How to use the code

1. General presentation

The file `river.cpp` in `riverCode/apps/` contains the end-user code used to simulate a river flow. To execute the code, a parameter file must be given. In this example, the code is run with the command:

```
./river parameters.xml
```

A STL geometry file is given to the program (see below for details). The code builds a bounce-back matrix according to the geometry described in the STL file. The bed of the river is poured with liquid up to an initial water level. Then, at each iteration, fluid is removed from an outlet volume and added to an inlet volume, in order to make the fluid flow down the river.

A brief explanation of each parameter is explained in the comments of the file `parameters.xml`. I detail below some importants parameters.

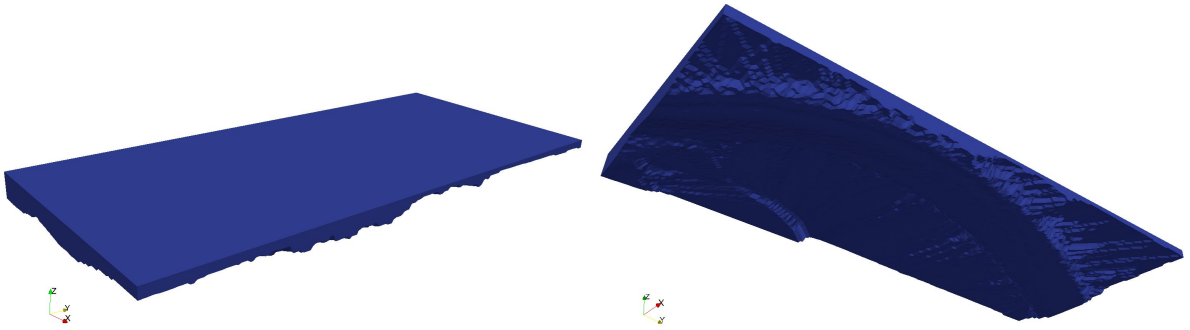
An example of runjob for Baobab cluster is given with the code, as well as a version of palabos for which the code compiles.

2. The bed geometry

In order to work, the code needs two different input geometries. The first one, called “outer bed”, provides the final geometry, whereas the second one, called “outer bed”, provides the initial geometry of the river bed (remember: the code also aims at erosion-sedimentation).

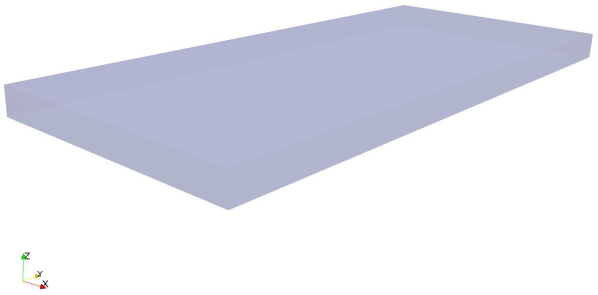
The outer bed has to be understood as a constraint specifying the geometry that cannot be eroded (i.e. it’s more a general constraint on the geometry of the bed rather than the final state that the bed will actually reach, most probably).

The inner bed in the proposed example is displayed below from two different points of view.

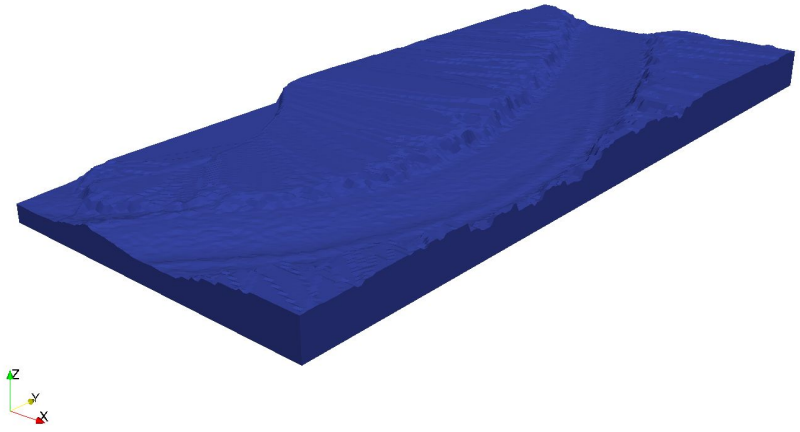


It is important to note that the water flows inside the specified geometry, not outside. Hence, the geometry given to the program represents the zone in which the water has to be comprised.

