected to the cluster. Navigate to the simulation folder, and type:

```
sbatch jobfile_name.sh
```

where `jobfile_name` should be substituted with the name of your job file. The job will then be queued, and will run as soon as the requested resources are available. You will receive an email when this happens. To check the status of the simulation, run:

```
squeue -u cluster_username
```

Substitute `cluster_username` with your personal username in the cluster. you will see a list of all submitted jobs, with their status. `R` means that a job is running, `P` means paused, `S` suspended, etc... Note that a numerical ID is associated to every job. If you want to kill that job, run:

```
scancel jobID
```

and to cancel all jobs, run

```
scancel -u cluster_username
```

When the job is finished, you will receive an email. Should the simulation file, you can check what was wrong by looking inside the `error` file in the result folder. if you cannot find this file, probably the folder was not prepared properly before submitting the job file. A more comprehensive guide to this system is available at `https://slurm.schedmd.com/quickstart.html`.

2.4 The configuration file

The configuration files, which includes setting for both DEM and LBM is now described.

2.4.1 Switchers for solvers

`demSolver` Activates the DEM solver. Note that particles can be included also if this parameter is off, but they will not move (see objects below).
Type: `Boolean (0-1)`.
Default: `0`.

`lbmSolver` Activates the LBM solvers. If off, no fluid is included.
Type: `Boolean (0-1)`.
Default: `0`.

`freeSurfaceSolver` Activates the free-surface solver for LBM. If off, the free surface will not evolve. This is mostly an optimization parameter: if there is no need for the free surface to evolve, keeping it off will increase the performance.
Type: `Boolean (0-1)`.
Default: `0`.

`forceFieldSolver` Activates the acceleration field for LBM. If off, there is no external forcing. This is another optimization parameter: keeping it off will increase (slightly) the performance.
Type: `Boolean (0-1)`.
Default: `0`.

`staticFrictionSolver` Activates static friction in LBM. If off, friction is only a dissipative force that cannot sustain stable structures. This is a key optimization parameter for DEM: keeping it off when not necessary will increase greatly the performance.
Type: `Boolean (0-1)`.
Default: `0`.

2.4.2 Problem Name

`problemName` Unless the code has been modified for a specific problem, this parameter should be kept to`NONE`. It is used to activate parts of the codes that have been implemented for a specific geometry or problem (e.g., an algorithm for printing some additional files, or a special geometry).
Type: `String`.
Default: `NONE`.

2.4.3 Output

`screenExpTime` Simulation time interval between two on-screen print of the state of the system. This is not automatically set to happen at every time step an in LES. If `screenExpTime`=0, this output is suppressed.
Type: `Double`.
Default: 0.0.

`fluidExpTime` Simulation time interval between two creations of fluid Paraview files. If `fluidExpTime`=0, this output is suppressed.
Type: `Double`.
Default: 0.0.

`fluidLagrangianExpTime` Simulation time interval between two creations of a special type of fluid Paraview files. It is called Lagrangian because the fluid nodes are printed as if they were a collection of point particles. This can be handy if the volume of simulated fluid is much smaller htan the volume of the domain. If `fluidLagrangianExpTime`=0, this output is suppressed.
Type: `Double`.
Default: 0.0.

`partExpTime` Simulation time interval between two creations of particle Paraview files. If `partExpTime`=0, this output is suppressed.
Type: `Double`.
Default: 0.0.

`partRecycleExpTime` Simulation time interval between two creations of particle restart files. If `partRecycleExpTime`=0, this output is suppressed.
Type: `Double`.
Default: 0.0.

`fluidRecycleExpTime` Simulation time interval between two creations of fluid restart files. If `fluidRecycleExpTime`=0, this output is suppressed.
Type: `Double`.
Default: 0.0.

`fluid2DExpTime` Simulation time interval between two creations of bi-dimensional (integrated) files. This is a type of file handy for free-surface flows, where the coordinate z is aligned with gravity. It shows variables integrated over the flow thickness, in the vertical direction. If `fluid2DExpTime`=0, this output is suppressed.
Type: `Double`.
Default: 0.0.

`objectExpTime` Simulation time interval between two creations of object restart files. If `objectExpTime`=0, this output is suppressed.
Type: `Double`.
Default: 0.0.

`singleObjects` Write here the ID number of objects that should be tracked, in terms of impact forces.
Type: `Array of integers`.
Default: `" " (empty)`.

2.4.4 Time integration

`demInitialRepeat` This activates the possibility to have DEM run for a specific time, before LBM starts to iterate. Useful if the particles need to arrange on a specific patterns before the fluid should begin. It is however recommended to not use this parameter. It is better to have a specific, DEM-only simulation run, and then use the particle recycle files.
Type: `Double`.
Default: 0.0.

`lbmInitialRepeat` This activates the possibility to have LBM run for a specific time, before DEM starts to iterate. Useful if the fluid pressure filed needs to stabilize before the DEM simulation triggers.
Type: `Double`.
Default: 0.0.

`maximumTimeSteps` If non-zero, the system stops when the prescribed maximum number of LBM iterations has been achieved. If 0, there is no control on the maximum number of iterations.
Type: `Integer`.
Default: 0.

`maxTime` Maximum simulated time allowed for the simulation (regardless of whether it is an LBM or DEM simulation). If 0.0, there is no control on the maximum simulated time.
Type: `Double`.
Default: 0.0.

`fluidTimeStep` The time step for LBM. This is a key parameter, both for stability, and scaling. If it is set to zero, the code automatically finds an optimal time step such that the corresponding relaxation time (based on the initial viscosity) is unitary. In a DEM-only simulation, this should be about two orders of magnitude larger than the expected DEM time step.
Type: `Double`.
Default: 0.0.

`multiStep` This number defines how many DEM steps should be performed between two LBM steps. Note that this is the only way to enforce a specific time step for DEM. If 0, the procedure based on the critical ratio (see below) is used instead
Type: `Integer`.
Default: 0

`criticalRatio` This sets a constant ratio between the estimated duration of particle collisions and the DEM time step. Essentially, it is a stability parameter used to determine an optimum DEM time step. The smallest `criticalRatio` is, the smallest the DEM time step will be. Recommended (safe) numbers are of the order of 0.002-0.005.
Type: `Double`.
Default: 0.1.

2.4.5 Domain and forcing

`domainSizeX,domainSizeY,domainSizeZ` Domain size in the three directions.
Type: `Double`.
Default: 0.0. Needs to be set by user.

`forceX,forceY,forceZ` External forcing in the three directions.
Type: `Double`.
Default: 0.0

`boundary0` Controls the boundary in the negative x direction.

`boundary1` Controls the boundary in the positive x direction.

`boundary2` Controls the boundary in the negative y direction.

`boundary3` Controls the boundary in the positive y direction.

`boundary4` Controls the boundary in the negative z direction.

`boundary5` Controls the boundary in the positive z direction.

`ALL BOUNDARIES` Possibilities active in this version are: 4: periodicity, 5: slip static wall, 6: slip moving wall, 7: no-slip static wall, 8: no-slip moving wall, 12: outlet (only LBM). It is recommended to stick to option 4 and 7.
Type: `Integer`.
Default: 7.

2.4.6 LBM parameters

`latticeSpacing` Lattice spacing for LBM. The domain size in all directions must be a multiple of the lattice spacing. This is a key parameter, both for stability, and scaling. In an LBM-DEM simulation, this should be sufficiently smaller than the particle diameter.
Type: `Double`.
Default: 0.0. Needs to be set by user.

`fluidDensity` Fluid density for LBM
Type: `Double`.
Default: 0.0. Needs to be set by user.

`minTau` Minimum value of the relaxation time. Important for non-Newtonian flows, as it controls the minimum viscosity that the fluid cells can have.
Type: `Double`.
Default: 0.0. Needs to be set by user.

`minTau` Maximum value of the relaxation time. Important for non-Newtonian flows, as it controls the maximum viscosity that the fluid cells can have.
Type: `Double`.
Default: 0.0. Needs to be set by user.

`TRTsolver` Activates the Two Relaxation Time (TRT) solver, which is currently under development. `false`
Type: `Boolean`.
Default: `false`.

`imposeFluidVolume` Triggers an algorithm that imposes exact volume conservation (in a non-orthodox way)
Type: `Boolean`.
Default: `false`.

`imposedFluidVolume` If `imposeFluidVolume` is activated, this sets the exact volume that needs to be simulated.
Type: `Double`.
Default: 0.0.

`restartFluid` Uses a recycle file to start the simulation. The restart files must have been created with a previous simulation using the `fluidRecycleExpTime` parameter.
Type: `Boolean`.
Default: `false`.

`fluidRestartFile` If `restartFluid` is active, this field must specify the address of the recycle file. Can be either an absolute or a relative address.
Type: `String`.
Default: `./`.

`applyTopography` Triggers the use of a topography file. This feature is currently under development.
Type: `Boolean`.
Default: `false`.

`topographyFile` If `applyTopography` is active, this field must specify the address of the topography file. Can be either an absolute or a relative address. This feature is currently under development.
Type: `String`.
Default: `./`.

`fluidFromTopography` If `applyTopography` is active, initiates the fluid initial volume and distribution starting from the topography file. This feature is currently under development.
Type: `Boolean`.
Default: `false`.

`translateTopographyX,translateTopographyY,translateTopographyZ` If `applyTopography` is active, translates the topography file before applying it in the simulation. This is necessary because `HYBIRD` always uses a relative reference frame, with the coordinate (0.0,0.0,0.0) the bottom-left corner of the domain, while topography files tend to be based on a geographical reference. This feature is currently under development.
Type: `Double`.
Default: 0.0.

`fluidMinX,fluidMinY,fluidMinZ` If no other specification is made (e.g. by defining a `problemName`), the fluid initial volume is a cuboid, and these parameters set the lowest boundaries of the cuboid in the three directions.

Type: `Double`.
Default: 0.0.

`fluidMaxX,fluidMaxY,fluidMaxZ` If no other specification is made (e.g. by defining a `problemName`), the fluid initial volume is a cuboid, and these parameters set the upper boundaries of the cuboid in the three directions.
Type: `Double`.
Default: the domain size in the respective directions.

`fluidInitVelocityX,fluidInitVelocityY,fluidInitVelocityZ` Sets the initial velocity of the fluid in all directions.
Type: `Double`.
Default: 0.0.

`slipCoefficient` If slip walls are used (see definition of boundaries), this sets the partial slip coefficient. A unitary value indicates full slip, while zero means no-slip.
Type: `Double`.
Default: 0.0.

`hydrodynamicRadius` This reduces the radius of the DEM particles for what concerns the coupling algorithms.
Type: `Double`.
Default: 1.0.

`rheologyModel` Befines which rehological model should be used. Possibilities are `NEWTONIAN`, `BINGHAM`, `FRICTIONAL`, `VOELLMY`, `BAGNOLD`, `MUI`. This also defines which rheological parameters will be used by the code
Type: `String`.
Default: `NEWTONIAN`.

`initVisc` The initial dynamic viscosity ν. For the `NEWTONIAN` mode, this is also the viscosity for the whole simulation.
Type: `Double`.
Default: 0.0. Needs to be set by user.

`plasticVisc` The plastic dynamic viscosity ν_{p}, used in the `BINGHAM` model.
Type: `Double`.
Default: 0.0.

`yieldStress` The yield stress σ_{y}, used in the `BINGHAM` model.
Type: `Double`.
Default: 0.0.

`frictionCoefFluid` The friction coefficient, used in the `FRICTIONAL` and `VOELLMY` models. This is also used in the `MUI` as the base friction coefficient (μ_0, for $I = 0$).
Type: `Double`.
Default: 0.0.

`deltaFriction` The difference between the maximum friction coefficient and the base friction coefficient $\mu_2 - \mu_0$, used in the `MUI` model.
Type: `Double`.
Default: 0.0.

`baseInertial` The base inertial number I_0, used in the `MUI` model.
Type: `Double`.
Default: 0.0.

`particleDiameter` The particle diameter (of those particles that constitute the granular continuum, not to be confused with the DEM), used in the `MUI` model.
Type: `Double`.
Default: 0.0.

`turbulenceSolver` This activates an LES turbulence model, with Smagorinsky constant defined below (`turbConst`). It should be used only with the `NEWTONIAN` model.
Type: `Boolean (0-1)`.
Default: `0`.

`turbConst` The Smagorinsky constant, which is used if `turbulenceSolver` is active.
Type: `Double`.
Default: 0.0.

2.4.7 DEM parameters and contact laws

`particleDensity` Mass density of the DEM particles.
Type: `Double`.
Default: 0.0. Needs to be set by user.

`contactModel` Model for the contact law. Possibilities are `LINEAR` and `HERTZIAN`.
Type: `String`.
Default: `NONE`. Needs to be set by user.

`youngMod` If `contactModel`=`HERTZIAN` sets the young Modulus of the particle material.
Type: `Double`.
Default: 1.0.

`poisson` If `contactModel`=`HERTZIAN` sets the Poisson ratio of the particle material.
Type: `Double`.
Default: 0.3.

`linearStiff` If `contactModel`=`LINEAR` sets the linear stiffness of the particle contacts.
Type: `Double`.
Default: 1.0.

`restitution` Sets the restitution coefficient of the particle contacts. Must be larger than 0 and smaller or equal to 1.
Type: `Double`.
Default: 1.0.

`viscTang` Damping constant of the tangential contact.
Type: `Double`.
Default: 1.0.

`frictionCoefPart` Friction coefficient of the particle-particle contacts.
Type: `Double`.
Default: 1.0.

`frictionCoefWall` Friction coefficient of the particle-wall contacts.
Type: `Double`.
Default: 1.0.

`frictionCoefObj` Friction coefficient of the particle-object contacts.
Type: `Double`.
Default: 1.0.

`rollingCoefPart` Rolling friction coefficient (same for all contacts).
Type: `Double`.
Default: 0.0.

`particleFile` This field specifies the address of the particle file (see the section on Input/Output for details). Can be either an absolute or a relative address. Even if particles are not used, an existing file must be specified (if no particles are needed, just set a file with a single "0" as content).
Type: `String`.
Default: `./`.

`particleTranslateX,particleTranslateY,particleTranslateZ` Used to translate the initial position of the particles.
Type: `Double`.
Default: 0.0.

`particleScale` Used to scale the initial position and radius of the particles.
Type: `Double`.
Default: 0.0.

`objectFile` This field specifies the address of the object file (see the section on Input/Output for details). Objects are immobile particles that are used to create stable obstacles or structures. The file can be either an absolute or a relative address. Even if objects are not used, an existing file must be specified (if no object is needed, just set a file with a single "0" as content).
Type: `String`.
Default: `./`.

`numVisc` Extra numerical viscosity to improve stability. It is recommended to keep it a couple of orders of magnitude below the viscosity of the fluid.
Type: `Double`.
Default: 0.0.

2.4.8 Rotational reference frame

`coriolisSolver` Activates the Coriolis apparent acceleration.
Type: `Boolean (0-1)`.
Default: `0`.

`centrifugalSolver` Activates the centrifugal acceleration.
Type: `Boolean (0-1)`.
Default: `0`.

`rotationX,rotationY,rotationZ` Coordinates of the rotational speed of the reference frame.
Type: `Double`.
Default: 0.0.

`rotationCenterX,rotationCenterY,rotationCenterZ` Coordinates of the center of rotation.
Type: `Double`.
Default: 0.0.

2.5 Restart and geometrical files

2.5.1 Particle/Object restart file

Every run with DEM activated requires the provision of two extra restart files: the particle restart file and the object restart file. Both have the same format, which is

```
1 NP
2 index_0 size_0 r_0 px_0 py_0 pz_0 ux_0 uy_0 uz_0 wx_0 wy_0 wz_0 q0_0 q1_0 q2_0 q3_0
      qp0_0 qp1_0 qp2_0 qp3_0
3 index_1 size_1 r_1 px_1 py_1 pz_1 ux_1 uy_1 uz_1 wx_1 wy_1 wz_1 q0_1 q1_1 q2_1 q3_1
      qp0_1 qp1_1 qp2_1 qp3_1
4 ...
5 index_NP size_NP r_NP px_NP py_NP pz_NP ux_NP uy_NP uz_NP wx_NP wy_NP wz_NP q0_NP
     q1_NP q2_NP q3_NP qp0_NP qp1_NP qp2_NP qp3_NP
```

Code 2.1: Format of the DEM particle restart file

Here, `NP` is the total number of particles, and is written in the first line. What follows is a sequence of lines, one for each particle, describing the state of motion and the geometrical characteristics of that particle.

