GPUSPH User Guide

version 5.0 — October 2016

Contents

1 Introduction

There are two ways to set up cases for GPUSPH: coding a `Case` file, or using the SALOME module GPUSPH solver.
When coding the case file, it is possible to create the geometrical elements using built-in functions of GPUSPH (only for particle-type boundary conditions at the moment) or to read particle files generated by the Particle Preprocessor module of SALOME. Creating a case by hand corresponds to the creation of a new `cu`source file, with the associated header (e.g. `MyCase.cu` and `MyCase.h`), placing them under `src/problems/user`. This folder does not exist by default in GPUSPH, but it is recognised as a place to be scanned for case sources. Beginners should use one of the provided sample files in `src/problems` as a template for their project. There are two main samples available in the `src/problems` directory: `ProblemExample` (for Lennard–Jones or dynamic boundaries) and `CompleteSaExample` (for semi-analytical boundaries).

2 Anatomy of a project apart from the use of SALOME

Below are the steps required to build a new project and run it with GPUSPH, without using the SALOME graphical interface:

1. in case you choose to read the particle data from a file, create the particle files using the Particle Preprocessor module, following its documentation available from Salome or those of the validation document;

2. create `MyCase.cu` and `MyCase.h` files in the `src/problems/user` directory;

3. in the GPUSPH folder, compile the code for your project:

   ```
   make MyCase
   ```

4. execute the test case:

   ```
   ./MyCase
   ```

5. follow the steps described in the section 6 to visualize and post-process the results.

In case you use the SALOME modules to perform the pre-processing and to run the solver, you need to follow the workflow provided in SALOME:

1. Create the geometry in the GEOM module
2. Set up the parameters for pre-processing in the Particle preprocessor and execute the pre-processing
3. Set up the GPUSPH simulation parameters in the GPUSPH solver module
4. Generate the GPUSPH source files from the GPUSPH solver interface
5. Compile GPUSPH from the GPUSPH solver interface
6. Execute GPUSPH from the GPUSPH solver interface
7. Follow the steps described in the section 6 to visualize and post-process the results.

3 Setting up and running the simulation without using the user interface

As said before, the simulation setup only involves manipulating the .cu and the .h files of your case in order to specify all its parameters before running GPUSPH.
The structure of a case, is in fact the structure of the .cu file, which could be defined as follows:

GEOMETRY. As the mesh geometry has previously been created by CRIXUS, we only have to specify the file containing this information: the .h5sph file.

SIMULATION PARAMETERS. There are several simulation parameters that need to be specified, concerning the time, the frequency of output writing or specific SPH parameters.

INITIAL CONDITIONS. We need to specify an initial value for each of the fields to be implemented on each particle.

BOUNDARY CONDITIONS. Boundary conditions have also to be stated before running GPUSPH.

In the following sections we develop each section to help the readers write the .cu file in order to build their own simulation.
Creating new cases in GPUSPH is done by creating a new class (whose name matches the case name) that derives from the `Problem` class.
As new features are introduced in GPUSPH, at times it becomes necessary to change the interface (API: Application Programming Interface) of the `Problem` class to support them. When the API has to be changed in an incompatible manner, a new version of `Problem` will be introduced. To preserve compatibility with existing cases, the `Problem` class is *versioned*, and cases should specify which version they intend to use. This is achieved by defining `PROBLEM_API` before the inclusion of the `Problem.h` interface header; for example:

```
#define PROBLEM_API 1
#include "Problem.h"
```

There are currently two `Problem` APIs defined: API version 0 corresponds to the low-level `Problem` interface available in GPUSPH up to version 4, and API version 1, corresponding to the high-level interface that was made available experimentally in GPUSPH version 4. Users are encouraged to rely on API version 1, since version 0 is not guaranteed to be stable.
The high-level `Problem` interface exposed in API version 1 simplifies much of the work needed in defining in placing objects and geometries that can be described as a combination of shape, particle type, filling type and intersection behavior.
New geometries are added to the case with commands of the form [escapeinside=<>]add<Shape> that take as parameter a `GeometryType`, a `FillType`, the reference point location, and the dimension(s). For example:

```
// add a boundary cube with corner in the origin. The side lengths a
// in the order X, Y, Z
addCube(GT_FIXED_BOUNDARY, FT_BORDER, Point(0, 0, 0), 10, 4, 8);
// add a sphere centered in (5, 2, 4) and radius 1
addSphere(GT_FLUID, FT_SOLID, Point(5, 2, 4), 1);
// remove a smaller sphere inside the fluid sphere
addSphere(GT_FLUID, FT_NOFILL, Point(5, 2, 4), 0.5);
```

The possible values for the `GeometryType` are:

GT_FLUID for fluid bodies;

GT_FIXED_BOUNDARY for fixed boundaries such as walls;

GT_OPEN_BOUNDARY for open boundaries (inlets, outlets);

GT_FLOATING_BODY for objects whose motions is determined by the interaction with the fluid;

GT_MOVING_BODY for objects whose motions is prescribed by the user (as defined in the `moving_bodies_callback` function that must be defined in the test case);

GT_PLANE, GT_DEM special fixed boundaries whose interaction with the fluid can be described geometrically; this is an internal type and there is no need to specify it explicitly (`addPlane` and `addDEM` handle them automatically);

GT_TESTPOINTS special particles that act as probes; this is an internal type and there is no need to specify it explicitly (`addTestPoint` handles them automatically);

GT_FREE_SURFACE a sepcial set of particles used to describe the free surface during the initial repacking, if enabled;