The outer bed, in the proposed example, is simply a box that is sufficiently large to fully contain the inner bed (here the outer bed is seen with some transparence):



The inner, respectively outer bed geometry as processed by the code correspond to output files `inner_bedRiverGeometry.stl`, respectively `outer_bedRiverGeometry.stl`. These files allow to visualize the bounce-back matrix that encloses the water in the simulation. However, for visualization purpose, an output file called `visu_bedRiverGeometry.stl` is produced, as displayed below. Using this file, you can see the actual bed.



3. How to generate the bed geometry

The inner bed is generated from the input file called `meshTwo`. The outer bed is generated from the input file called `meshOne`.

The method to produce these two files depends on the raw data of the bed river that you have. Below, I describe a general method used on “cloud of points” data. This method is general and quite slow, so if you can exploit any uniformity in your data, your scripts generating the input geometry may be significantly faster than mine. However, keep in mind that you must give to the code STL files that represent the volume in which water is enclosed. This means that you must produce STL files that represent a box with a bottom (i.e. the river bed), four sides and a roof. Also, you can add directly in the input STL files any static structure that you want to include in the simulation (e.g. a dam), using an external 3D modeling tool (Blender works fine).

I provide here a method to produce the input files if you have a raw data of the bed just like the DBF file given as example, which is on the form : `x1 y1 z1 x2 y2 z2 ...`

Note : in the example given, the first two lines are not relevant. They are explicitly ignored in line 16 of the script `convertMesh.py`.

`The script is found in the folder named bathymetry.`

Executing the script as

```
python3 convertMesh.py
```

will produce several files :

- `output_with_no_roof.stl`: this reflects the raw data
- `inner_bed.stl`: inner bed to be given in the parameters of the program
- `visu.stl`: bed to be used for visualization, along with the files `interface*.stl` which represent isosurface of the free-surface fluid.
- `outer_bed.stl`: outer bed to be given in the parameters of the program

Note that by default, matplotlib is also used by the script to produce a graph of the height map.

4. Inlet, outlet and initial water level

The way to add fluid to the simulation and to remove fluid from the simulation described below probably does not correspond to realistic conditions. However, these boundary conditions are robust and, if put far enough from the area of interest of the domain, they should not impact the results as long as the flow or the water level match the desired ones.

Inlet:

The XML parameter file contains the parameters Xi0, Xi1, … , Zi0, Zi1. They represent the coordinates of the box in which fluid is added to the simulation. It must be one cell thick in one of the three directions (i.e. it's a 3D plane with a thickness of one cell).

At each iteration, fluid is added to the simulation through this inlet box with the help of a data processor called `PouringLiquid3D`. It simply initializes at equilibrium the cells inside the inlet box as fully wet cells, using the velocity prescribed in the parameter file as `velocityInletPhys`.

Outlet:

The parameters for the outlet are defined in the same way as for the inlet and are name Xo0, Xo1, … , Zo0, Zo1. Unlike to the inlet box, the outlet box can have any size and is not constrained to be a "plane".

At each iteration, fluid is removed from the simulation inside this outlet box with the help of a data processor called `RemoveMass3D`. It simply sets the mass, volume fraction and velocity value of every cell in the box to zero.

Initial water level:

The XML parameter file contains the parameter `waterLevelFactor`. It is a real number between 0 and 1. A value of 0.25 means that, before the simulation begins, all the empty cells of the domain that are below 25% of the domain's height are filled with fluid. By default, z-direction (index 2) defines the vertical axis.

Once the simulation begins, the fluid is free to evolve (impacted by the inlet and outlet conditions) and the initial water level is not imposed again.

5. Checkpointing

The state of the simulation is saved every `checkpointIter` iterations, as specified in the parameters file. These checkpoint files are by default saved in the same directory as the compiled file and are named `checkpoint_lattice.dat` and `checkpoint_flags.dat`.

To run a new simulation, the parameter `initialTimeStep` must be equal to zero. Otherwise, the checkpoint files will be used as the starting point of the simulation, using `initialTimeStep` value as the number of the first iteration.