`index` This is a sequential integer, growing from 0 to N_P.

`size` In this version, has to be 1 (spherical particle).

`r` The radius of the particle.

`px py pz` The particle initial position.

`ux uy uz` The particle initial velocity.

`wx wy wz` The particle initial rotational velocity.

`q0 q1 q2 q3` The particle initial orientation, expressed in quaternions. When unknown, the values (1 0 0 0) are sufficient.

`qp0 qp1 qp2 qp3` The particle initial rotational speed, expressed in quaternion rates. When unknown, the values (0 0 0 0) are sufficient.

In case this file is used to generate objects (immobile particles), all velocities and rotational velocities entries are ignored by the code.

2.5.2 Fluid restart file

Missing...

Chapter 3

Post-processing

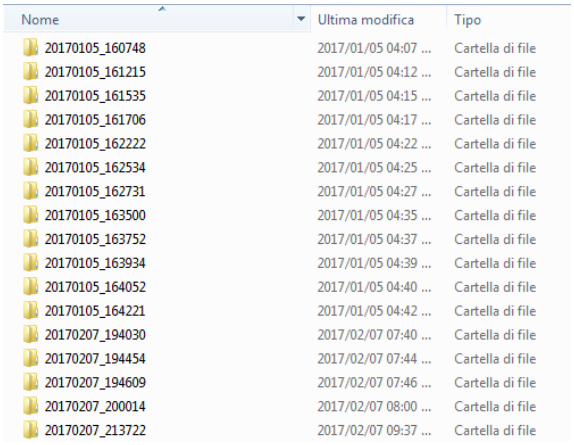

Figure 3.1: Sample of a home directory filled with simulation folders automatically names.

Each run of the code creates a simulation subfolder inside the folder specified by the option `-d path_to_result_directory` in the command line, see. The subfolder is named after the string given by the option `-n simulation_name`, always in the command line. If not specified, or if the name is set to `time`, the name will follow the format `YYYYMMDD_HHMMSS` with `YYYY` the year, `MM` the month, `DD` the day, `HH` the hour, `MM` the minute, and `SS` the second of the launching time.

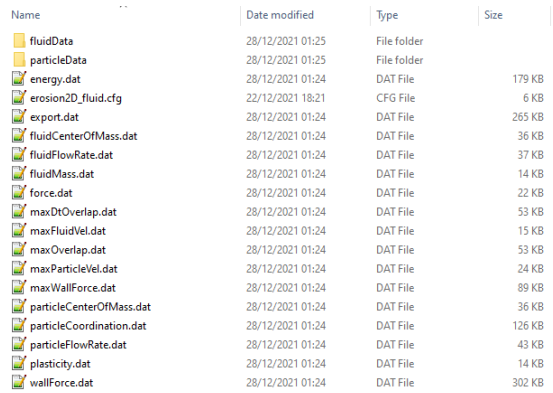

Figure 3.2: Sample of a simulation directory with the output files and folders.

Inside the simulation folder, a set of directories are automatically created, each containing one or more types of output. These are:

`fluidData` Folder that contains the fluid Paraview visualization files and the fluid restart files.

`particleData` Folder that contains the particle restart files, the particle Paraview files, and the object Paraview files.

A number of additional files are created and updated within the simulation folder. These are useful to track the state of a simulation and for post-processing. Setting a `ProblemName` might activate the generation of additional files. The subfolder also contains a copy of the configuration file. Note that any modification of the configuration files executed via the command line is not visible here.

`energy.dat` Contains the energy of fluid and particles, listed in columns with this order: time, particle mass, particle translational kinetic energy, particle rotational kinetic energy, particle potential energy, fluid mass, fluid kinetic energy, fluid potential energy, fluid mass inside particles, fluid kinetic energy inside particles, fluid potential energy inside particles.

`export.dat` A generic file that reports some of the info that are printed on screen by the code

`fluidCenterOfMass.dat` Contains the position of the fluid centre of mass, at each instant.

`particleCenterOfMass.dat` Contains the position of the particle centre of mass, at each instant.

`fluidFlowRate.dat` Contains the total fluid flow rate in each direction, at each instant.

`particleFlowRate.dat` Contains the total particle flow rate in each direction, at each instant.

`fluidMass.dat` Contains the total fluid mass, at each instant.

`force.dat` Contains the integral of all collisional forces exchanged by the particles, and of all hydrodynamic forces exchanged by particles and fluid at each instant.

`maxOverlap.dat` Contains the maximum and mean overlaps between particles (both in absolute terms, and scaled by the particle radius), registered at the printing time.

`maxDtOverlap.dat` Contains the maximum and mean overlaps between particles (both in absolute terms, and scaled by the particle radius), registered within two consecutive visualization intervals.

`maxFluidVel.dat` Contains the maximum fluid speed recorded at each instant.

`maxParticleVel.dat` Contains the maximum particle speed (translational and rotational) recorded at each instant.

`wallForce.dat` For each wall included in the simulation (e.g. boundary walls), reports the total force exerted by particles and fluid, at each instant.

`maxWallForce.dat` For each wall included in the simulation (e.g. boundary walls), reports the maximum total force exerted by particles and fluid, registered within two consecutive visualization intervals.

`plasticity.dat` Shows which percentage of the fluid cells is currently in a plastic state (pre-yield). The code defines nodes in a plastic state as those whose viscosity is more than 95% of the maximum allowed viscosity.

3.1 Paraview

Most output files are meant to be loaded using the open-source visualization program Paraview, which can be obtained at www.paraview.org/download. Paraview is relatively easy to use and has a large number of users, meaning finding answers on the Internet is usually quick. Once Paraview is downloaded and installed, the files can be opened by selecting *File* → *Open* and selecting the sequence of gather files in one of the data folders, such as `fluidData` or `particleData`. All files to be opened with Paraview are written using the Visualization ToolKit (VTK) standard. Useful information about how these files are structured can be found at www.vtk.org.

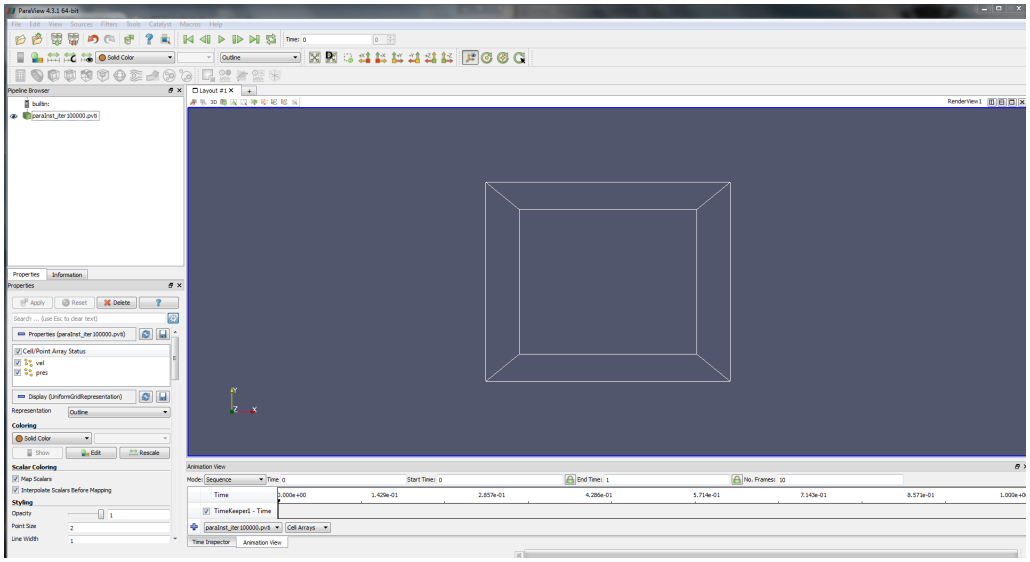

Figure 3.3: Initial state of visualization after loading a Paraview fluid file.

3.2 The fluid files

The fluid output files appear in the folder `fluidData` and are named with the format `fluidIIIIII.vti`, with the iteration at the time of printing appearing in place of `IIIIII`. This files cannot be used to restart the fluid simulation from a specific iteration, it is only for visualization purposes.

The data contained in the Paraview fluid is:

`v` The fluid velocity (vector, 3 components).

`pressure` The fluid pressure (scalar).

`dynVisc` The dynamic viscosity (scalar, for non-Newtonian or turbulent flows).

`friction` The friction coefficient (scalar, for frictional rheologies).

`type` A scalar helpful to identify the different types of nodes in the domain (0:fluid, 2:gas, 3:interface, 4: periodicity, 5: slip static wall, 6: slip moving wall, 7: no-slip static wall, 8: no-slip moving wall, 9:cylinder, 10:object, 11:topography, 12: outlet)

When loaded, the data looks initially as an empty cuboid. To visualize variables, it is very useful to use slices, or stream tracers.

3.3 The Lagrangian fluid files

The Lagrangian fluid output files also appear in the folder `fluidData` and are named with the format `fluidLagIIIIII.vti`, with the iteration at the time of printing appearing in place of `IIIIII`. This files cannot be used to restart the fluid simulation from a specific iteration, it is only for visualization purposes.

The data contained in the Lagrangian fluid file is identical to the ordinary fluid file:

`v` The fluid velocity (vector, 3 components).

`pressure` The fluid pressure (scalar).

`dynVisc` The dynamic viscosity (scalar, for non-Newtonian or turbulent flows).

`friction` The friction coefficient (scalar, for frictional rheologies).

`type` A scalar helpful to identify the different types of nodes in the domain (0:fluid, 2:gas, 3:interface, 4: periodicity, 5: slip static wall, 6: slip moving wall, 7: no-slip static wall, 8: no-slip moving wall, 9:cylinder, 10:object, 11:topography, 12: outlet)

When loaded, the data looks initially as a collection of points. These points are the centroid of the LBM lattice nodes for all fluid and interface nodes. To visualize variables, it is very useful to use, either as colourful glyphs spheres or a point gaussian representation (for scalar quantities), or arrow glyphs for vectors.

3.4 The particle and object Paraview file

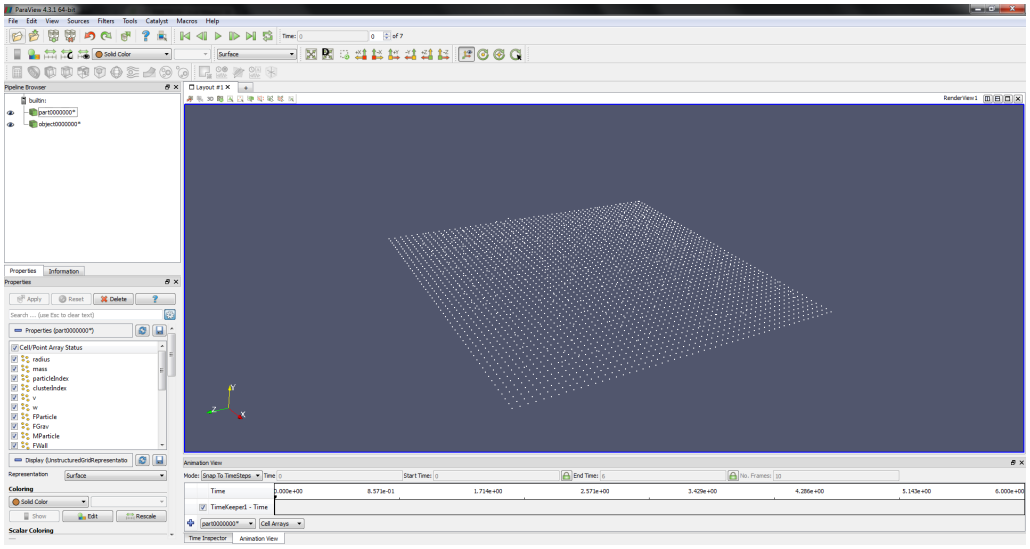

Figure 3.4: Structure of a particle immersed-boundary file once loaded in Paraview.

This files are useful for tracking the location, velocity, forces applied on, and any other relevant characteristic for all particles. They are unstructed file, where every particle/object is represented as a independent point. Each point comes with the following informations:

`radius` The radius of the particle.

`particleIndex` The particle index.

`clusterIndex` The cluster index. Ghost particles share the same cluster index with their mirror particle

`coordinationNumber` The amount of contacts each sphere is currently engaged in.

`v` The translational velocity (vector, 3 components).

`w` The rotational velocity (vector, 3 components).

`FParticle` The resultant force on the particle coming from the collision with other particles (vector, 3 components).

`FWall` The resultant force on the particle coming from the collision with walls and objects (vector, 3 components).

`FGrav` The particle weight (vector, 3 components).

`FHydro` The resultant force on the particle coming from the fluid (vector, 3 components).

`MParticle` The resultant torque on the particle coming from the collision with other particles (vector, 3 components).

`MWall` The resultant torque on the particle coming from the collision with walls and objects(vector, 3 components).

`MFluid` The resultant torque on the particle coming from the fluid (vector, 3 components).

The visualization can be enhanced using the representation style called *Point Gaussian*, which can be activated using the representation toolbar, highlighted in red in figure 3.5. To achieve a nice visualization, one can use the *properties* toolbar (green in the figure), selecting the field `radius` as input for the option *Gaussian Scale Array*. The *Scale Transfer Function* should be already set, otherwise a unitary scale does the trick. In the figure, the particles have been coloured using the filed `clusterIndex`, which attaches the same colour to the original particle and their corresponding ghost particles.

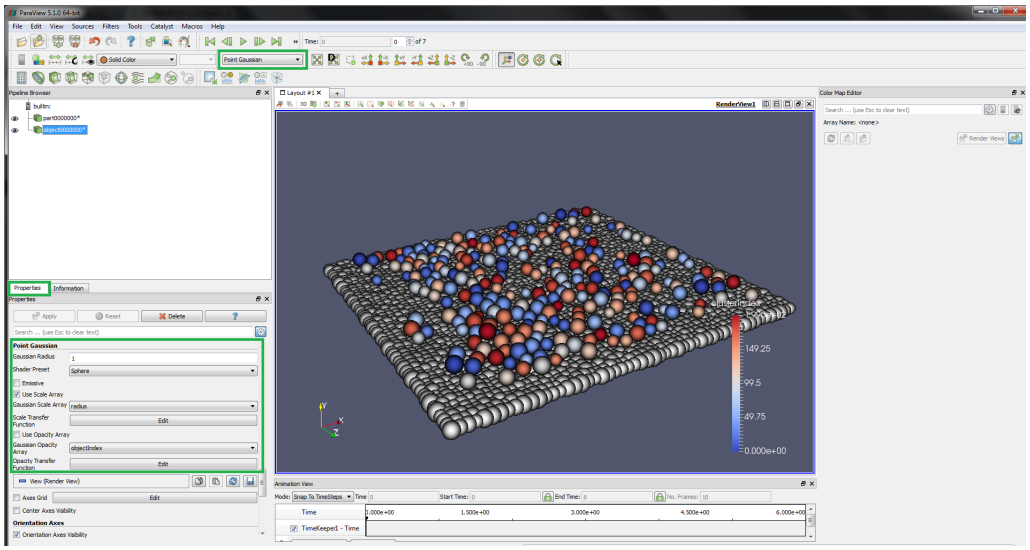

Figure 3.5: Visualization of particles and objects using the *Point Gaussian* representation.

Chapter 4

Tutorials

A set of tutorials is given to the user. The first tutorial guides the user through the configuration of a simple fluid-only LBM simulation.

4.1 Tutorial 1: Laminar channel flow (Poiseuille flow)

This first example consists in developing a very basic flow using LBM. We consider a laminar flow between two parallel walls, driven by a constant pressure gradient, see fig.4.1. The flow has periodic

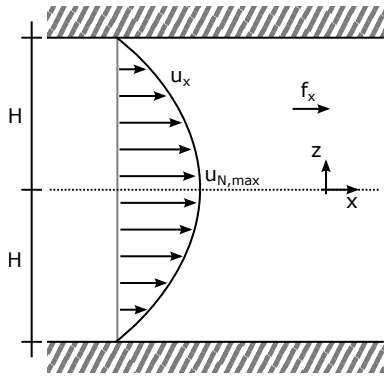

Figure 4.1: Geometry for the Newtonian Poiseuille flow.

boundary conditions in x and y directions, while in z direction a no-slip condition is applied. This type of flow is called Poiseuille flow. For this case, an analytical solution of the Navier-Stokes equations exists. The theory can be found in classical fluid mechanics books. From the analytical solution, it is known that the velocity profile in the vertical direction assumes a parabolic shape. This profile refers to a flow at $\mathrm{Re} = 12.5$, where $\mathrm{Re} = U\rho_\mathrm{f}H/\nu$ is the Reynolds number, based on the mean velocity U, half channel height H and the dynamic viscosity ν. The maximum velocity at the centre of the channel should be:

$$u_{\mathrm{N,max}} = \frac{f_x \rho_\mathrm{f} H^2}{2\nu} \tag{4.1}$$

with f_x the imposed pressure gradient.