The possible values for the `FillType` are

FT_NOFILL no particles will be created for this geometry;

FT_UNFILL no particles will be created for this geometry, and the geometry will only be used to cut (by intersection or subtraction) other geometries; this is needed for planes, to differentiate between their use as geometric boundaries (`FT_NOFILL`) and their use to cut other geometries (`FT_UNFILL`);

FT_SOLID both the boundary and the interior of the geometry will be filled with particles;

FT_SOLID_BORDERLESS only the interior of the geometry will be filled with particles;

The `Problem` API 1 also supports the import of meshes and cloud points as produced e.g. by the Particle preprocessor using methods such as `addSTLMesh`, `addOBJMesh`, `addHD5File` and `addXYZFile`.
It has the added feature of taking care of the setup for floating or moving bodies that are handled through a dynamics library: `Project Chrono`. It also takes care of the open boundaries identification, and their type (pressure driven or velocity driven). Simple initializations such as hydrostatic density configurations or setting up the mass for the objects can be automatically performed by this API. More sophisticated initializations (e.g. as required by multi-fluid simulations) can be coded by the user by defining a `initializeParticles` method.

3.1 Case Examples

The supplied Case examples are located in the `src/problems` directory:

AccuracyTest schematic single-fluid dam break case on a flat bottom;

Bubble two-phase flow case, representing the motion of a bubble lighter than the surrounding fluid;

BuoyancyTest a rectangular tank of still water with a submerged torus that is released when the problem begins;

DamBreak3D schematic single-fluid dam break case with an obstacle;

DamBreakGate same case as the previous one but the dam break is managed by a vertically sliding gate;

DynBoundsExample double-periodic channel flow;

InputProblem several problems are included in this one, all based on the semi-analytical boundaries.

OffshorePile waves propagating in a y-periodic channel and hitting a cylinder;

OilJet oil contained in a tube propelled by a piston towards the top and flowing on a plate;

OpenChannel channel flow (with y-periodicity or side walls);

Seiche sloshing case;

SolitaryWave solitary wave generated by a piston. Possibility to add cylinders on the waves path;

Spheric2LJ schematic dam break case on a obstacle (available measurements) – Lennard–Jones boundary conditions;

Spheric2SA schematic dam break case on a obstacle (available measurements) – semi-analytical boundary conditions;

StillWater still water case;

DEMExample example of use of a topography file for the geometry;

WaveTank wave generation on an inclined bed;

Objects example showing how to handle moving objects;

CompleteSaExample generic example for semi-analytical boundaries;

ProblemExample generic example for dynamic and Lennard–Jones boundaries.

Each of these examples can be run by typing `make ProblemName`, in the top level GPUSPH directory. It is recommended that the user try them to ensure everything checks out in terms of CUDA and GPUSPH.
Some of the problems are described with more details below.
`BuoyancyTest` includes a rectangular tank of still water with a submerged torus (or by changing object_type, a cube or sphere) that is released when the problem begins. As time advances, the torus rises through the water column as it has a density half that of water and then it reaches the free surface and floats.
The output of `BuoyancyTest` is written every 0.01 seconds into a file in the directory tests designated by the problem name and the date and time. The files are in VTU format that can be read by ParaView. Alternative formats, such as text, can be chosen by changing the writer in the `add_writer` command.
`ProblemExample` shows how a matrix of objects can easily be added to a problem. The basic problem is a semi-infinite domain with a plane used as a floor (`addPlane`). A 4 x 4 array of solid cubes is set-up using the addCube command multiple times. The `GT_FIXED_BOUNDARY` (GT=Geometry Type) means that the cubes are solid. The cubes are also rotated 45 degrees by a rotate command. Then a smaller array of spheres of fluid are defined. `GT_FLUID` is used in the `addSphere` command. Note for the fluid the `setDensityByMass` establishes the fluid density.
In `DamBreak3D` `makeUniverseBox()`, which has as its arguments two opposite corners of the project domainthe first corner is the origin. This command sets up the domain using analytical planes as boundaries. These planes do not require the use of particles. Water is added to the domain with the `addBox()` command – note that the fluid is denoted by `GT_FLUID` (GT=GeometryType). The fluid behind the dam is 0.4 m deep. A variety of obstacles can be added in front of the dam. As provided, there is just a single object, but by invoking the model with `./GPUSPH --num_obstacles 3` three obstacles will be in front of the dam. These obstacles can be rotated from their original position by `./GPUSPH --num_obstacles 3 --rotate_obstacle true`
Another run-time option includes `--wet true or false`, which puts a $0.1m$ layer of water around the obstacles (and in front of the dam).
`CompleteSaExample` is an example using the Semi-Analytical boundary conditions (SA). This type of boundary condition was chosen in the `SETUP_FRAMEWORK`, which

is a class that contains the various simulation choices. For example it contains `boundary<SA_BOUNDARY>` as the choice. This example consists of a tank with a free surface and a submerged inlet. There is a floating cube as well. The example requires data files that are available on the `www.gpusph.org` web site: `wget http://www.gpusph.org/downl`
or, in your browser, `www.gpusph.org/downloads/data_files_XCompleteSaExample.tgz`
The file (`data_files_XCompleteSaExample.tgz`) is uncompressed in the root GPUSPH directory. It will create a directory `data_files`, containing four `.h5sph` to set up the fluid and boundaries and one `.stl` file to define the cube. (In addition there are five files that were used to generate the input files using Crixus, an open source pre-processor). The problem is large and will take some time as it involves the semi-analytical boundaries. There are 122,642 particles in total, of which 56821 are fluid particles and the rest are boundary and vertex particles.
To write your own example, you can use one of the examples as a template, but they all have a similar format as ProblemExample. For example, looking at the file, `BuoyancyTest` in the directory `src/problems`, we see that, after the appropriate `includes`, including `BuoyancyTest.h`, the example is defined as a child of the `Problem` class. Then the setup is done following the structure below.