We impose density $\rho_\mathrm{f} = 1$ kg/m^3, dynamic viscosity $\nu = 0.2$ kg/ms, external acceleration $f_x = 1$ m/s^2, and channel half width $H = 2$ m. With these settings, the maximum velocity should be $u_{\mathrm{N,max}} = 2.5$ m/s. We discretize using a cuboid with dimensions $1 \times 10 \times 200$ in the three directions.

4.1.1 Setting the environment

All configurations file should be set inside the `tutorials/laminar_flow` folder. Here the DEM part is deactivated by choosing empty particles (particleFile=./null.dat) and object files (objectFile=./null.dat). The null.dat files must be provided (these are files containing a single 0), otherwise the code will

stop execution. The configuration file is already ready for use (`laminar_flow.cfg`), but the most important features are described here. As an exercise, the user can try to reconstruct it from scratch.
We are using only the LBM solver, and we do not need the free-surface algorithm for this, so we set:

- `demSolver=0` No DEM.
- `lbSolver=1` LBM is on.
- `freeSurfaceSolver=0`.
- `forceFieldSolver=1` We need an external forcing for our flow to develop.

In this case, a minimal setup is necessary for the output files:

- `maxTime=20.0`; The code will run for 20 s simulation time.
- `restartFluid=false`; Computation is from scratch, no restart file is required.
- `fluidRecycleExpTime=false`; No restart files will be generated.
- `screenExpTime=0.1`; The code will show a state line on screen every 0.1 s simulation time.
- `fluidExpTime=1.0`; The code will generate a fluid Paraview file every 1.0 s simulation time.
- `fluidLagrangianExpTime=0`; The code will not generate Lagrangian fluid Paraview files.

The boundary conditions are periodicity in x ad y, and no-slip walls on z. This can be set by specifying:

- `boundary0=periodic`;
- `boundary1=periodic`;
- `boundary2=periodic`;
- `boundary3=periodic`;
- `boundary4=stat_wall`;
- `boundary5=stat_wall`;

Domain setup:

- `latticeSpacing=0.01`. See below.
- `domainSizeX=0.1` The lattice spacing in x is equal to 10 lattice spacings, so that we can visualise the flow field.
- `domainSizeY=0.01` The lattice spacing in y is equal to the lattice spacing, so that we work with a single sheet of fluid cells.
- `domainSizeZ=2.0`, so that our half-channel height L is unitary.
- `fluidMinX=-100.0` and `fluidMaxX=100.0`. Since we set the minimum and maximum borders of the fluid to be beyond the domain borders, the fluid will fill the whole space (same for directions y and z).

The physical parameters are the following:

- `rheologyModel=NEWTONIAN`. We use a simple Newtonian model.
- `fluidDensity=1.0`. The fluid mass density ρ_f
- `initVisc=0.2`. The dynamic viscosity ν.
- `forceX=1.0`. The imposed pressure gradient f_x.
- `turbulenceSolver=0`. No need for a turbulence model.

The code is configured by setting the following options:

- `fluidTimeStep=0` The code will choose the time step automatically, so that the relaxation time τ will be unitary. In this case, this corresponds to $\Delta t = 8.3 \cdot 10^{-5}$;

4.1.2 Running the code

To run the code, we need to create a folder somewhere that can be reached with a terminal (e.g., the Cygwin folder). We need to put in this folder:

- The configuration file `laminar_flow.cfg`.
- A file with just 0 in it, to indicate that no particles are used.
- A folder to store outputs (which can be simply called "results").
- The executable `hybird.exe`.

At the end, the folder should look something like this:

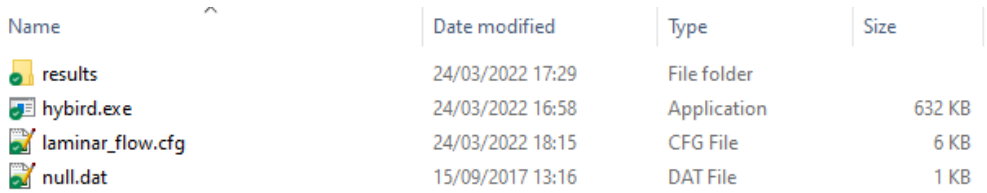

Figure 4.2: Structure of the folder to run the laminar flow tutorial.

The following command should be issued in the terminal, one this has positioned inside the `examples/laminarFlow` folder. To run it, execute the command line:

```
./hybird.exe -c incline_flow.cfg -d results -n time
```

Code 4.1: Running the first tutorial

The code will run and it will print on screen the value of the maximum velocity. You can see it increasing slowly towards convergence.

The code will also create a subfolder with a timestamp inside the `results` folder. To visualize the results, open Paraview and load the fluid files (see Sec.3.2). Loading the Paraview files will

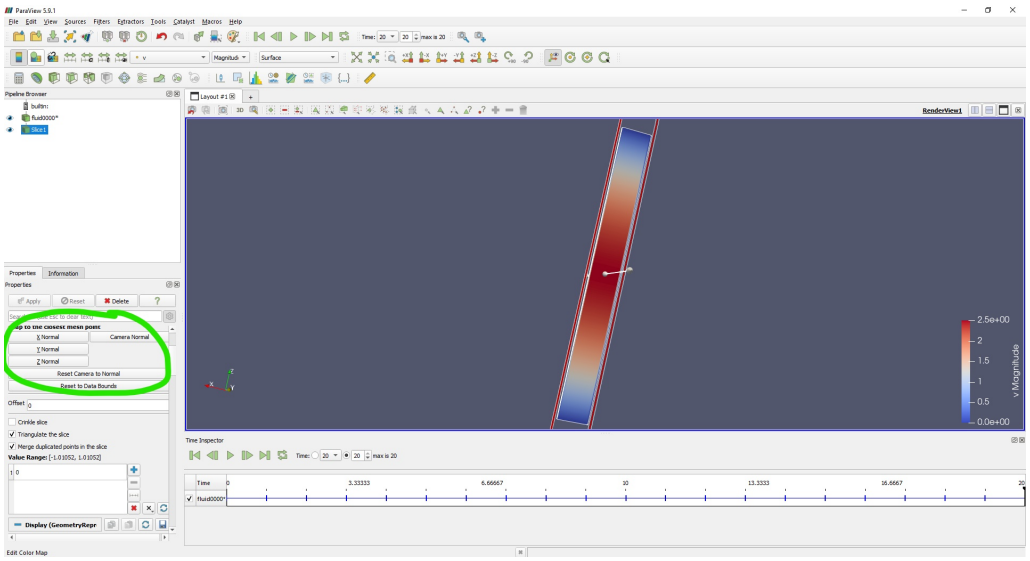

Figure 4.3: Field visualization of the laminar Poiseuille flow.

result in a cuboid dataset. To visualize the flow, the easiest way is by using a section, provided by the filter *Slice*, shown in the figure with a green square. The slice position is set by the *Plane Parameters*, highlighted in Fig.4.3. This can be automatically done by pressing the *Y Normal* option. The slice can be coloured using the field `v (Magnitude)`.

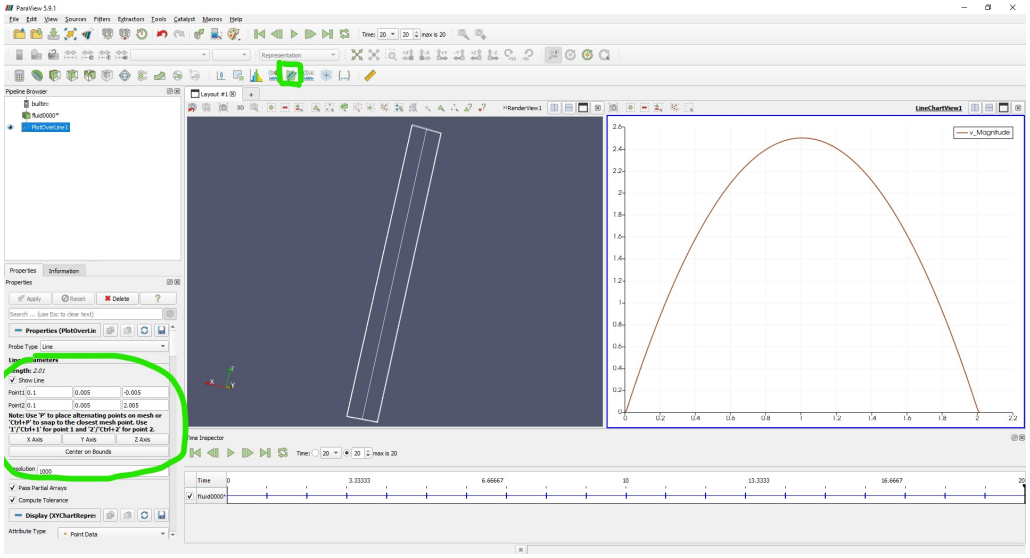

Figure 4.4: Graphical visualization of the laminar Poiseuille flow.

Another way to visualize the data is by obtaining a graph. This can be done by selecting the filter *Plot Over Line*, see the green highlights in Fig. 4.4. Once again, the settings in *Line Parameter* should be defined, but this is done automatically by pressing *Z Axis*. The profile should be a parabola, with maximum $u = 2.5$ for $z = 1.0$.

4.2 Tutorial 2: Intro to non-Newtonian: Bingham plastic

This second example introduces non-Newtonian fluids in LBM. *Non-Newtonian fluids* is an umbrella term for any fluid that does not exhibit a linear relationship between stress and strain rate.

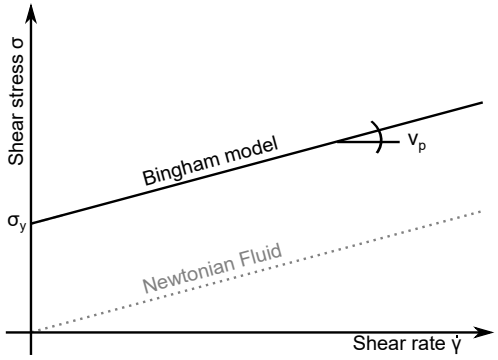

Figure 4.5: The Bingham plastic.

Here, we work with a relatively simple non-Newtonian fluid called a *Bingham plastic*. This is a fluid that does not shear unless a minimum stress, called a *yield stress*, σ_y is reached. The yield stress is therefore a material parameter. Note that this is a shear stress, but we use a σ symbol in order to distinguish it from the relaxation time τ. When the applied stress exceeds the yield stress, the Bingham plastic behaves according to the following law:

$$\sigma = \sigma_y + \nu_\mathrm{p}\dot{\gamma} \tag{4.2}$$

where σ is the flow resistance to shear, $\dot{\gamma}$ is the shear rate, and ν_p is the plastic viscosity, which is the second material parameter (the first being the yield stress). The model is depicted in Fig. 4.5.

In analogy with Newtonian fluids, we call the ratio between shear stress and shear rate an (apparent) viscosity:

$$\nu = \frac{\sigma}{\dot{\gamma}} = \nu_\mathrm{p} + \frac{\sigma_y}{\dot{\gamma}} \tag{4.3}$$

When the shear rate approaches zero (the no-flow condition), the apparent viscosity diverges to $+\infty$.

In this tutorial, we consider a free-surface flow on an infinite incline with constant slope θ, driven by gravity, see fig.4.6. We set a reference frame such that the x direction is aligned with

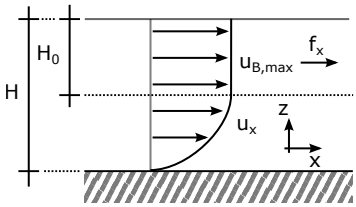

Figure 4.6: Geometry for the Bingham incline flow.

the incline, and th imposed pressure gradient (the external forcing f_x) is also aligned with x. The domain has periodic boundary conditions in the x and y directions. In the z direction, a no-slip condition is applied at the bottom, and the free-surface condition is applied at the top. This type of flow is analogous to the Poiseuille flow, because the free-surface condition, being stationary, is analogous to a symmetry condition.

For this case, an analytical solution of the Navier-Stokes equations (adapted to the Bingham law) exists. Since the shear stress in the fluid must exceed the yield stress, we can only have flow if the stress induced by gravity is larger than the yield stress. This is true if

$$H > \frac{\sigma_y}{\rho_\mathrm{f} f_x} \tag{4.4}$$

or, in other terms,

$$\sigma_y < \rho_\mathrm{f} f_x H \tag{4.5}$$

If the above is true, the type of flow that is generated consists of a plug (i.e., an area with equal velocity and no internal deformation) lying on top of a shear layer, see Fig.4.6. The thickness of the plug can again be derived using simple equilibrium conditions, and is:

$$H_0 = \frac{\sigma_y}{\rho_\mathrm{f} f_x} \tag{4.6}$$

In the shear layer ($z < H - H_0$), velocity follows a parabolic profile

$$u_x(z) = \frac{\rho_\mathrm{f} f_x}{\nu_\mathrm{p}} z \left(H - H_0 - \frac{z}{2} \right), \tag{4.7}$$

From this result, the velocity of the fluid in the plug region can be calculated as

$$u_x\left(H_0\right) = u_{\mathrm{B,max}} = \frac{\rho_\mathrm{f} f_x}{2\nu_\mathrm{p}} \left(H - H_0\right)^2. \tag{4.8}$$

We will try to recover this result using LBM.

The main issue here is that the basic LBM formulation is suited only for Newtonian fluids, i.e., fluids with a constant finite viscosity. The LBM code cannot reproduce this result exactly, because the simulation of a Bingham model requires the treatment of the viscosity ν as a variable, which reaches its minimum value ν_p for $\dot{\gamma} \to \infty$ and diverges for $\dot{\gamma} \to 0$. Such a large range of viscosity variation would cause instability in the code. This is managed by bounding the viscosity range by two values, ν_min and ν_max. How these values are derived is explained in Sec. 5.2.1, but it is important to highlight that they are a function of the time step Δt and the lattice resolution ΔS, and can therefore be controlled, albeit with constrains, by the user.

The resulting approximated Bingham rheology is a tri-linear model, as sketched in Fig. 4.7. However, if $\nu_\mathrm{p} > \nu_\mathrm{min}$, which is usually easy to obtain by adapting the time step Δt, the rheology reduces to a bi-linear model, with a range of simulated viscosities wide enough to yield a good approximation for most cases.

In this tutorial we impose density $\rho_\mathrm{f} = 1000$ kg/m^3, dynamic plastic viscosity $\nu_\mathrm{p} = 1.0$ kg/ms, external forcing $f_x = 1$ m/s^2, and flow height width $H = 1$ m. With these settings, the maximum velocity should be $u_\mathrm{max} = 1.25$ m/s. To speed up the computations, we discretize using a cuboid with dimensions $1 \times 1 \times 120$ in the three directions, therefore a 1D domain. Only the first 100 nodes in direction z,will be filled with fluid, the rest will be "gas" cells above the free surface.

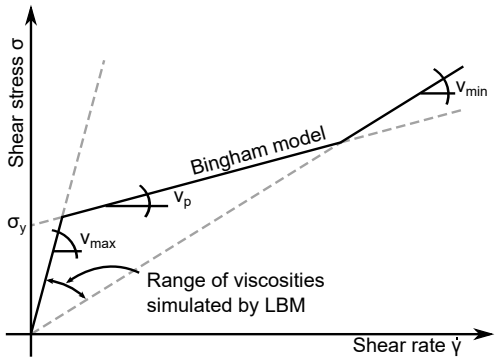

Figure 4.7: The trilinear model obtained by approximating the Bingham plastic.

4.2.1 Setting the environment

All configurations file should be set inside the `tutorials/Bingham_flow` folder. Here too (see Tutorial 1, Sec. 4.1) the DEM part is deactivated. The configuration file is already ready for use (`bingham_flow.cfg`), but the most important features are described here.
We are using only the LBM solver, and we do need the free-surface algorithm for this, so we set:

- `demSolver=0` No DEM.
- `lbSolver=1` LBM is on.
- `freeSurfaceSolver=1` free-surface solver is on. However, the free surface will not evolve, because the initial condition corresponds already to the stationary condition.
- `forceFieldSolver=1` We need an external forcing for our flow to develop.

In this case, a minimal setup is necessary for the output files:

- `maxTime=20.0`; The code will run for 20 s simulation time.
- `restartFluid=false`; Computation is from scratch, no restart file is required.
- `fluidRecycleExpTime=false`; No restart files will be generated.
- `screenExpTime=0.1`; The code will show a state line on screen every 0.1 s simulation time.
- `fluidExpTime=0`; The code will not generate fluid Paraview files.
- `fluidLagrangianExpTime=1.0`; The code will generate Lagrangian fluid Paraview files every 1.0 s simulation time.

The domainboundary conditions are periodicity in x ad y, and no-slip walls on z. Note that in z the upper boundary condition will not be used because the fluid will not touch this boundary. This can be set by specifying:

- `boundary0=periodic`;
- `boundary1=periodic`;
- `boundary2=periodic`;
- `boundary3=periodic`;
- `boundary4=stat_wall`;
- `boundary5=stat_wall`;

Domain setup:

- `latticeSpacing=0.002`. This is ΔSSee below for its influence on the flow viscosity resolution.

- `domainSizeX=0.002` The lattice spacing in x is equal to 1 lattice spacings, so that we work with a single column of fluid cells.
- `domainSizeY=0.0002` The lattice spacing in y is equal to the lattice spacing, so that we work with a single column of fluid cells.
- `domainSizeZ=0.12`, this is the flow height $H = 0.1$ m plus a thin layer of "gas" cells on top.
- `fluidMinX=-100.0` and `fluidMaxX=100.0`. Since we set the minimum and maximum borders of the fluid to be beyond the domain borders, the fluid will fill the whole space (same for direction y).
- `fluidMinZ=0.0` and `fluidMaxZ=0.1`. We want the fluid to fill the domain up to the prescribed flow height $H = 0.1$ m.

The physical parameters are the following:

- `rheologyModel=BINGHAM`. We use a Bingham plastic model.
- `fluidDensity=1.000`. The fluid mass density ρ_{f}
- `initVisc=0.2`. The initial dynamic viscosity, set only for initialization purposes.
- `plasticVisc=1.0`. The plastic dynamic viscosity ν_{p}
- `yieldStress=0.2`. The yield stress σ_{y}
- `forceX=1.0`. The imposed pressure gradient f_x.
- `turbulenceSolver=0`. Turbulence models should never be used for non-Newtonian fluids (although the code will not stop you from using this option).

The code is configured by setting the following options:

- `minTau=0.5005` A typical minimum value for the relaxation time $\tau_{\min}$ for stability purposes
- `maxTau=1.0` The theoretical maximum value for the relaxation time $\tau_{\max}$ for accuracy and stability purposes.
- `fluidTimeStep=1E-5` The time step Δt. In this case it cannot be chosen automatically, since it is a key choice with respect to stability and accuracy.

Based on the choices of time step Δt and lattice spacing ΔS, the code will work with the following boundaries for the dynamic viscosity:

$$\nu_{\min} = \frac{\tau_{\min} - 1/2}{3} \frac{\rho_{\mathrm{ref}} \Delta S^2}{\Delta t} \simeq 0.067 \ \frac{\mathrm{kg}}{\mathrm{ms}} \tag{4.9}$$

$$\nu_{\max} = \frac{\tau_{\max} - 1/2}{3} \frac{\rho_{\mathrm{ref}} \Delta S^2}{\Delta t} \simeq 67 \ \frac{\mathrm{kg}}{\mathrm{ms}} \tag{4.10}$$

Note how $\nu_{\mathrm{p}} > \nu_{\min}$, therefore the minimum cut-off will have no influence on the results. We are here using a bilinear model, as described in Fig. 4.7. The maximum viscosity, on the other hand, will be used everywhere in the domain where the theoretical solution will prescribe $\nu \to \infty$, i.e. the plug area. The plug are will therefore behave like a very viscous fluid, rather than a solid, and a minimum amount of shear will occur there. Note that reducing the time step will result in higher cut-off values for both $\nu_{\max}$ and $\nu_{\min}$, thus yielding better results. However, this is done at the expenses of a much longer computational time. Thus, the time-step choice is usually a compromise between accuracy and simulation speed. The best accuracy is achieved when $\nu_{\min} = \nu_{\mathrm{p}}$.

The tutorial folder contains an excel sheet "scaling_Bingham.xlsx", which can be used to explore the effect of a change in the parameter setup on the viscosity range.

4.2.2 Running the code

To run the code, the procedure is identical to the one explained in Tutorial 1 (Sec. 4.1). However, the configuration file is now called `bingham_flow.cfg`. Therefore, the command to be issued in the terminal, is:

```
./hybird.exe -c bingham_flow.cfg -d results -n time
```

Code 4.2: Running the second tutorial

The code will run and it will print on screen the value of the maximum velocity. You can see it increasing slowly towards convergence. You can also check the amount of material that is in a "plastic" state, i.e. unyielded. Of course, the approximated implementation of viscosity makes so that the "unyielded" material will be the one that has reached the maximum allowable viscosity $\nu_{\max}$.

The code will also create a subfolder with a timestamp inside the `results` folder. To visualize the results, open Paraview and load the Lagrangian fluid files (see Sec.3.2). Note that Lagrangian and Eulerian fluid files differ in the style of visualization, not in the information that they display. The Lagrangian files display every fluid cell as a point. This precludes the possibility to draw contours and interrogate variables over lines, but they open up other ways to visualize variables. As an exercize, try to run the code again, this time asking to generate Eulerian files as in Tutorial 1, and visualize the velocity profile. Here we focus on the type of representation that works well

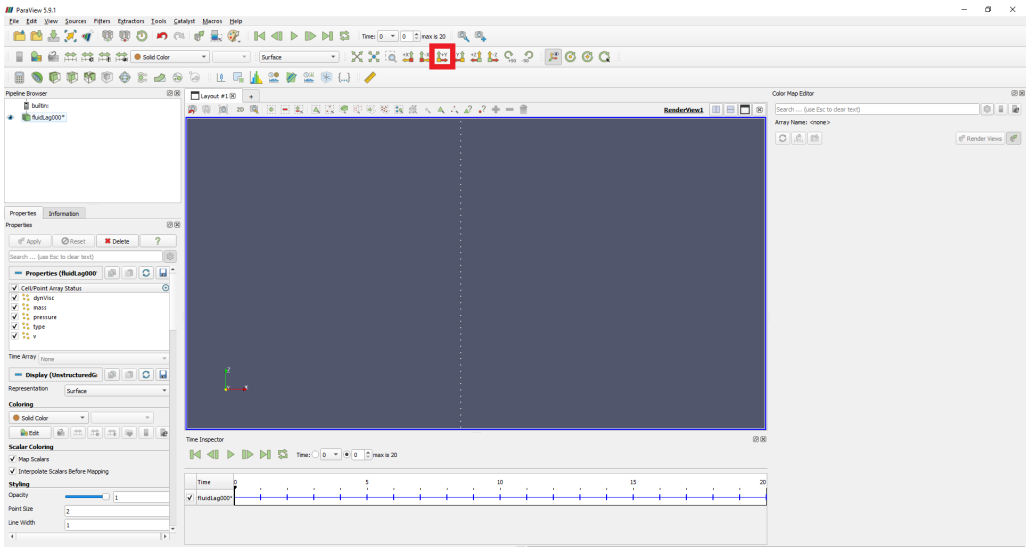

Figure 4.8: Lagrangian visualization of the Bingham channel flow.

with Lagrangian files. Loading these files will result in a column of fluid nodes, displayed as points dataset, see Fig. 4.8. Because the nodes are aligned, Paraview will use automatically a 2D representation. However, we should switch the reference plane to xz, by selecting the "Set view direction to +Y" button, highlighted in red.

To visualize the flow velocity, we can use arrows, which are provided by the filter *Glyph*, shown in Fig.4.3 with a green square. The glyph filter needs a vector for scaling its size, and a vector for scaling its orientation. In both cases, we should choose the filed "v", which contains the velocity vector, see the red squares in Fig.4.3. We can then press the "update" button (blue square) to rescale the arrows based automatically. To activate the filter, it is necessary to press the "Apply" button. The arrow can also be coloured using the field `v (Magnitude)`, see the yellow square. In the first time step, no arrows will be displayed, as velocity is zero everywhere. If we press "play", Paraview will automatically run through all available files. It might be necessary to stop and rescale the arrow glyphs.

We can also visualize scalars using glyphs. As an example, we could use spheres, as depicted in Fig. 4.10. Change the glyph type from "arrow" to "sphere", blue square. Set their radius to be equal to half the lattice spacing ($0.002/2 = 0.001$ m), then reset the orientation and scale arrays, and choose a "scale factor" of 1 (red squares), and press "Apply". At this point, a column of identical spheres should be created. We can colour these spheres based on a scalar field, for

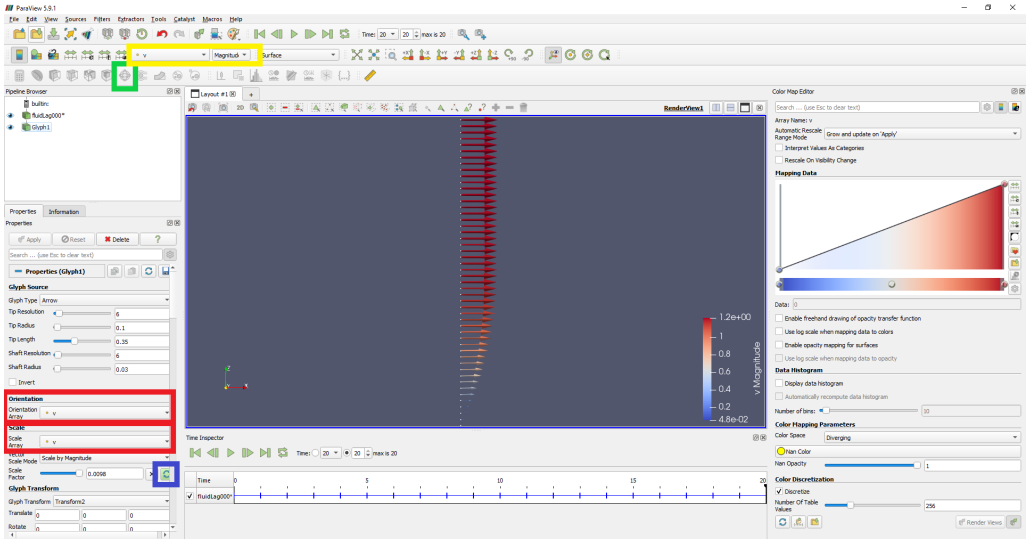

Figure 4.9: Graphical visualization of the velocity field in the bingham free-surface flow.

example viscosity. Since viscosity has a large variability, we can obtain a better representation by

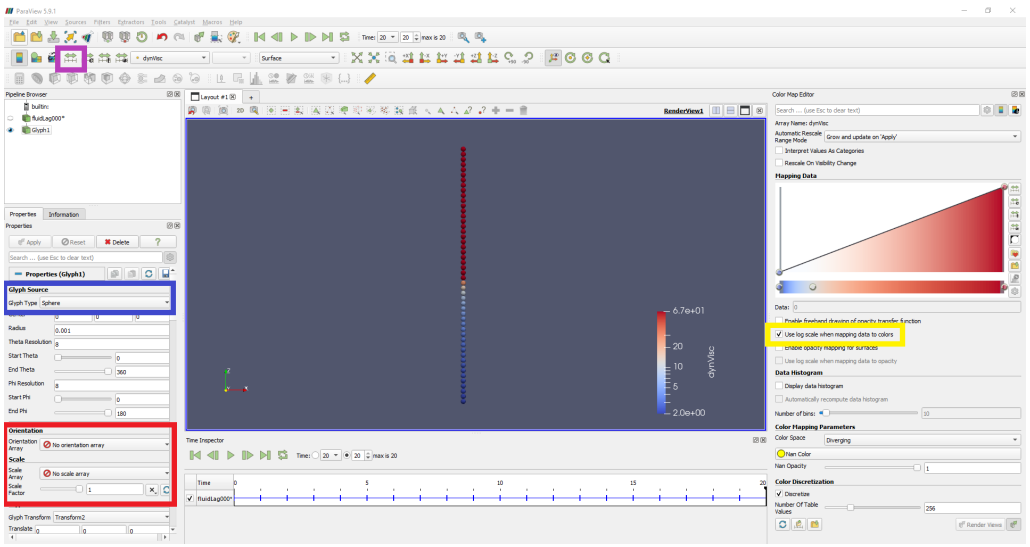

Figure 4.10: Graphical visualization of the viscosity field in the bingham free-surface flow.

choosing a log colour scale (yellow square). It is now wise to rescale the colours to the automatic range (purple square). Note how at the end of the simulation, the upper half of the fluid (the plug) has converged to the maximum allowable value of viscosity.

4.3 Tutorial 3: Granular flow down an incline

This tutorial reproduces a steady, free-surface granular flow on a simple infinite incline of constant slope θ, as shown on Fig. 4.11. The flow thickness H is assumed constant, and the inclination is set by tilting the principal axis with respect to an imposed pressure gradient g, i.e. $g_x = g\sin\theta$, $g_z = g\cos\theta$. The boundary conditions are no-slip at the bottom wall, and zero pressure at the top ($z = H$).

We assume that the granular flow behavior can be well represented by the $\mu(I)$ rheology. Following **(author?)** [4], the rheology is implemented by locally defining the viscosity as:

$$\nu = \frac{p\mu(I)}{\dot{\gamma}}, \tag{4.11}$$

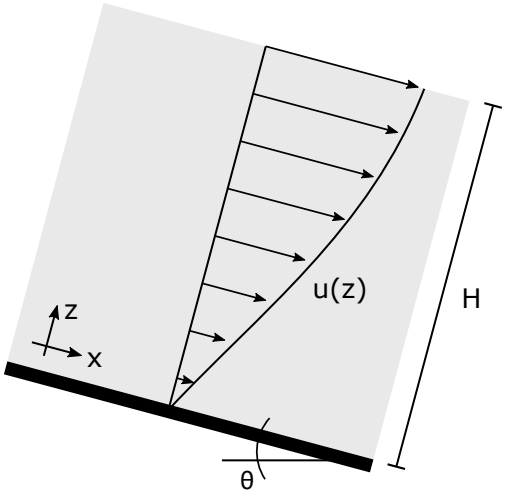

Figure 4.11: Geometry of the granular flow on an incline.

where $\dot{\gamma}$ is the second invariant of the shear rate tensor, and I is the inertial number, defined as:

$$I = \frac{\dot{\gamma} d}{P/\rho_s}, \tag{4.12}$$

where d and ρ_s are diameter and density of the grains, respectively. The dependence of the friction coefficient on the inertial number is phenomenologically set as:

$$\mu(I) = \mu_s + \frac{\mu_2 - \mu_s}{I_0/I + 1}. \tag{4.13}$$

The $\mu(I)$ rheology needs a set of five material-dependent parameters: d, ρ_s, the critical friction coefficient μ_s, its asymptotic value μ_2, and the constant I_0. The three last parameter can be obtained using an imposed-pressure rheometer.

In LBM, in analogy to other non-Newtonian rheologies, the $\mu(I)$ rheology can be reproduced, but a few restrictions apply. Whenever the shear rate is very low, the viscosity diverges. This is the way the fluid reproduces the yield threshold implicit to the Coulomb law. Areas where this threshold is not reached are static, hence the divergence of the viscosity. A second problem arises when pressure is close to zero, e.g., at the free surface. In this case,viscosity reduces to zero.