3.1.1 Framework setup

The `SETUP_FRAMEWORK` function enables to change the general options of the simulation. The general format is

```
SETUP_FRAMEWORK(
  kernel<WENDLAND>,
  formulation<SPH_F1>,
  densitydiffusion<BREZZI>,
  rheology<NEWTONIAN>,
  turbulence_model<LAMINAR_FLOW>,
  computational_visc<KINEMATIC>,
  visc_model<MORRIS>,
  visc_average<ARITHMETIC>,
  boundary<SA_BOUNDARY>,
  periodicity<PERIODIC_NONE>,
  add_flags<ENABLE_INLET_OUTLET | ENABLE_DENSITY_SUM
      | ENABLE_MOVING_BODIES | ENABLE_REPACKING>
);
```

where any of the options can be omitted to leave the default. The available options are:

kernel for the choice of SPH kernel; possible values:

QUADRATIC

CUBICSPLINE

WENDLAND

GAUSSIAN

formulation for the choice of SPH formulation; possible values:

SPH_F1 standard WCSPH single-fluid formulation;

SPH_F2 WCSPH formulation for multiple fluids;

SPH_GRENIER Grenier's WCSPH formulation for multiple fluids;

densitydiffusion for the specification of the density diffusion model; possible values:

DENSITY_DIFFUSION_NONE for no density diffusion

FERRARI

COLAGROSSI

BREZZI

rheology for the rheological model; possible values:

INVISCID no laminar viscous contribution;

NEWTONIAN constant (per-fluid) kinematic viscosity;

BINGHAM Bingham plastic rheology (constant viscosity with a yield strength);

PAPANASTASIOU regularized Bingham plastic;

POWER_LAW power-law rheology (stress proportional to a power of the strain rate);

HERSCHEL_BULKLEY power-law rheology with a yield strength;

ALEXANDROU regularized Herschel–Bulkley rheology;

DEKEE_TURCOTTE exponential rheology with a yield strength;

ZHU regularized DeKee & Turcotte rheology;

visc_model for the viscous model; possible values:

MORRIS from Morris et al., JCP 1997

MONAGHAN from Monaghan & Gingold, JCP 1983

ESPANOL_REVENGA from Español & Revenga, Phys Rev E 2003

visc_average for the viscous averaging operator; possible values:

ARITHMETIC arithmetic mean ($(a+b)/2$);

HARMONIC harmonic mean ($2ab/(a+b)$);

GEOMETRIC geometric mean ($\sqrt{ab}$);

computationa_visc for the computational viscosity choice; possible values:

KINEMATIC computations are built around the kinematic viscosity (SI units: $\mathrm{m^2 s^{-1}}$);

DYNAMIC computations are built around the dynamic viscosity (SI units: Pa s);

turbulence_model for the specification of the turbulence model; possible values:

LAMINAR_FLOW for no turbulence model;

ARTIFICIAL Monaghan's artificial viscosity; while strictly speaking not a turbulence model, it behaves in a similar way and is thus included here; usually used in combination with the `INVISCID` rheological model;

SPS sub-particle scale turbulence model;

KEPSILON $\kappa - \varepsilon$ turbulence model;

boundary for the boundary model; possible values:

LJ_BOUNDARY Lennard–Jones

MK_BOUNDARY Monaghan–Kajtar

DYN_BOUNDARY dynamic particles;

SA_BOUNDARY semi-analytical model;

periodicity to denote the directions in which the domain is periodic; possible values:

PERIODIC_NONE no periodicity

PERIODIC_X periodic in the X direction;

PERIODIC_Y periodic in the Y direction;

PERIODIC_Z periodic in the Z direction;

PERIODIC_XY periodic in X and Y;

PERIODIC_XZ periodic in X and Z;

PERIODIC_YZ periodic in Y and Z;

PERIODIC_XYZ periodic in all directions;

add_flags can be used to enable individual features, such as support for open boundaries or adaptive time-stepping; default flags can be disabled with the `disable_flags` option; possible values:

ENABLE_DTADAPT adaptive time-stepping (enabled by default);

ENABLE_XSPH XSPH correction;

ENABLE_PLANES support for geometric plane boundaries

ENABLE_DEM support for geometric Digital Elevation Model boundaries;

ENABLE_MOVING_BODIES support for moving boundaries;

ENABLE_INLET_OUTLET support for open boundaries;