On an infinite incline, and at steady state, the conservation of momentum in the two directions leads to writing:

$$\tau_{xz} = \rho g \sin\theta H(1 - z/H), \qquad p = \rho g \cos\theta H(1 - z/H), \tag{4.14}$$

which implies that the friction coefficient, defined as $\mu(I) = \tau/p$ is constant for a given slope θ and equal to $\mu(I) = \tan\theta$ and that the inertial number is itself constant, and equal to

$$I(\theta) = I_0 \frac{\tan\theta - \mu_0}{\mu_2 - \tan\theta}. \tag{4.15}$$

For more details, please see [5]. From the definition of the inertial number (Eq.4.12) one obtains the shear rate as

$$\frac{\partial u}{\partial z} = \frac{I(\theta)}{d} \left(gH\cos\theta\right)^{\frac{1}{2}} \left(1 - z/H\right)^{\frac{1}{2}}, \tag{4.16}$$

which can be integrated to yield the velocity profile:

$$u(z) = \frac{2}{3} \frac{I(\theta)}{d} \left(gH^3\cos\theta\right)^{\frac{1}{2}} \left(1 - (1 - z/H)^{\frac{3}{2}}\right), \tag{4.17}$$

and the maximum velocity (at the surface) is

$$u_{\max} = u(H) = \frac{2}{3} \frac{I(\theta)}{d} \left(gH^3\cos\theta\right)^{\frac{1}{2}}. \tag{4.18}$$

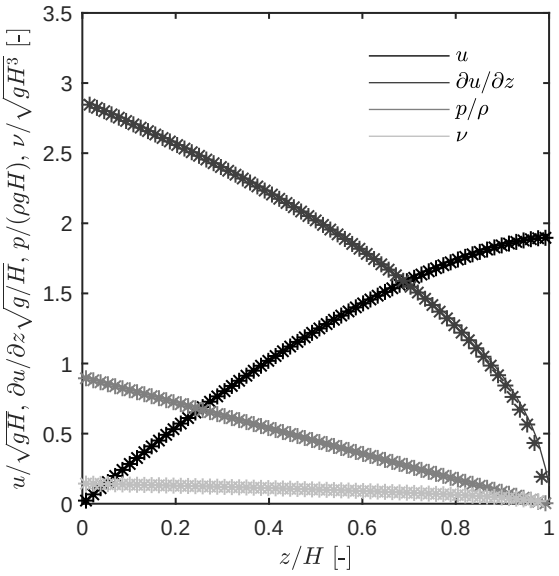

Figure 4.12: Velocity, pressure, and equivalent viscosity profiles.

In this tutorial, we retrieve this solution by applying the same geometrical setup as in Tutorial 2 (Sec. 4.2)

The simulation is halted after stationary conditions are reached. The velocity profile $u(z)$ that should be obtained is illustrated, for a slope with $\theta = 0.43$, in Fig. 4.12. If $\tan\theta < \mu_0$ no flow is expected to occur, and Eq. 4.15 predicts a vanishing inertial number for $\tan\theta \to \mu_0$. The regularization of viscosity and pressure implies that a slow creeping flow will be reproduced when $\tan\theta < \mu_0$. However, the magnitude of this flow is small enough to be considered negligible. Fig. 4.13 shows the velocity at the free surface (the maximum) for variable θ

4.3.1 Setting the environment

All configurations file should be set inside the `tutorials/granular_flow` folder. It is up to the user on whether to visualize the flow using Eulerian or Lagrangian files. The configuration file is already ready for use (`granular_flow.cfg`), and is very similar to the one used for the previous tutorial. Only the differences are described in the following:
Domain setup:

- `forceX=3.99`. The imposed pressure gradient parallel to the incline $g\sin(\theta)$, with $\theta = 0.42$ (24°).
- `forceZ=-8.95`. The imposed pressure gradient orthogonal to the incline $g\cos(\theta)$, with $\theta = 0.42$ (24°).

The physical parameters are the following:

- `rheologyModel=MUI`. We use a simple Newtonian model.
- `fluidDensity=1.5e3`. The bulk granular density ρ_b (a typical value)
- `initVisc=0.2`. The initial dynamic viscosity ν (will be used to initialize viscosity)
- `frictionCoefFluid=0.382`. The base friction coefficient, $\mu_0 = \tan(20.9°)$
- `deltaFriction=0.42`. The difference $\mu_2 - \mu_0 = \tan(32.76°) - \tan(20.9°)$
- `baseInertial=0.279`. The base Inertial number I_0.
- `particleDiameterFluid=0.00054`. The diameter of the particles constituting the granular continuum.

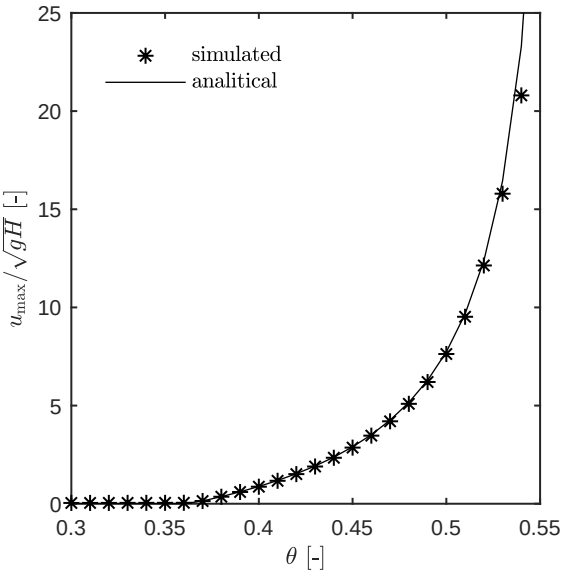

Figure 4.13: Variation of the free-surface velocity as a function of the incline slope.

- `turbulenceSolver=0`. The turbulence model should not be used for non-Newtonian models.

The code is configured by setting the following options:

- `fluidTimeStep=1E-5` The time step Δt. In this case, it cannot be chosen automatically, since it is a key choice with respect to stability and accuracy.

Based on the choices of time step Δt and lattice spacing ΔS, the code will work with the same boundaries for the dynamic viscosity as in the previous tutorial. The tutorial folder contains an Excel sheet "scaling_granular.xlsx", which can be used to explore the effect of a change in the parameter setup on the viscosity range.

4.4 Tutorial 4: Granular column creation - discrete

This tutorial introduces the DEM part of the code. DEM is conceptually very different from LBM, and is more difficult to initialize. The code needs to be given as input the initial position and velocity of every particle that it simulates. To do that, an additional file needs to be provided via the option `particleFile`. The structure of this file is described in Sec. 2.5.1. There are two ways to create this file: (i) using an external editor (a python or MATLAB script, a spreadsheet...) or (ii) use a "recycle" file produced by another simulation. In this tutorial, we aim at the former strategy, and we produce an initial sample of particles using a MATLAB script.

Let us first have a look at the geometry. We want to create a sample of densely-packed particle of mean diameter $d = 0.002$ mm, starting from a loose collection of particles, see Fig. 4.14(a). The particles will have a radius randomly picked within an interval of $d \pm 5\%$. The particles will be placed inside a container of lateral size $l = 0.035$ mm. The container width is slightly wider than the particles: $w = 1.2d$. In this way, the particle will be arranged in a quasi-2D packing, as shown in Fig. 4.14(b).

The script for particle generation is very easy to read, and can be found in the folder. It generates a file called `particle_init.dat`. The structure of this file is explained in detail in Sec. 2.5.1. Once created, move this file in the main simulation folder.

We want to generate a dense packing. To do this, we will use a simple linear contact model, whose stiffness does not refer to any specific material. The chosen value, $2 \cdot 10^3$ N/m is chosen just high enough that the particle will not overlap by more than a few percentages of their radius. We will exploit a numerical technique to obtain a dense sample: deactivating friction between particles. This will be done by setting to zero every parameter related to friction. In the next tutorial, friction will be reactivated to simulate a collapse problem.

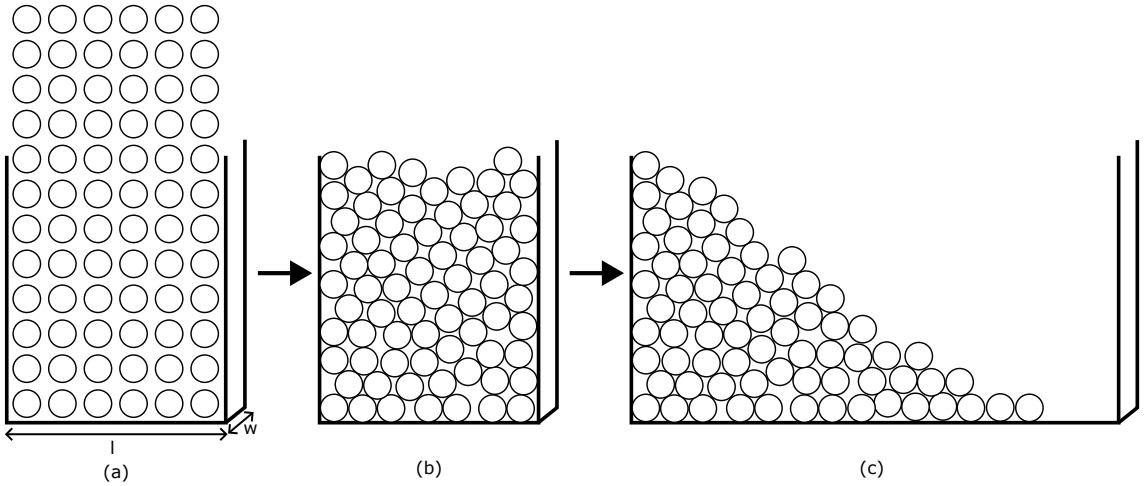

Figure 4.14: The first part of the simulation: creation of a dense granular column by compaction under gravity, and collapse by removal of the right wall (next tutorial).

4.4.1 Setting the environment

All configurations file should be set inside the `tutorials/granular_column` folder. Here the DEM part is active, and as such we will need a particle file, generated using the MATLAB script. There are no immobile particles, and therefore the object file will be empty, as in the previous simulations (objectFile=./null.dat). Therefore, a null.dat file must be provided. The configuration file is already ready for use (`granular_column.cfg`), but the most important features are described here. As an exercise, the user can try to reconstruct it from scratch.
We are using only the DEM solver, and we do not need the free-surface algorithm for this, so we set:

- `demSolver=1` The DEM is on.
- `lbSolver=0` no LBM.
- `freeSurfaceSolver=0` This is related to the LBM part and is irrelevant here.
- `forceFieldSolver=1` We need an external forcing (vertical gravity).
- `staticFrictionSolver=0` We do not need static friction. In fact, turing this off will help obtaining a dense sample at the end of the simulation.

We want to have both files for visualization, and recycle files that will be needed for the next tutorial:

- `maxTime=1.0`; The code will run for 1 s simulation time, which should be sufficient to obtain a dense sample.
- `screenExpTime=0.01`; The code will show a state line on screen every 0.01 s simulation time.
- `partExpTime=0.01`; The code will generate a particle Paraview file every 0.01 s simulation time.
- `partRecycleExpTime=0.1`; The code will generate a particle recycle file every 0.1 s simulation time.

The boundary conditions are solid walls in every direction. This can be set by specifying:

- `boundary0=stat_wall`;
- `boundary1=stat_wall`;
- `boundary2=stat_wall`;

- `boundary3=stat_wall;`
- `boundary4=stat_wall;`
- `boundary5=stat_wall;`

Domain setup:

- `latticeSpacing=0.00024` This is not an important parameter here, but it is good practice to choose a value much smaller than the minimum dimension of the simulation domain. In our case, it is ten times smaller than the dimension in y (see below)
- `domainSizeX=0.035` The domain in x is 0.035m.
- `domainSizeY=0.01` The domain in y is 0.0024m, which is 1.2 times the mean particle diameter, so just a bit larger.
- `domainSizeZ=0.2` The actual size of the domain in z is irrelevant, as long as there is enough space for the particles. It is important to have a look at the initial particle file, and check what are the maximum values of the coordinate in the z direction, in order to have an idea of the space needed.

The physical parameters are the following:

- `contactModel=LINEAR` We use a simple linear contact model (see Sec. 5.1.1) model.
- `linearStiff=0.2E-4` This is the contact stiffness in the normal direction $k_{\mathrm{L}}^{\mathrm{n}}$
- `particleDensity=2500.0` The particle mass density ρ_{s}
- `restitution = 0.88` The restitution coefficient ζ.
- `forceZ=-9.806`. This is gravity, acting in the z coordinate.
- `viscTang=0`. The damping ratio in the tangential direction α_{t}
- `frictionCoefPart=0`, `frictionCoefWall=0`, `frictionCoefObj=0` We will deactivate friction in order to obtain a packing as dense as possible.
- `particleFile=./particle_init.dat` The file containing the intial location and velocity of the particles, as well as their radius.
- `numVisc=0.001` This is a stabilization parameter. Choose something just small enough so that it will not affect the results. To get an idea about the value, imagine the particles immersed in a fluid (e.g. air) This is the dynamic visocisty of that fluid.

The code is configured by setting the following options:

- `fluidTimeStep=5.0e-5` Although LBM is running, the code needs a value here, smaller than the output step `screenExpTime`, but larger than the DEM time step. Something of the order of 10^{-4} or 10^{-5} will usually work well.
- `criticalRatio=0.002` This is the ratio between the DEM time step, and the *expected* collision time. Keep small, but remember that the smallest, the longest the simulation. Something of the order of 0.01-0.05 usually works well. This is the standard way in the code to set up the DEM time step. The alternative is to use the `multiStep` option, which is not covered in this tutorial.

Chapter 5

Outline of the DEM-LBM model

5.1 Outline of the DEM model

What follows is adapted from [6]. DEM is a method that has become over the last years the most common choice for the representation of granular systems [7]. Every grain p is represented as a Lagrangian point $\boldsymbol{x}_\mathrm{p}$ and is idealized as a particle with mass m_p and moment of inertia $\boldsymbol{J}_\mathrm{p}$. In a three-dimensional space, the state of a particle is defined by six degrees of freedom, position $\boldsymbol{x}_\mathrm{p}$ and orientation $\boldsymbol{\theta}_\mathrm{p}$. The evolution of these is governed by Newton's equations of motion,

$$\begin{aligned} m_\mathrm{p}\frac{d^2\boldsymbol{x}_\mathrm{p}}{dt^2} &= \boldsymbol{F}_\mathrm{p}, \\ \boldsymbol{J}_\mathrm{p}\frac{d^2\boldsymbol{\theta}_\mathrm{p}}{dt^2} &= \boldsymbol{M}_\mathrm{p} - \frac{d\boldsymbol{\theta}_\mathrm{p}}{dt}\times\left(\boldsymbol{J}_\mathrm{p}\frac{d\boldsymbol{\theta}_\mathrm{p}}{dt}\right), \end{aligned} \tag{5.1}$$

where the force $\boldsymbol{F}_\mathrm{p}$ and the moment $\boldsymbol{M}_\mathrm{p}$ are the results of the interactions acting on the particle. These are the fluid-particle coupling interactions F_hydro and M_hydro coming from LES, the collisions between particles (see Eq. 5.8), and external force fields such as gravity. In general, the interaction resultants are a function not only of position and orientation of the particles, but also of their linear velocity $\boldsymbol{u}_\mathrm{p}$ and angular velocity $\boldsymbol{\omega}_\mathrm{p}$:

$$\begin{aligned} \boldsymbol{F}_\mathrm{p} &= \boldsymbol{F}_\mathrm{p}\left(\boldsymbol{x}_\mathrm{p}, \boldsymbol{u}_\mathrm{p}, \boldsymbol{\theta}_\mathrm{p}, \boldsymbol{\omega}_\mathrm{p}\right), \\ \boldsymbol{M}_\mathrm{p} &= \boldsymbol{M}_\mathrm{p}\left(\boldsymbol{x}_\mathrm{p}, \boldsymbol{u}_\mathrm{p}, \boldsymbol{\theta}_\mathrm{p}, \boldsymbol{\omega}_\mathrm{p}\right). \end{aligned} \tag{5.2}$$

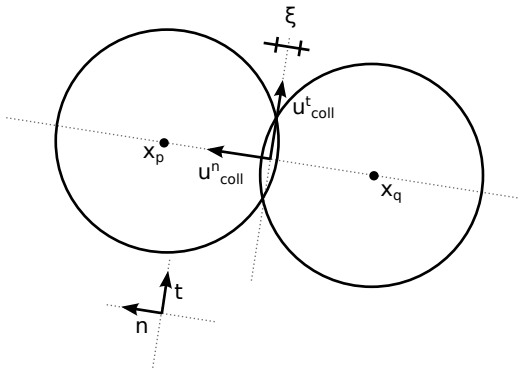

Figure 5.1: Representation of the DEM and of the contact law.