ENABLE_WATER_DEPTH computation of water depth at pressure boundaries

ENABLE_DENSITY_SUM density sum instead of continuity equation

ENABLE_GAMMA_QUADRATURE numerical quadrature of semi-analytical gamma

ENABLE_INTERNAL_ENERGY internal energy computation

ENABLE_REPACKING (experimental) repacking feature

3.1.2 Generic simulation parameters

```
// Initialization of simulation parameters
m_name = "XCompleteSaExample";
set_deltap(0.02f);
physparams()->r0 = m_deltap;
// Set world size and origin.
// HDF5 file loading does not support bounding box
// detection yet
```

```
const double MARGIN = 0.1;
const double INLET_BOX_LENGTH = 0.25;
// size of the main cube, excluding the
// inlet and any margin
double box_l, box_w, box_h;
box_l = box_w = box_h = 1.0;
// world size
double world_l = box_l + INLET_BOX_LENGTH
    + 2 * MARGIN; // length is 1 (box) + 0.2 (inlet box length)
double world_w = box_w + 2 * MARGIN;
double world_h = box_h + 2 * MARGIN;
m_origin = make_double3(- INLET_BOX_LENGTH - MARGIN,
    - MARGIN, - MARGIN);
m_size = make_double3(world_l, world_w ,world_h);
// time parameters
simparams()->tend = 40.0;
simparams()->dt = 0.00004f;
simparams()->dtadaptfactor = 0.3;
```

- `m_name` is the problem name.
- `deltap` is the distance between particles used in the current simulation. For consistency reasons, it has to be the same that we have set in the INI file for CRIXUS.
- `m_size` and `m_origin` are the size and the origin of the domain (defined in SALOME in case semi-analytical boundaries are used).
- `tend` is the time at which the simulation should stop.
- `dt` is the size of the first time-step or the time-step size if the ENABLE_DTADAPT flag is not activated.
- `dtadaptfactor` is the CFL coefficient, usually taken as 0.3.

3.1.3 SPH parameters

```
// Initialization of SPH parameters
simparams()->neiblistsize = 256;
// For SA boundaries, specify the amount of non-vertex neighbours
```

```
// vs the amount of vertex neighbours
resize_neiblist(192, 256-192)
// buildneibs at every iteration
simparams()->buildneibsfreq = 1;
// Slightly extend kernel radius for gamma computation
simparams()->nlexpansionfactor = 1.1;
// Density diffusion
simparams()->densityDiffCoeff = 0.1;
```

- `neiblistsize` is simply a limit for the number of neighbors computed in the SPH method. It allows the user control the amount of calculus done for each particle at each iteration. If the mesh of the geometry is coherent with `deltap`, the maximum neighbor number should not be over 280, so setting this number to 300 would be a good choice.

- `resize_neiblist` is a function used for SA boundaries to specify the maximum amount of non-vertex neighbours (in this case, 192) vs the maximum number of vertex neighbours per particle (in this case, 256-192).

- `buildneibsfreq` is the neighbour counting frequency, in terms of number of time steps.

- `nlexpansionfactor` is the factor increasing the area where we count the neighbor particles for the computation of γ with SA boundaries.

- `densityDiffCoeff` is the density diffusion coefficient, which is used to potentiate diffusion dissipation (0 for no extra diffusion, 1 for the maximum)

3.1.4 Physical parameters

```
physparams()->gravity = make_float3(0.0, 0.0, -9.81);
size_t water = add_fluid(1000.0);
set_kinematic_visc(0, 1.0e-2f);
set_equation_of_state(water, 7.0f, 50.0f);
enableHydrostaticFilling();
```

This specifies the gravity field, the fluid density (through the `add_fluid` function), and the equation of state to be used through `set_equation_of_state`. In this function, the first argument is the fluid considered, the second one is the exponent in the equation of state (usually 7), and the third one is the numerical speed of sound.

The numerical speed of sound can also be set by specifying reference velocity and water height:

```
// Reference quantities for speed of sound computation
setWaterLevel(0.5);
setMaxParticleSpeed(7.0);
```

An initial hydrostatic pressure is prescribed in the domain. The code automatically finds out which is the highest particle in the domain, and initializes the pressure based on that value. To change the water level to be considered in the initialization, the function `setWaterLevel` can be used. To disable the hydrostatic initialization, use:

```
disableHydrostaticFilling();
```

3.1.5 Results parameters

```
// Drawing and saving times
add_writer(VTKWRITER, 1e-1f);
```

The writer (for the output data) is chosen with the `add_writer` command (usually the VTK writer, which provides files to be read by ParaView). The file writing frequency (in terms of simulated seconds) can also be specified. That is, in this case, we will have a VTU file every 0.1 simulated second for example. It is important to note that, since some simulations could become too large, this frequency is essential in order to limit the size of the result files.

3.2 Building and initializing the particle system

With the internal GPUSPH geometrical elements:
With DYNAMIC or LENNARD-JONES boundary conditions, the problem geometry and the filling with particles can be done inside GPUSPH. An example of generation of arrays of cubes and spheres in the computational domain is given in `ProblemExample.cu`. The geometrical objects can be added using functions like:

```
addCube(GT_FIXED_BOUNDARY, FT_BORDER,
              Point(X,Y,Z),cube_size);
```

The geometry type (GT) may be fluid, fixed boundary, open boundary, floating body, moving body, plane, and testpoint, as discussed in section 3.
GPUSPH has a variety of geometrical objects that can be used to generate Problems. The geometrical objects are defined in the `src/geometry` folder. The `Problem` API 1 makes it possible to rotate or shift them after they were defined. They can be assigned a mass and a center of gravity. In two dimensions, the objects (in C++ terms, classes)

include *Point, Vector, Segment, Rect (rectangle), Circle*. In three dimensions, there are additional objects: *Cone, Cube, Cylinder, Sphere and TopoCube*. Using these objects, many types of Problems can be constructed. For the three dimensional case, the bottom (bathymetry) of the problem domain can be input via a file, using the TopoCube object and a DEM file.

The *Point* object is usually used as a three dimensional object containing the location of a point in three dimensions. All numbers are double precision. Associated with the Point object are functions that determine distance (or distance squared) of a point from the origin or the distance from another point.

A *Vector* object is a three dimensional double precision object of three space coordinates, x,y, and z. Vector has a number of associated and useful functions, such as Vector.norm, for the length of the vector.

The *Cube* object is really a parallelepipeds, defined by an origin, given by a Point object, and three vectors are used to define the size and orientation of the cube. For example, here is a box that delimits an experimental domain (taken from the DamBreak3D.cc example), called *experiment_box*.

experiment_box = Cube(Point(0, 0, 0),Vector(1.6, 0, 0),Vector(0, 0.67, 0), Vector(0, 0, 0.4));

This box has a corner located at the origin of the domain, with $(x, y, z) = (0, 0, 0)$, and three vectors from this point describe the cube, which happens to be 1.6 m long in the x direction, 0.67 m long in the y direction, and 0.4 in the z direction.

So far we have only defined the cube *experimentbox*, we have given it no properties. For this particular box, which bounds the computational domain, its bottom and four sides will be set as boundary particles, as we will see later.

Associated with the Cube object are commands to fill the inner part of the box with particles, or to fill the boundaries as with boundary particles.

The *Cylinder* object is defined by a point that determines the location of the center of the disk that forms its base, a vector that defines the radius about the point, and then another vector that defined the height of the cylinder. The cylinder object also has fill and FillBorder commands. For example,

jet = Cylinder(Point(0.,0.,0.), Vector(0.5,0.,0.), Vector(0.,0.,1.));

would define a cylinder located at the origin with radius 0.5 and height 1.0 with the name jet. The Cylinder object can be used to define a cylindrical column of fluid, using the `jet.Fill` command for the defined cylinder, jet. The mass of the particles forming jet is set by `jet.SetPartMass` function. If the jet was supposed to be a pipe, the `jet.FillBorder`, with suitable arguments, would use boundary particles for the pipe called jet. Two of the arguments (Booleans: true or false) of the method determine if the cylinder is closed on the bottom or the top.

The *Sphere* object is defined by a point that determines the center of the sphere, a vector that determines its radius (and equatorial normal), and a vector pointing to the sphere's pole. For a sphere, these two vectors have equal magnitude and are normal to each other. The Sphere object uses the Circle object in layers to create a sphere.

A *TopoCube* object is used to define a domain that has the bottom of the cube provided by a data file. The geometry of the TopoCube is determined the same was as in the Cube object. The data file has a strict format; for example:

north: 13.2
south: -0.2
east: 43.2
west: 0.54
rows: 134
cols: 432
{data in 134 rows with 432 entries per line; numbers space separated}

The numbers following the compass directions are the length of the domain described by the data, in meters. (North and south correspond to the +Y axis and the -Y axis, while E and W are aligned with the +X and -X directions.) The internal variables (see problem TestTopo.cc) *nsres* and *ewres* are grid resolutions determined by $nsres = (north - south)/(nrows - 1)$ and $ewres = (east - west)/(ncols - 1)$.

The data file is read using the TopoCube.SetCubeDem function, which is called with arguments (float H, float *dem, int ncols, int nrows, float nsres, float ewres, bool interpol), where H is the depth of the cube, *dem points to the array of bathymetric data in the data file, ncols and nrows are the number of columns and rows in the dem data set, nsres and ewres is the spacing between the bathymetric data in the north/south direction and the east/west direction, and interpol (not the police) is the boolean variable for interpolation. FillBorder will fill a face with particles–the particular face is determined by face_num, which takes on the values of (0,1,2,3), for the front face, the right side face, the back face, and the left side face (facing the -x direction) for a rectangular box.

Other objects can be defined and added to the source directory to allow for additional flexibility.

Reading particle files:

The fluid initialization performed by the Particle preprocessor and stored in the H5SPH files can be used by GPUSPH to start the simulation with any type of boundary conditions. The specification of the file containing the fluid particles occurs

with the following statement:

```
addHDF5File(GT_FLUID, Point(0,0,0),
"./data_files/MyCase/my_case.fluid.h5sph",
NULL);
```

The specification of the file containing the special boundary particles occurs with the following statement:

```
// Main container
GeometryID container =
addHDF5File(GT_FIXED_BOUNDARY, Point(0,0,0),
  "./data_files/MyCase/my_case.boundary.h5sph",
  NULL);
disableCollisions(container);

// Inflow boundary
GeometryID inlet =
  addHDF5File(GT_OPENBOUNDARY, Point(0,0,0),
  "./data_files/MyProject/0.my_project.boundary.kent1.h5sph",
  NULL);
disableCollisions(inlet);

GeometryID cube =
  addHDF5File(GT_FLOATING_BODY, Point(0,0,0),
  "./data_files/MyProject/0.my_project.boundary.kent2.h5sph",
  "./data_files/MyProject/MyProject_object_file.stl");
// output forces on the cube
enableFeedback(cube);
// set the cube density
setMassByDensity(cube, 500);
```

In order to specify whether the open boundary is pressure driven or velocity driven, the following lines are used:

```
setVelocityDriven(inlet, 1);
setVelocityDriven(outlet, 0);
```

Once again the GT (GeometryType) can be fluid, fixed boundary, open boundary, floating body, moving body, plane, or testpoint, eg. `GT_OPENBOUNDARY`.