The simplest form of the DEM considers particles to be spheres of radius r_p. In this case, checking if two particles p1 and p2 are in contact is very simple, the overlap between them being

$$\xi = r_\mathrm{p1} + r_\mathrm{p2} - d_\mathrm{p1,p2}, \tag{5.3}$$

where $\boldsymbol{d}_\mathrm{p1,p2} = \boldsymbol{x}_\mathrm{p2} - \boldsymbol{x}_\mathrm{p1}$ is the vector distance between the two centers, which also defines the collision reference system (see Fig. 5.1), whose first component is

$$\boldsymbol{n} = \frac{\boldsymbol{d}_\mathrm{p1,p2}}{d_\mathrm{p1,p2}}. \tag{5.4}$$

The second and third components, $\boldsymbol{t}$ and $\boldsymbol{b}$, are defined according to the component collision velocity vector $\boldsymbol{u}_{\text{coll}}$. The component of $\boldsymbol{u}_{\text{coll}}$ parallel to $\boldsymbol{n}$, the normal collision velocity, can be computed as a function of the translational velocities of the particles $\boldsymbol{u}_{\text{p1}}$ and $\boldsymbol{u}_{\text{p2}}$

$$\boldsymbol{u}^{\text{n}}_{\text{coll}} = ((\boldsymbol{u}_{\text{p2}} - \boldsymbol{u}_{\text{p1}}) \cdot \boldsymbol{n})\boldsymbol{n}, \tag{5.5}$$

while the component aligned to $\boldsymbol{t}$, the tangential velocity, is computed as a function of the rotational velocities of the particles $\boldsymbol{\omega}_{\text{p1}}$ and $\boldsymbol{\omega}_{\text{p2}}$ too, as

$$\boldsymbol{u}^{\text{t}}_{\text{coll}} = \boldsymbol{u}_{\text{p2}} - \boldsymbol{u}_{\text{p1}} - \boldsymbol{u}^{\text{n}}_{\text{coll}} - r_{\text{p1}}\boldsymbol{\omega}_{\text{p1}} \times \boldsymbol{n} - r_{\text{p2}}\boldsymbol{\omega}_{\text{p2}} \times \boldsymbol{n}. \tag{5.6}$$

From the definition of the relative tangential velocity, the remaining two vectors of the collision reference system are defined as

$$\begin{aligned} \boldsymbol{t} &= \boldsymbol{u}^{\text{t}}_{\text{coll}}/u^{\text{t}}_{\text{coll}}, \\ \boldsymbol{b} &= \boldsymbol{n} \times \boldsymbol{t}. \end{aligned} \tag{5.7}$$

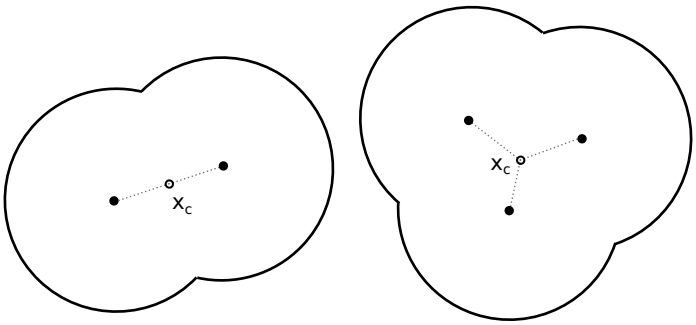

Figure 5.2: A 2-sphere and a 3-sphere cluster.

The overlap represents the elastic deformation of the particles. If ξ is positive, the two particles exchange a repulsive force $\boldsymbol{F}_{\text{coll}}$, whose expression as a function of ξ is defined according to the contact constitutive model. This representation of debris material as spherical particles is motivated by computational convenience, but is only acceptable as a first approximation of their actual behavior. One of the possibility to overcome this simplification is to represent grains as clusters of spheres, rigidly connected (see Fig. 5.2). This preserves the simplicity of the contact search of Eq. 5.3, but allows for a more realistic representation.

The collision force $\boldsymbol{F}_{\text{coll}}$ is computed by calculation of the moduli of its normal component $F^{\text{n}}_{\text{coll}}$ and tangential component $F^{\text{t}}_{\text{coll}}$, which are both dependent on the employed constitutive model. The final system of interactions during a collision is

$$\begin{aligned} \boldsymbol{F}_{\text{coll,p1}} &= -F^{\text{n}}_{\text{coll}}\boldsymbol{n} + F^{\text{t}}_{\text{coll}}\boldsymbol{t}; & \boldsymbol{F}_{\text{coll,p2}} &= F^{\text{n}}_{\text{coll}}\boldsymbol{n} - F^{\text{t}}_{\text{coll}}\boldsymbol{t}; \\ \boldsymbol{M}_{\text{coll,p1}} &= r_{\text{p1}}F^{\text{t}}_{\text{coll}}\boldsymbol{b}; & \boldsymbol{M}_{\text{coll,p2}} &= r_{\text{p2}}F^{\text{t}}_{\text{coll}}\boldsymbol{b}. \end{aligned} \tag{5.8}$$

If the particles are spherical, only the tangential component generates a moment. If the particles are non-spherical, Eq. 5.8 must be adapted to take into account the secondary moment generated by the normal component.

5.1.1 Normal contact

For the normal collision, two models have been implemented: the damped Hertzian model, and the simpler linear-dashpot model. The model choice depends on the desired level of accuracy, and can be controlled by the user with the command `contactModel`

The linear dashpot

The linear dashpot is one of the earliest developed models, and idealizes the contact as a parallel connection with a spring of stiffness k^{n}_{L} and a damper with viscous damping coefficient $\alpha^{\text{n}}_{\text{L}}$. The normal collision force acts in the direction $\boldsymbol{n}$ normal to the contact surface and is expressed as

$$F^{\text{n}}_{\text{coll}} = k^{\text{n}}_{\text{L}}\xi + 2\alpha^{\text{n}}_{\text{L}}\sqrt{k^{\text{n}}_{\text{L}}\tilde{m}}\dot{\xi}, \tag{5.9}$$

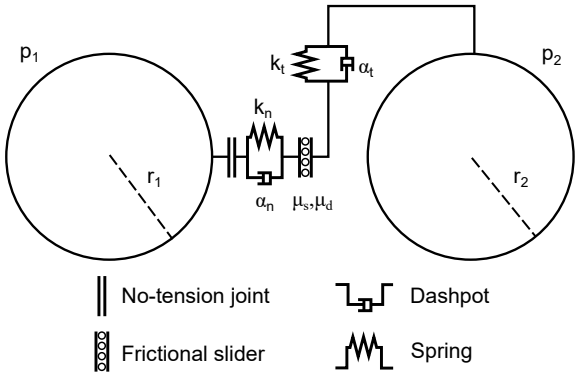

Figure 5.3: Representation of the DEM and of the contact law.

with $\tilde{m}$ being the effective mass defined as $\tilde{m} = m_\mathrm{p} m_\mathrm{q}/(m_\mathrm{p} + m_\mathrm{q})$.

The overlap obeys Newton's law in the form of a harmonic oscillator

$$\ddot{\xi} + 2\gamma_\mathrm{L}\dot{\xi} + \omega_\mathrm{L}^2 \xi = 0, \tag{5.10}$$

where $\gamma_\mathrm{L} = \alpha_\mathrm{L}^\mathrm{n}\sqrt{k_\mathrm{L}^\mathrm{n}/\tilde{m}}$ is the damping ratio, and $\omega_\mathrm{L} = \sqrt{k_\mathrm{L}^\mathrm{n}/\tilde{m}}$ is the frequency of the undamped oscillator. The restitution coefficient ζ, defined as the ratio between the velocity of the particles before and after contact, is constant in this model and is expressed as

$$\zeta = \exp\left(-\frac{\pi\gamma_\mathrm{L}}{\sqrt{\omega_\mathrm{L}^2 - \gamma_\mathrm{L}^2}}\right), \tag{5.11}$$

if the system is not overdamped. This behavior is unphysical, since experiments [8] showed that the coefficient of restitution changes with the impact velocity, albeit not dramatically. The idealization of a constant coefficient of restitution is however desirable from a practical point of view. The coefficient of restitution can be easily estimated with simple experiments, providing access to the determination of $\gamma_\mathrm{L}^\mathrm{n}$ and $\alpha_\mathrm{L}^\mathrm{n}$. The time of contact $t_\mathrm{coll,L}$ is also constant in this model, and is inversely proportional to the oscillator frequency

$$t_\mathrm{coll,L} = \frac{\pi}{\sqrt{\omega_\mathrm{L}^2 - \gamma_\mathrm{L}^2}}. \tag{5.12}$$

In a DEM simulation the time step is chosen so that contacts are represented with an acceptable degree of accuracy. This is usually satisfied by imposing a time step $\Delta t^\mathrm{DEM} < \frac{1}{10} t_\mathrm{coll,L}$. A constant contact time means that the accuracy of the simulation can be easily controlled, which is a favorable property of this model.

The Hertzian model

The linear-dashpot model presented in the last section does not take correctly into account how the deformation ξ is a function of the shape of the particle. If these are spheres, the contact law should be modified according to the theory of Hertz [9], which has the advantage of being expressed in terms of the physical properties of the material, namely Young's modulus E_p and Poisson's ratio ν_p. The normal force reads:

$$F_\mathrm{coll}^\mathrm{n} = k_\mathrm{H}^\mathrm{n}\xi + 2\alpha_\mathrm{H}^\mathrm{n}\sqrt{k_\mathrm{H}^\mathrm{n}\tilde{m}}\frac{d\xi}{dt}, \tag{5.13}$$

where the tangent stiffness k_H^n is now a function of the overlap, yielding a non-linear behavior. Its expression is

$$k_\mathrm{H}^\mathrm{n} = \frac{2}{3}\frac{E_\mathrm{p}}{1-\nu_\mathrm{p}^2}\sqrt{\tilde{r}}\xi^{1/2}, \tag{5.14}$$

where $\tilde{r} = r_\mathrm{p1} r_\mathrm{p2}/(r_\mathrm{p1} + r_\mathrm{p2})$ is the effective radius. Eq. 5.14 slightly differs from the canonical Hertz theory. It is a model developed by Tsuji *et al.* [10], with the purpose of obtaining a pseudo-Hertzian model characterized by a constant coefficient of restitution. While being not completely

sound, this approach is practical since the estimation of the coefficient of restitution ζ is usually an easy task. The coefficient $\alpha_{\mathrm{H}}^{\mathrm{n}}$ can then be determined from ζ as [11]

$$\alpha_{\mathrm{H}}^{\mathrm{n}} = \frac{-\sqrt{5}\ln\zeta}{\sqrt{\ln^2\zeta + \pi^2}}, \tag{5.15}$$

In contrast to the linear dashpot, this model however does not provide a simple way to estimate the time step. The collision time, as a matter of fact, is a function of the normal collision velocity (see Eq. 5.5), as

$$t_{\mathrm{coll,H}} \simeq 1.1 \left(\frac{E_{\mathrm{p}}}{1-\nu_{\mathrm{p}}^2}\rho_{\mathrm{p}} \right)^{2/5} \frac{\tilde{r}}{(u_{\mathrm{coll}}^{\mathrm{n}})^{1/5}}. \tag{5.16}$$

The DEM time step can then be adapted according to the principle $\Delta t^{\mathrm{DEM}} < \frac{1}{10} t_{\mathrm{coll,H}}$ by an evaluation of the maximum particle velocity in the system. Alternatively, the maximum velocity can be predicted in the beginning. This allows the use of a constant time step, a simplification that comes at the expenses of the overall performance of the method.

5.1.2 Tangential contact

`HYBIRD` allows the user to choose between two models: a fully developed frictional model, and a simpler viscous model. The former is necessary when the granular system is in a static or quasi-static state, but is much more computationally demanding. The latter is a simpler model, which can be used in highly dynamic systems as a first approximation. The command `staticFrictionSolver` allows the user to control which model to use.

The viscous model

This is the simplaset model, which is triggered if the command `staticFrictionSolver` is deactivated. The tangential contact force is in this case assumed to be proportional to the component of the relative velocity of the two spheres lying on the contact surface (see Eq. 5.6) as

$$F_{\mathrm{coll}}^{\mathrm{t}} = -\min\left(2\alpha_{\mathrm{t}}\sqrt{k^{\mathrm{t}}\tilde{m}}u_{\mathrm{coll}}^{\mathrm{t}}, \tan(\phi_d)F_{\mathrm{coll}}^{\mathrm{n}} \right), \tag{5.17}$$

where k_{t} is a tangential stiffness, whose definition depends on the normal contact model:

$$k^{\mathrm{t}} = \begin{cases} k_{\mathrm{L}}^{\mathrm{n}} & \text{if the normal model is linear,} \\ \frac{2E_{\mathrm{p}}}{(2-\nu_{\mathrm{p}})(1+\nu_{\mathrm{p}})}\sqrt{\tilde{r}}\left(F_{\mathrm{coll}}^{\mathrm{n}}\right)^{-1/3} & \text{if the normal model is Hertzian.} \end{cases} \tag{5.18}$$

This model applies a dissipative force proportional to the shear between the two particles, an approach that mimics the viscous behavior of a fluid. The maximum force is given by a Coulomb friction criterion. This model simplifies the frictional behavior by implementing only the dynamic component of friction, and thus only the dynamic friction coefficient $\tan(\phi_d)$ appears. It was originally developed by Haff and Werner [12] and has been successfully used in many situations [13]

5.2 Outline of the LBM model

In more traditional CFD solvers like the Finite Element Method or the Finite Volume Method, the conservation of mass and momentum is imposed directly on macroscopic quantities, i.e. velocity and pressure, through a solution of the Navier-Stokes equations. In LBM, the fluid is represented as a collection of particles moving in space, conceptually similar to the real composition of a fluid as a molecular system. Like in a real fluid, mass and momentum conservation are applied at the microscopic level whenever a collision occurs. However, the number of molecules present in any practically relevant volume of fluid is so large, that there is no hope to simulate it using computers. LBM solves this problem by drastically reducing the number of degrees of freedom, describing time, space, and velocity in a discrete form. While time and space are discrete also in traditional CFD solvers, the discretization of the velocity space is peculiar to LBM, and is inherited by Lattice Automata, from which LBM stems [14].

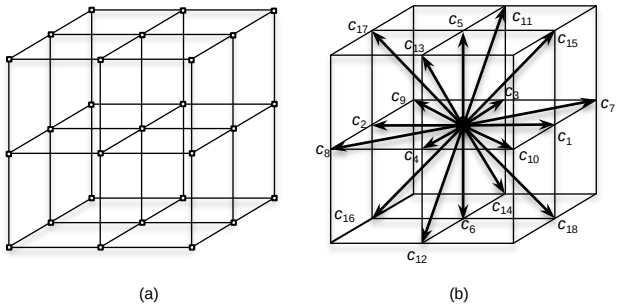

Figure 5.4: The LBM cubic lattice (a), and the set of discrete velocity used in the D3Q19 (b).

The theoretical foundations of LBM lies within the kinetic theory of gases. Imagine a system of gas particles, following a chaotic motion typical of particles. This system can be described by employing a probability density function, $f(\boldsymbol{x}, t, \boldsymbol{c})$, which represent the probability of finding a particle in the position $\boldsymbol{x}$ at time t, moving with speed $\boldsymbol{c}$. As mentioned before, the overwhelming amount of degrees of freedom in this description is drastically reduced by employing a discretized time-space. In the LBM formulations presented here, a cubic lattice is used, as represented in Fig. 5.4 (a). At every time step, the particles located in a node $\boldsymbol{x}_0$ are allowed to move to a neighbor node $\boldsymbol{x}_i$. Since the time step Δt is fixed, every movement corresponds to a specific velocity, calculated as

$$\boldsymbol{c}_i = \frac{\boldsymbol{x}_i - \boldsymbol{x}_0}{\Delta t}. \tag{5.19}$$

LBM limits the eligible neighbors to a fixed set. The more neighbors are allowed, the higher the resolution power of LBM becomes, together with an increase in computational time, iterations, and complexity of the code. For the simulation of fluid dynamics, a very common compromise consist in choosing the 19 neighbors depicted in Fig. 5.4 (b), which correspond to the set of velocities:

$$\boldsymbol{c}_i = \frac{\Delta S}{\Delta t} \cdot \begin{cases} (0,0,0) & \text{for } i = 0, \\ (\pm 1,0,0) & \text{for } i = 1,2, \\ (0,\pm 1,0) & \text{for } i = 3,4, \\ (0,0,\pm 1) & \text{for } i = 5,6, \\ (\pm 1,\pm 1,0) & \text{for } i = 7..10, \\ (0,\pm 1,\pm 1) & \text{for } i = 11..14, \\ (\pm 1,0,\pm 1) & \text{for } i = 15..18. \end{cases} \tag{5.20}$$

A lattice defined in this way is commonly named D3Q19 (3 dimensions, 19 velocity vectors). At every node $\boldsymbol{x}$ and time t, 19 probabilities distribution functions f_i are defined, each corresponding to one of the discrete velocities: $f_i(\boldsymbol{x}, t) = f(\boldsymbol{x}, t, \boldsymbol{c}_i)$. Each of these functions can be imagined as a population of particles streaming with the same velocity and in the same direction. In the following, time and space discretizations are considered unitary, i.e. $\Delta S = 1$ and $\Delta t = 1$, which greatly simplifies the notation, since the lattice speed $c = \Delta S/\Delta t$ is also unit. The reference density ρ_{ref} is also unitary, yielding a system of completely dimensionless variables. In an actual simulation, the equations are indeed solved with unit discretizations, and the results are scaled during post-processing. In the following, we will express all units in dimensionless forms, and a few notes on scaling will be given later on.