4 Running your simulation

To run your simulation you first need to compile GPUSPH for your problem. To do so, in the GPUSPH folder, run:

```
make MyProblem
```

Remark:

- If you are running a multi-node simulation, do not forget to add the option `mpi=1`.
- If your are running a simulation with moving objects, do not forget to add the option `chrono=1`.

See the installation guide or run `make --help` for the complete list of compilation options.

5 Setting up and running the simulation with the SALOME user interface

In order to start a new project in SALOME, click on `File/New`. When you save your project, SALOME creates a file with the `.hdf` extension, which stores all the geometry elements, meshes and simulation parameters that you design for your project.

5.1 Preparing the geometry in GEOM

The complete SALOME documentation for the GEOM module can be found here: `http://docs.salome-platform.org/7/gui/GEOM/` or from the Salome Help menu. Designing is easy in SALOME. To start building the geometry elements, click on the Geometry module. There are 7 types of basic geometrical elements:

1. VERTEX: it can be created by providing its coordinates, by clicking on the vertex of another geometry element, by using another point as referenceThere are many ways which are described in the Point Construction window which appears when we click on Create a point
2. SEGMENT: it can be created providing two points that were previously stated, or using the intersection of two planar elements.

3. WIRE: a wire is just a series of segments. It can be a closed wire or if the end matches the start, or an open one.

4. FACE: a face is just a limited plane

5. SHELL: a shell is a series of faces. SALOME would consider it a closed shell when it encloses a volume

6. SOLID: a solid is just a limited part of the 3D space; it can be easily created on the basis of a closed shell

7. COMPOUND: a compound is just the combination of two or more elements of different type, merged into one single element

Apart from these basic geometry types, we can find three special types, which are DIVIDED DISK, DIVIDED CYLINDER, and SMOOTHING SURFACE. Among these three special types, the most interesting is the smoothing surface, as it is useful to create 3D surfaces from a point cloud. Finally, we can find auxiliary geometry elements such as circles, ellipses, arcs, vectors, sketches, polylines, cylinders, cones, spheres, cubes, torus, disks, T shape pipes, etc.
SALOME makes it possible to import geometries from a wide range of file types: STL, BREP, STEP,etc. It is possible to import a geometry in STL format (generated with another 3D modeling software, such as Autocad, SolidWorks, Catia, Blender, etc.).
Caution: STL files are ASCII or binary files in which geometry is described by triangles. Each element of an STL file is composed by the 3 coordinates of each 3 vertex of the triangle, and the 3 components of the triangles normal vector. This means that when we export some geometry elements in STL format, triangles would be automatically created. When importing this file in SALOME, the geometry is then composed of triangular faces and the meshing operations to be implemented afterwards are influenced by this previous and automatic discretization of the geometry. This results in bad mesh quality. So when importing geometry on a STL format, a redesigning of it is necessary in order to obtain a good mesh quality. Meshlab can be used for this purpose (in particular the Poisson resampler feature).
Other very useful tools of SALOME are the boolean operations on solids. It is possible to fuse, intersect solid objects, use a solid object as a cutting tool for another one, etc.
It is also possible to perform operations like rotation, translation, etc. on the geometrical objects.

5.2 Generating the mesh (optional)

The complete SALOME documentation for the MESH module can be found here: `http://docs.salome-platform.org/latest/gui/SMESH/index.html` or from the Salome Help menu.
The input files for GPUSPH are meshes of:

- the domain's fixed boundaries
- the free-surface
- the special boundaries (for moving objects and/or open boundaries)

These meshes must be composed of triangles of homogeneous size over the domain, otherwise the quality of the GPUSPH results may be affected. The Particle preprocessor will create these meshes, but in case the obtained discretisation is not satisfactory, it is possible to create the meshes in the MESH module and define them as input files for the Particle preprocessor.
In order to access the meshing tools, change from the GEOM module to the MESH module. To create a mesh from a geometry element, click on `Create Mesh`, and a window opens (see Figure 1) in which the following options are available:

- Name: the name of the mesh which is going to be created
- Geometry: the geometry element that we want to mesh
- 3D/2D/1D/0D: its the nature of the mesh that we are going to create, it automatically chooses the correct one depending on the type of element that we have specified in Geometry
- Algorithm: is the meshing methods algorithm. Netgen 1D-2D works well for shell meshing.
- Hypothesis: here we can specify the hypothesis to be used by the algorithm method. Clicking on the first icon on the right, we can specify the parameters of the algorithm. See the SALOME documentation for more details about the options. For example, for the Netgen 1D-2D algorithm, a window opens with all the options shown in the Figure 2.

The most relevant mesh options with the Netgen 1D-2D algorithm are Max Size and Minimum Size. It is important to note that SALOME usually respects the Max Size,

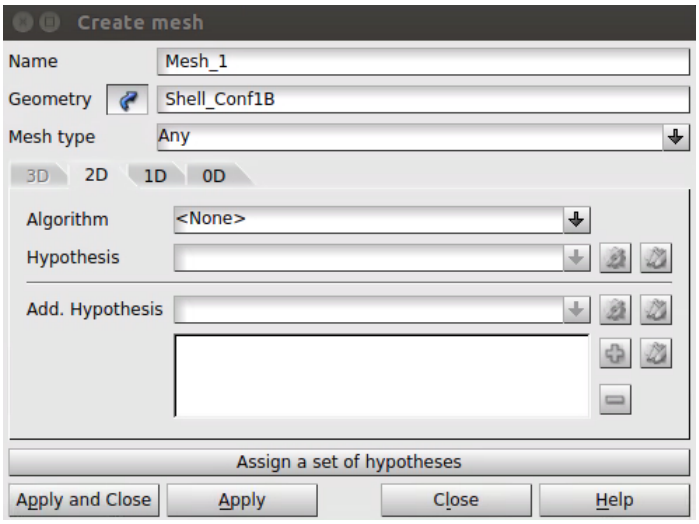

Figure 1: Screenshot of the mesh options window in SALOME.

whereas the minimum size is often ignored due to geometry-mesh adaptation problems. In addition, the minimum size would be always delimited by the characteristic size of the geometry, that is to say, the minimum length of the faces composing the shell. Regarding the option Fineness, Fine works usually well. Once we press OK and then Apply, an element of mesh type will appear in the Object Browser on the left side of the screen. The icon will appear with an exclamation mark on it: that means that the mesh has not been computed yet. To do so, we click on the icon Compute or we just do right click and we select the option Compute. The algorithm will now begin iterating until a solution has been found. The mesh is then ready to be used by the Particle preprocessor.

5.3 Generating particle files with the Particle preprocessor

The full user documentation of the Particle preprocessor is available in the Salome Help menu.

5.4 Setting up and running the simulation with the GPUSPH solver

The full user documentation of the GPUSPH solver is available in the Salome Help menu.

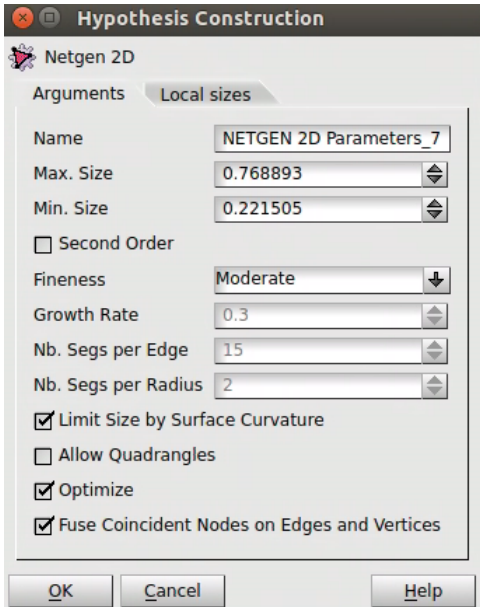

Figure 2: Screenshot of the Netgen 1D-2D hypothesis window in SALOME.

6 Visualizing the results

The results of the simulation are stored in a directory under `tests`, named after the used Problem and the date of execution (e.g. `tests/DamBreak3D_2014-6-12T13h23`). Data files (found in a `data` sub-directory of the specific test directory) are normally written in VTK Unstructured Grid format (`.vtu`) and can be visualized with ParaView.
The files necessary to hotstart the simulation are also stored in the `tests/MyProject_2014-6-12T13h23/data` folder.
The run directories and their content are preserved until manually removed. The `scripts/rmtests` auxiliary script can be used to clean up the `tests` directory.
A tutorial to start using ParaView is available here: `http://www.paraview.org/Wiki/Beginning_ParaView`
To open a file, click on the first upper icon on the left. The VTU files are named as `PART_00025.vtu` where the number corresponds to the output files numbering. PARAVIEW allows the user to visualize at the same time all the VTU files, just clicking on VTUinp.pvd or selecting the all set of `PART_..vtu` files (see the Figure 3). The set of VTU files can be analyzed as a movie by clicking on the play buttons at the top of the screen.

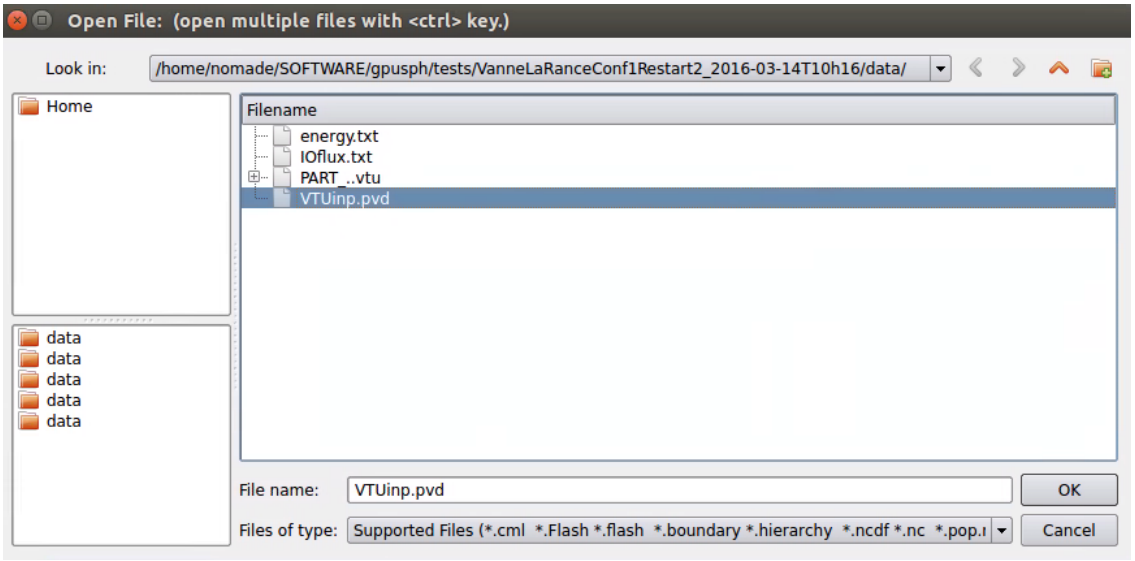

Figure 3: Screenshot of the window for file opening in PARAVIEW.