Once the lattice is defined, the macroscopic variables density ρ_{f} and velocity $\boldsymbol{u}_{\text{f}}$ are reconstructed at every node by simple summation:

$$\rho_{\text{f}}(\boldsymbol{x}, t) = \sum_i f_i(\boldsymbol{x}, t), \quad \boldsymbol{u}_{\text{f}}(\boldsymbol{x}, t) = \left(\sum_i f_i(\boldsymbol{x}, t) \boldsymbol{c}_i \right) / \rho_{\text{f}}(\boldsymbol{x}, t). \tag{5.21}$$

The density of the fluid is therefore treated as a variable, and the medium is granted a limited compressibility. The pressure can be reconstructed as

$$p_{\text{f}}(\boldsymbol{x}, t) = c_{\text{s}}^2 \cdot \rho_{\text{f}}(\boldsymbol{x}, t), \tag{5.22}$$

where c_{s} is the norm of the lattice speed of sound, defined as

$$c_{\text{s}} = \frac{c}{\sqrt{3}} = \frac{1}{\sqrt{3}}. \tag{5.23}$$

At every time step, the populations converging to the same node are colliding at that position. The dynamics of the collisions is governed by the Boltzmann equation, another result from the kinetic theory. The evolution equation, in absence of body forces, reads

$$\frac{\partial f}{\partial t} + \boldsymbol{c}\nabla f = \Omega_{\text{coll}}, \tag{5.24}$$

where the term Ω_{coll} implements the effect of collisions. In the lattice, Eq. 5.24 appears in a simple form

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i, t+1) = f_i(\boldsymbol{x}, t) + \Omega_{\text{coll},i}(\boldsymbol{x}, t). \tag{5.25}$$

The definition of Ω_{coll} is a crucial point in the development of a LBM formulation. Its precise expression, as given by Boltzmann, is of no practical use in this case. Fortunately, Bhatnagar, Gross, and Krook [15] in 1954 proposed an alternative approximate form of Ω_{coll} as an operator simply driving the system towards thermodynamics equilibrium. Its expression reads

$$\Omega_{\text{coll},i} = \frac{f_i^{\text{eq}} - f_i}{\tau}, \tag{5.26}$$

where τ, is a time constant that governs how hard the system is pushed towards equilibrium. The expression of f^{eq} follows a Maxwell-Boltzmann equilibrium distribution.

$$f^{\text{eq}} = \frac{\rho_{\text{f}}}{(2\pi RT)^{3/2}} e^{\frac{-(\boldsymbol{c}-\boldsymbol{u}_{\text{f}})^2}{2RT}} \tag{5.27}$$

with R denoting the universal gas constant and T the temperature. The discretized form of Eq. 5.27 is obtained after a Taylor expansion on $\boldsymbol{u}_{\text{f}}$ up to the second order. Considering that $c = \sqrt{3RT}$, one obtains

$$f_i^{\text{eq}}(\boldsymbol{u}_{\text{f}}, \rho_{\text{f}}) = \rho_{\text{f}} w_i \left(1 + 3\boldsymbol{c}_i \cdot \boldsymbol{u}_{\text{f}} + \frac{9}{2}\left(\boldsymbol{c}_i \cdot \boldsymbol{u}_{\text{f}}\right)^2 - \frac{3}{2}\boldsymbol{u}_{\text{f}} \cdot \boldsymbol{u}_{\text{f}}\right). \tag{5.28}$$

The set of weights w_i is chosen so that the collision operator conserves mass and momentum. The values for the D3Q19 are

$$w_i = \begin{cases} 1/3 & \text{for } i = 1 \\ 1/18 & \text{for } i = 2..7 \\ 1/36 & \text{for } i = 8..19. \end{cases} \tag{5.29}$$

With the LBM presented so far, the dynamics of a Newtonian fluid can be solved. A realization of LBM can be proven to be analogous to the solution of the Navier-Stokes equation if the Knudsen number is small, i.e. provided that the size of an element is much smaller than the length scale of the system [16]. Stability and accuracy are however limited to the range of small Mach numbers, i.e. in the simulation the condition $\text{Ma} = u_{\text{f}}/c_s \ll 1$ should always be respected. The viscosity of the fluid is directly related to the relaxation time τ, according to

$$\nu(\boldsymbol{x}, t) = \frac{\tau(\boldsymbol{x}, t) - 1/2}{3}. \tag{5.30}$$

If a volumetric force is acting on the system (e.g. gravity), an additional operator Ω_{force} needs to be added to the system. There exist different techniques for the implementation of such an operator. Commonly employed is the procedure suggested in Ref. [17], which, given a uniform force field F, consists of the addition of the following operator to the right hand of Eq. 5.25

$$\Omega_{\text{force},i} = w_i \left(1 - \frac{1}{2\tau}\right) \left[3\left(\boldsymbol{c}_i - \boldsymbol{u}_{\text{f}}\right) + \boldsymbol{c}_i\left(\boldsymbol{c}_i \cdot \boldsymbol{u}_{\text{f}}\right)\right] \boldsymbol{F}. \tag{5.31}$$

The reconstruction of the macroscopic velocity also needs to be modified, and Eq. 5.21 becomes

$$\rho_{\text{f}}(\boldsymbol{x}, t) = \sum_i f_i(\boldsymbol{x}, t), \quad \boldsymbol{u}_{\text{f}}(\boldsymbol{x}, t) = \left(\sum_i f_i(\boldsymbol{x}, t)\boldsymbol{c}_i + \boldsymbol{F}/2\right) / \rho_{\text{f}}(\boldsymbol{x}, t). \tag{5.32}$$

This basic formulation needs to be extended with a few more terms in order to model debris flow. These represent the effects from the non-Newtonian rheology, the free surface and the interaction with particles, and are addressed in this order in the following.

5.2.1 Non-Newtonian fluids

In LBM, a non-Newtonian fluid requires a formulation with a variable viscosity. The code uses a simplified approach to this problem, adapting Eq. 5.30 so that it can take into account the spatial variability of ν. Before doing that, let us express Eq. 5.30 in dimensional terms, i.e. using the spacial discretization ΔS, the time step Δt and the reference density ρ_{ref}:

$$\nu(\boldsymbol{x}, t) = \frac{\tau(\boldsymbol{x}, t) - 1/2}{3} \frac{\rho_{\text{ref}} \Delta S^2}{\Delta t}. \tag{5.33}$$

Inverting Eq. 5.33, one obtains the constitutive law for the relaxation time:

$$\tau(\boldsymbol{x}, t) = \frac{1}{2} + 3\nu(\boldsymbol{x}, t) \frac{\Delta t}{\rho_{\text{ref}} \Delta S^2}. \tag{5.34}$$

For many non-Newtonian rheologies, the viscosity is defined as a function of the shear rate $\dot{\gamma}$. Unlike in traditional CFD solvers, where the computation of $\dot{\gamma}$ needs the reconstruction of spatial derivatives, in LBM it can be computed locally from the distribution function as

$$\dot{\gamma}_{ab}(\boldsymbol{x}, t) = \frac{3}{2\tau(\boldsymbol{x}, t)} \sum_i \boldsymbol{c}_{i,a} \boldsymbol{c}_{i,b} \left(f_i(\boldsymbol{x}, t) - f_i^{\text{eq}}(\boldsymbol{x}, t) \right), \tag{5.35}$$

where the Einstein notation convention has been used for the indices a and b. The capability of LBM to compute the shear rate locally gives a great computational advantage, again simplifying its implementation in parallel architectures.

Many non-Newtonian models, however, require the viscosity to vary within very large ranges, often diverging to $+\infty$ or vanishing to zero. In LBM term, this means that the relaxation time would need also to vary within the range $[0.5, +\infty]$. This however causes stability issues, because the relaxation time should not be larger than 1, and always strictly larger than 0.5. In the literature, various alternative solution to this problem are available [18, 19, 20, 21, 22]. The simplest solution, and the easier to implement, is to limit τ between two values $\tau_{\min}$ and $\tau_{\max}$ [23]. The limit values are determined empirically, and are taken to be $\tau_{\min} = 0.5005$ and $\tau_{\max} = 1.0$. Using Eq. 5.33, we can also determine the related minimum and maximum values of viscosity:

$$\mu_{\min} = \frac{\tau_{\min} - 1/2}{3} \frac{\rho_{\text{ref}} \Delta S^2}{\Delta t}. \tag{5.36}$$

$$\mu_{\max} = \frac{\tau_{\max} - 1/2}{3} \frac{\rho_{\text{ref}} \Delta S^2}{\Delta t}. \tag{5.37}$$

Note that, while the values of $\tau_{\min}$ and $\tau_{\max}$ are fixed (or, at least, have limited variability), the values of $\mu_{\min}$ and $\mu_{\max}$ can be controlled by the user by changing the time and space discretizations ΔS and Δt. A discussion about this and its effect on the accuracy of the simulations is given in Sec. 4.2.

5.2.2 The mass-tracking algorithm for the free-surface resolution

Debris flows are free-surface flows, a feature that needs a treatment in the LBM. Strictly speaking, this would require the solution of two fluid phases, the liquid and the gas. However, the motion of the gas is of no practical interest, and its influence on the motion of the liquid is minimal. Because of this, Körner *et al.* [24] created a multiphase version of LBM where the dynamics of the gas phase is neglected. Their method, called mass-tracking algorithm, is employed here because of its accuracy and its concise formulation. More applications of the same method can be found in Refs. [25, 26].

The lattice nodes are divided into three categories: liquid, gas, and interface. An illustration of the categories is provided in Fig. 5.5. The state of a node is determined by an additional variable, the liquid fraction λ:

$$\begin{cases} \lambda = 1 & \text{if the node is liquid,} \\ 0 < \lambda < 1 & \text{if the node is interface,} \\ \lambda = 0 & \text{if the node is gas.} \end{cases} \tag{5.38}$$

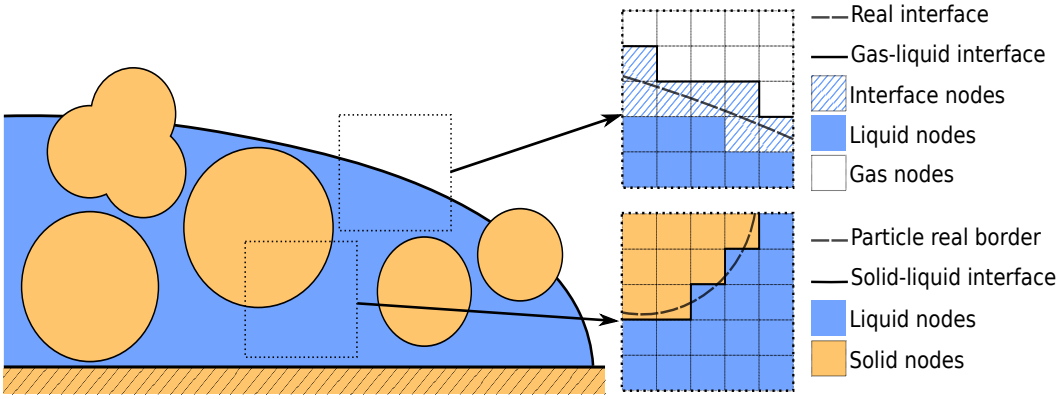

Figure 5.5: The division of the lattice nodes into categories, according to their location in the flow.

The liquid mass contained in a node can be obtained by multiplying the liquid fraction to the fluid density, i.e. $m_\mathrm{f} = \lambda \rho_\mathrm{f}$, and is updated by tracking the populations streaming in and out of every node using the equation

$$m_\mathrm{f}(\boldsymbol{x}, t+1) = m_\mathrm{f}(\boldsymbol{x}, t) + \sum_i \alpha_i \left[f_{i'}(\boldsymbol{x} + \boldsymbol{c}_i, t) - f_i(\boldsymbol{x}, t) \right], \tag{5.39}$$

where i' is the direction opposite to i (see Fig. 5.4). The conservation of mass is guaranteed by the parameter α_i, which is a function of the liquid fraction of the neighbor nodes (located at $\boldsymbol{x} + \boldsymbol{c}_i$) as

$$\alpha_i = \begin{cases} \frac{1}{2}\left[\lambda(\boldsymbol{x}, t) + \lambda(\boldsymbol{x} + \boldsymbol{c}_i, t)\right] & \text{if the neighbor node is interface,} \\ 1 & \text{if the neighbor node is liquid,} \\ 0 & \text{if the neighbor node is gas.} \end{cases} \tag{5.40}$$

An interface node will evolve into a gas node if its mass falls below zero, and into a liquid node if it rises above unit. Whenever an event of this sort happens, the neighbor nodes also evolve, so that liquid and gas nodes are always separated by interface nodes. While the theoretical algorithm conserves mass exactly, the discrete integration and the evolution of the nodes result in small losses and gains of mass. The surplus (or shortfall) of mass is then computed at every time step and is corrected by distributing an opposite quantity of mass among all interface nodes.

The LBM equations are solved only for liquid and interface nodes. Gas nodes are supposed to have no mass, and are therefore neglected. This creates a problem since the populations streaming to the interface nodes from beyond the interface, necessary for Eq. 5.39, are not defined. The missing distributions are computed by assuming that the gas node has the same velocity of the interface node $\boldsymbol{u}_\mathrm{int}$ and a constant atmospheric density ρ_atm, obtaining

$$f_{i'}(\boldsymbol{x} + \boldsymbol{c}_{i'}, t+1) = f_i^{eq}(\boldsymbol{u}_\mathrm{int}, \rho_\mathrm{atm}) + f_{i'}^{eq}(\boldsymbol{u}_\mathrm{int}, \rho_\mathrm{atm}) - f_i(\boldsymbol{x}, t). \tag{5.41}$$

This is analogous to applying a fixed-pressure boundary condition at the interface, and local symmetry conditions for the velocity. In the simulations, $\rho_\mathrm{atm} = 1$ in lattice units, implying that $\rho_\mathrm{f} > 1$ everywhere in the liquid.

5.2.3 Solid boundaries and the fluid-particle coupling

The particles constituting the granular phase are solved using the DEM (see Sec. 5.1). However, the computation of the hydrodynamic interaction forces is performed within the LBM framework. The traditional way to handle the fluid-particle coupling is by considering the surface of the particles as a moving no-slip boundary condition. This method is based on the bounce back mechanism and is reviewed in Sec. 5.2.3. However, this method is unstable if the particles are moving quickly inside the lattice, a fact that motivated the development of alternative methods. Among the many available possibilities, the Immersed Boundary Method with Direct Forcing seems to be the most suitable for debris flow simulations, because of its efficiency and stability.

The discretization of particles in the lattice, regardless of the coupling method, follows the same logic used for the resolution of the free surface. Nodes are divided into two categories: solid and

fluid. Fluid nodes can be either of the three categories already described (i.e. liquid, interface, or gas), while solid nodes are all the nodes that are located inside the volume of a grain. The main difference between the "dry" and the "wet" coupling scheme is in the treatment of the solid nodes. The dry scheme works with fluid only in the exterior of the particles, and therefore the solid nodes are inactive and do not contain fluid. In the wet coupling the fluid is also present inside the solid nodes.