After selecting the `VTUinp.pvd` or the desired `PART` files, a window appears at the bottom-left part of the screen. Press `Apply` to confirm the file opening: the set of particles appears at the center of the screen. When pressing `Apply`, a window opens with three main sections: `Properties`, `Display` and `Information`. The second one enables the user to decide which field should be printed (first tab of the section `Coloring`). With the `Show` option, we can make the color legend appear, and with the `Edit` one, we can customize it. Below this section we find a set of options that enables us to manage the plotting as desired. The `Information` section provides information like the total size of the dataset.
Below some useful filters of ParaView are listed. The filters are all available through the `Filters` tab at the top of the screen, in the section `Alphabetical`, or through shortcuts in the main window.

- **Find Data** Find data by scalar value makes it possible to select particles on the basis of the value of their fields. When activating a filter a window opens. In order to select the particles we want, in that window (see the Figure 4), we have to set the Find tab on Point in that window. In the left tab we can choose the desired field, whereas in the right one we can set the value (note that there are several conditions: >=, <=, =, between, etc.). Once we have set all the options, we press on Run Selection Query and we will be able to

see a table containing all the particles that match the imposed condition. In addition; we will be able to see these particles on the Layout, colored with the chosen Selection Color.

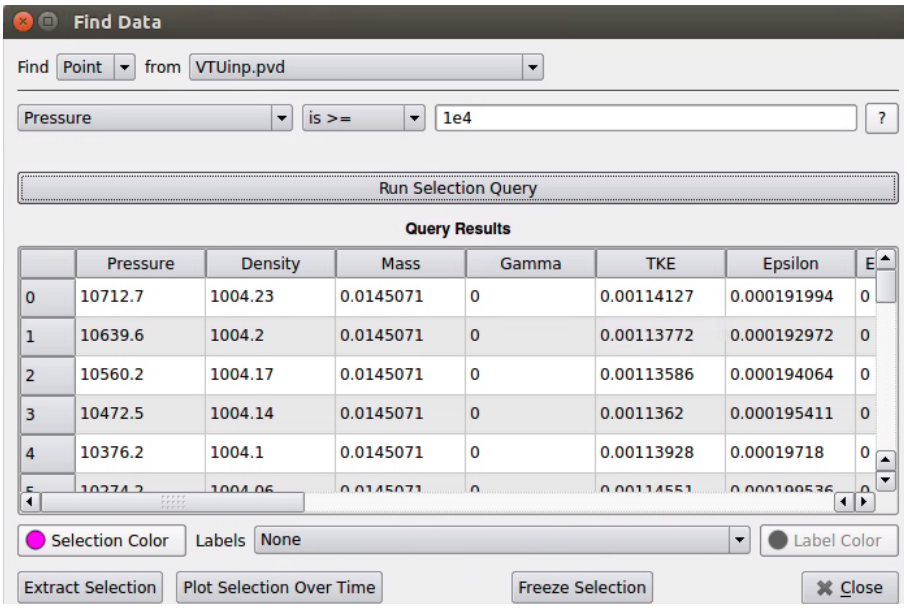

Figure 4: Screenshot of the `Find Data` window in PARAVIEW.

- **Clip** In order to visualize a section of the domain, since the fields are discrete, we are obliged to perform a `Clip`. The `Clip` window is shown in the Figure 5. By changing the `Clip Type` tab into `Box`, it is possible to set the dimensions and position of the box. It is important to click on the `Inside Out` button to select the particles that are inside the box. Once ready, click on `Apply` to get a new `Clip` object in the Pipeline Browser. You can manipulate it in the same way as the main dataset. You can also perform clips with planes. **Remark**: the Slice option does not work because the flow fields are not continuous. To make a slice, we currently apply a thin box-type clip to the dataset.

- Other useful filters are the **Threshold**, **Calculator**, **Scatter Plot**, etc.

Saving your results

Save Data You can generate a table in CSV format containing the values of the fields for each particle for each PART file or filtered dataset. If you click on `File/Save Data`,

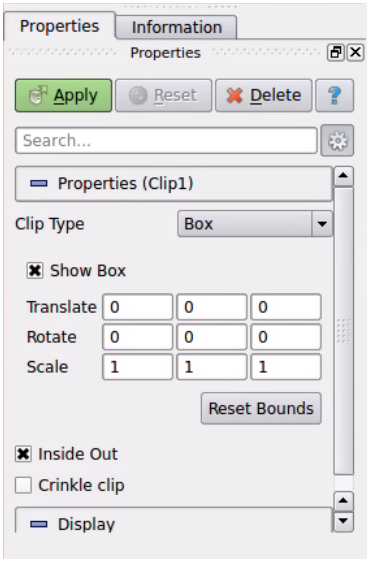

Figure 5: Screenshot of the `Clip` window in PARAVIEW.

a window appears where you can set the name and other options for the result file. There are many options for the format of the file, but the recommended one is .csv, as you can visualize it on the Linux LibreOffice Calc or Windows Excel. In addition, you can change the format of the file to .dat in order to open it with a text editor.
Save State You can save your PARAVIEW postprocessing state in a file by clicking on `File/Save State`. The state file is in ascii format so you can edit it with a text editor. You can also apply it to other datasets than the original one, which is very useful in order to avoid having to repeatedly perform the same filtering operations.