The bounce-back condition and the dry coupling scheme

The dry coupling is based on the no-slip boundary condition, which is commonly implemented in LBM through the bounce-back rule. This states that whenever a population is streaming towards a wall, this population is reflected and bounced back in the opposite direction. In LBM notation

$$f_{i'}(\boldsymbol{x}, t+1) = f_i(\boldsymbol{x}, t), \tag{5.42}$$

where $\boldsymbol{x}$ is the position of the fluid node close to the wall, i the streaming direction that points at the wall, and i' the opposite direction. If the wall is moving, the reflection has to take into account the momentum transfer between population and wall. The correction is implemented by imposing

$$f_{i'}(\boldsymbol{x}, t+1) = f_i(\boldsymbol{x}, t) - 6 w_i \rho_\mathrm{f} \boldsymbol{u}_\mathrm{w} \cdot \boldsymbol{c}_i, \tag{5.43}$$

where $\boldsymbol{u}_\mathrm{w}$ is the velocity of the wall at the bounce-back location. To achieve second-order accuracy, the bounce-back location should be located exactly halfway between the solid node and fluid node. In a debris-flow simulation, this rule is applied whenever the fluid is at contact with a wall, e.g. the channel bed. The same rule can also be applied to grains [27, 28], with the additional complexity of calculating the velocity of the grain surface at every time step. The grains never undergo large deformations, and can therefore be safely assumed to be rigid bodies for what concerns the determination of $\boldsymbol{u}_\mathrm{w}$. If $\boldsymbol{u}_\mathrm{p}$ and $\boldsymbol{\omega}_\mathrm{p}$ are the translational and rotational velocities of the particle, respectively, the velocity of the particle at the bounce back-location can be determined as

$$\boldsymbol{u}_\mathrm{w} = \boldsymbol{u}_\mathrm{p} + \boldsymbol{r}_\mathrm{BB} \times \boldsymbol{\omega}_\mathrm{p}, \tag{5.44}$$

where $\boldsymbol{r}_\mathrm{BB}$ is the vector connecting the centre of the grain to the bounce-back location.

The momentum exchange described by Eq. 5.43 results in a force applied to the wall, in the i direction. The resultant of all forces given by momentum exchange on a wall can be written as

$$\boldsymbol{F}_\mathrm{hydro} = \sum \left(2 f_i(\boldsymbol{x}, t) - 6 w_i \rho_f \boldsymbol{u}_w \cdot \boldsymbol{c}_i\right) \boldsymbol{c}_i, \tag{5.45}$$

and the resultant moment

$$\boldsymbol{M}_\mathrm{hydro} = \sum \left(2 f_i(\boldsymbol{x}, t) - 6 w_i \rho_f \boldsymbol{u}_w \cdot \boldsymbol{c}_i\right) \boldsymbol{c}_i \times \boldsymbol{r}_\mathrm{BB}. \tag{5.46}$$

This implementation was initially suggested by Aidun and Lu [29], who published a scheme that improved the previous method by Ladd [27] removing the requirement that particles are filled with fluid. This method converges at the limit of $\Delta x \to 0$, since the zig-zag description of the particle surface becomes closer to the exact shape for finer lattice resolutions. This is not an important problem in a debris-flow simulation because the force on the grains does not need to be computed exactly. In fact, the spherical shape of the particles is anyway only a crude approximation of the actual shape of the grains, which is irregular.

The motion of the particles in the lattice requires a continuous update of the state of the nodes. A moving particle creates new solid nodes at its front and new fluid nodes at its rear. This is problematic for two reasons. The newly created fluid nodes need to be initialized, and their velocity, density, and liquid fraction are in general unknown. The easiest way to deal with this is by computing the new variables using their averaged value over the surrounding liquid and interface nodes. Another problem is the appearance and disappearance of mass whenever a fluid node is created or destroyed. While the total mass is likely to only slightly oscillate around its initial value, the transfer of mass between wake and front can strongly affects the dynamics of the flow. These problems motivated the search for more precise coupling method, and many alternatives have been developed in the last two decades. Particularly successful are the application that use the Immersed Boundary Method [30, 31, 32, 33, 34] or the fictitious domain technique [35, 36]. The drawback of

these methods is that, the higher their precision, the more complex and computationally expensive their implementation becomes. For this reason, the next section presents a technique, the simplified direct forcing, which is a compromise between precision and speed. Note that even though the dry scheme described here is unsuitable for the coupling with the particle, it is still used for any other stationary object, like the bottom of the flow and in general any other stationary wall.

The wet coupling scheme: the direct forcing

The Immersed Boundary Method (IBM) and similar schemes have enjoyed lately increasing popularity because they allow a more precise resolution of the particle shapes without a small lattice spacing. The method proposed by Feng and Michaelides [37, 38, 39, 40] uses an additional discretization of the particles into a set of segments, each characterized by its center of mass $\boldsymbol{x}_{\text{seg}}$, velocity $\boldsymbol{u}_{\text{seg}}$ and volume V_{seg}. For every segment, the velocity of the fluid calculated at the position $\boldsymbol{x}_{\text{seg}}$ is computed with an interpolating function $\mathcal{F}$ running over every lattice node n in the neighborhood

$$\boldsymbol{u}_{\text{f}}(\boldsymbol{x}_{\text{seg}}, t) = \sum_{n} \mathcal{F}(\boldsymbol{u}_{\text{f},n}(\boldsymbol{x}, t)). \tag{5.47}$$

This velocity is used to compute a force $\boldsymbol{F}_{\text{hydro,seg}}$ for every segment as

$$\boldsymbol{F}_{\text{hydro,seg}}(\boldsymbol{x}_{\text{seg}}, t) = \rho_{\text{f}}(\boldsymbol{x}, t) \left[\boldsymbol{u}_{\text{f}}(\boldsymbol{x}_{\text{seg}}, t) - \boldsymbol{u}_{\text{seg}}(\boldsymbol{x}_{\text{seg}}, t)\right] V_{\text{seg}}. \tag{5.48}$$

This force is transmitted to the DEM for solution of the particle dynamics. Its counterpart for the LBM is a volumetric force $\boldsymbol{e}(\boldsymbol{x}, t)$, whose determination is carried on using the same interpolation function $\mathcal{F}$, this time running over every m segment $\boldsymbol{x}_{\text{seg},m}$ in the proximity of a lattice node locations $\boldsymbol{x}$:

$$\boldsymbol{e}(\boldsymbol{x}, t) = -\sum_{m} \mathcal{F}(\boldsymbol{F}_{\text{hydro,seg}}(\boldsymbol{x}_{\text{seg},m}, t)). \tag{5.49}$$

This force $\boldsymbol{e}(\boldsymbol{x}, t)$ is then applied to the fluid together with the other volumetric forces using Eq. 5.31. Note that this approach requires the interior of the particle to be filled with fluid, whose velocity quickly relaxes to the velocity of the particle. To fasten the relaxation, the interior of the particle can be assigned a higher viscosity (e.g. $\mu_{\max}$).

The direct forcing described so far has the great advantage of allowing the resolution of particles of arbitrary shapes, and has been used with success for the simulation of fiber-reinforced fresh concrete [23, 41]. For the simulations of debris flows, where particles are mostly rounded, the additional complexity given by the generation of the segments might be unjustified. For this reason, a simplified version is used, where the segments correspond to the cubic cells of the lattice, and their center of mass to the lattice nodes $\boldsymbol{x}_{\text{cell}} = \boldsymbol{x}$. This results in a simpler and faster algorithm, at the price of a less precise resolution of the particle shape, and a less smooth evolution in time of the force. No interpolation is required, the volume of a cubic cell is unitary, and the force can be computed directly for the particle as

$$\boldsymbol{F}_{\text{hydro,cell}}(\boldsymbol{x}, t) = \rho_{\text{f}}(\boldsymbol{x}, t) \left[\boldsymbol{u}_{\text{f}}(\boldsymbol{x}, t) - \boldsymbol{u}_{\text{p}}(\boldsymbol{x}, t)\right], \tag{5.50}$$

and for the fluid as

$$\boldsymbol{e}(\boldsymbol{x}, t) = -\boldsymbol{F}_{\text{hydro,cell}}(\boldsymbol{x}, t). \tag{5.51}$$

This approach works well if the particle is large enough to cover a significant number of lattice nodes. The resultant force on the particle is then computed by simple summation over each of the l covered nodes

$$\boldsymbol{F}_{\text{hydro}} = \sum_{l} \boldsymbol{F}_{\text{hydro},l}, \tag{5.52}$$

together with the resultant moment

$$\boldsymbol{M}_{\text{hydro}} = \sum_{l} \boldsymbol{F}_{\text{hydro},l} \times \boldsymbol{r}_{\text{DF},l}, \tag{5.53}$$

where $\boldsymbol{r}_{\text{DF},l}$ is the vector connecting the center of mass of the particle and the node location $\boldsymbol{x}_l$.

Bibliography

[1] A. Monitzer. Combining lattice Boltzmann and discrete element methods on a graphics processor. *International Journal of High Performance Computing Applications*, 26(3):215–226, 4 2012.

[2] Timm Krüger, Halim Kusumaatmaja, Alexandr Kuzmin, Orest Shardt, Goncalo Silva, and Erlend Magnus Viggen. *The Lattice Boltzmann Method. Principles and Practice*. Springer International Publishing, 2017.

[3] A.A. Mohamad. *Lattice Boltzmann Method: Fundamentals and Engineering Applications with Computer Codes*. 2011.

[4] P Jop, Y Forterre, and Ol Pouliquen. A constitutive law for dense granular flows. *Nature*, 441(7094):727–30, 6 2006.

[5] P.-Y. Lagrée, L. Staron, and S. Popinet. The granular column collapse as a continuum: validity of a two-dimensional Navier–Stokes model with a mu(I)-rheology. *Journal of Fluid Mechanics*, 686:378–408, 9 2011.

[6] Alessandro Leonardi. Numerical simulation of debris flow and interaction between flow and obstacle via DEM, 2015.

[7] Nenad Bicanic. Discrete Element Methods. In Erwin Stein, René de Borst, and Thomas J R Hughes, editors, *Encyclopedia of Computational Mechanics*, pages 311–337. John Wiley & Sons, Ltd, 2007.

[8] Goro Kuwabara and Kimitoshi Kono. Restitution Coefficient in a Collision between Two Spheres. *Japanese Journal of Applied Physics*, 26(8):1230–1233, 1987.

[9] H Hertz. *Die Prinzipien der Mechanik*. Barth, 1894.

[10] Y Tsuji, T Tanaka, and T Ishida. Lagrangian numerical simulation of plug flow of cohesionless particles in a horizontal pipe. *Powder Technology*, 71:239–250, 1992.

[11] D. Antypov and J. A. Elliott. On an analytical solution for the damped Hertzian spring. *EPL (Europhysics Letters)*, 94(5):50004, 6 2011.

[12] P.K. Haff and B.T. Werner. Computer simulation of the mechanical sorting of grains. *Powder Technology*, 48(3):239–245, 11 1986.

[13] Thorsten Pöschel and Thomas Schwager. *Computational granular dynamics: Models and algorithms*. Springer, 2005.

[14] Guy R. McNamara and Gianluigi Zanetti. Use of the Boltzmann Equation to Simulate Lattice-Gas Automata. *Physical Review Letters*, 61(20):2332–2335, 1988.

[15] P. L. Bhatnagar, E. P. Gross, and M. Krook. A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems. *Physical Review*, 94(3):511–525, 1954.

[16] S Chapman and T G Cowling. *The mathematical theory of non-uniform gases: an account of the kinetic theory of viscosity, thermal conduction and diffusion in gases*. Cambridge University Press, 1991.

[17] Zhaoli Guo, Chuguang Zheng, and Baochang Shi. Discrete lattice effects on the forcing term in the lattice Boltzmann method. *Physical Review E*, 65:046308, 2002.

[18] Tasos C. Papanastasiou. Flows of Materials with Yield. *Journal of Rheology*, 31(5):385, 1987.

[19] I Ginzburg and K Steiner. Free surface Lattice-Boltzmann method to model the filling of expanding cavities by Bingham Fluids. *Philosophical Transactions of the Royal Society*, 360(1792):453–466, 2001.

[20] Chen-Hao Wang and Jeng-Rong Ho. Lattice Boltzmann modeling of Bingham plastics. *Physica A*, 387(19-20):4740–4748, 8 2008.

[21] A. Vikhansky. Lattice-Boltzmann method for yield-stress liquids. *Journal of Non-Newtonian Fluid Mechanics*, 155(3):95–100, 12 2008.

[22] G.H. Tang, S.B. Wang, P.X. Ye, and W.Q. Tao. Bingham fluid simulation with the incompressible lattice Boltzmann model. *Journal of Non-Newtonian Fluid Mechanics*, 166(1-2):145–151, 1 2011.

[23] Oldřich Švec, Jan Skoček, Henrik Stang, Mette R. Geiker, and Nicolas Roussel. Free surface flow of a suspension of rigid particles in a non-Newtonian fluid: A lattice Boltzmann approach. *Journal of Non-Newtonian Fluid Mechanics*, 179-180:32–42, 2012.

[24] C. Körner, M. Thies, T. Hofmann, N. Thürey, and U. Rüde. Lattice Boltzmann Model for Free Surface Flow for Modeling Foaming. *Journal of Statistical Physics*, 121(1-2):179–196, 10 2005.

[25] Shiyi Chen and Gary D Doolen. Lattice Boltzmann Method for Fluid Flows. *Annual Review of Fluid Mechanics*, 30:329–364, 1998.

[26] M Mendoza, F K Wittel, and H J Herrmann. Simulation of flow of mixtures through anisotropic porous media using a lattice Boltzmann model. *The European physical journal. E, Soft matter*, 32(4):339–48, 8 2010.

[27] Anthony J. C. Ladd. Numerical simulations of particulate suspensions via a discretized Boltzmann equation. Part 1. Theoretical foundation. *Journal of Fluid Mechanics*, 271:285–309, 1994.

[28] Anthony J. C. Ladd. Numerical simulations of particulate suspensions via a discretized Boltzmann equation. Part 2. Numerical results. *Journal of Fluid Mechanics*, 271:311–339, 1994.

[29] Cyrus K. Aidun and Yannan Lu. Lattice Boltzmann simulation of solid particles suspended in fluid. *Journal of Statistical Physics*, 81(1-2):49–61, 1995.

[30] Y T Feng, K Han, and D R J Owen. Coupled lattice Boltzmann method and discrete element modelling of particle transport in turbulent fluid flows: Computational issues. *International Journal for Numerical Methods in Engineering*, 72(1):1111–1134, 2007.

[31] C R Leonardi, D R J Owen, and Y T Feng. Numerical rheometry of bulk materials using a power law fluid and the lattice Boltzmann method. *Journal of Non-Newtonian Fluid Mechanics*, 166(12-13):628–638, 7 2011.

[32] C R Leonardi, D R J Owen, and Y T Feng. Simulation of fines migration using a non-Newtonian lattice Boltzmann-discrete element model Part I : 2D implementation aspects. *Engineering Computations*, 29(4):366–391, 2012.

[33] D R J Owen, C R Leonardi, and Y T Feng. An efficient framework for fluid – structure interaction using the lattice Boltzmann method and immersed moving boundaries. *International Journal for Numerical Methods in Engineering*, 87:66–95, 2011.

[34] C R Leonardi, D R J Owen, and Y T Feng. Simulation of fines migration using a non-Newtonian lattice Boltzmann-discrete element model Part II : 3D extension and applications. *Engineering Computations*, 29(4):392–418, 2012.

[35] R Glowinski, T Pan, T I Hesla, and D D Joseph. A distributed Lagrange multiplier/fictitious domain method for particulate flows. *International Journal of Multiphase Flow*, 25:755–794, 1999.

[36] R. Glowinski, T.W. Pan, T.I. Hesla, D.D. Joseph, and J. Périaux. A Fictitious Domain Approach to the Direct Numerical Simulation of Incompressible Viscous Flow past Moving Rigid Bodies: Application to Particulate Flow. *Journal of Computational Physics*, 169(2):363–426, 5 2001.

[37] Zhi-Gang Feng and Efstathios E. Michaelides. Interparticle forces and lift on a particle attached to a solid boundary in suspension flow. *Physics of Fluids*, 14(1):49, 2002.

[38] Zhi-Gang Feng and Efstathios E. Michaelides. Hydrodynamic force on spheres in cylindrical and prismatic enclosures. *International Journal of Multiphase Flow*, 28(3):479–496, 3 2002.

[39] Zhi-Gang Feng and Efstathios E Michaelides. The immersed boundary-lattice Boltzmann method for solving fluid-particles interaction problems. *Journal of Computational Physics*, 195(2):602–628, 4 2004.

[40] Zhi-Gang Feng and Efstathios E. Michaelides. Proteus: a direct forcing method in the simulations of particulate flows. *Journal of Computational Physics*, 202(1):20–51, 1 2005.

[41] Oldřich Švec, Henrik Stang, John Forbes Olesen, and Lars Nyholm Thrane. Application of the fluid dynamics model to the field of fibre reinforced self-compacting concrete. In P. Rossi and J.L. Tailhan, editors, *Proceedings of the International Conference on Numerical Modeling Strategies for Sustainable Concrete Structures (SSCS)*, pages 1–9, Aix-en-Provence, France, 2012.