User Guide for the Multi-GPU Version of CUTEFLOW

GRANIT – Research Group on Numerical Applications in Engineering
and Technology at École de Technologie Supérieure, Montreal

March 26, 2025

Contents

1 Foreword

The following is a guide to using the *CuteFlow* code and associated pre/post-processing codes. This code is hosted on Github https://github.com/ETS-GRANIT/CuteFlow.

CuteFlow is a parallel finite element code developed in the GRANIT laboratory at ETS. It is dedicated to the solution of the St-Venant equations and is mainly used to simulate dam failures, river flows, floods and determining flood coasts.

Since its initial release, *CuteFlow* has undergone several major evolutions. The first sequential version on CPU was first developed by Azzeddine Soulaïmani and Youssef Loukili [6], then by Jean-Marie Zokagoa [9]. The code was then ported to the GPU in CUDA Fortran by Arun Kumar Suthar [7]. Finally, the code was ported to a multi-GPU architecture by Vincent Delmas [4]. More recent advances include the development of a calibration module for physical simulation parameters (Manning's coefficient) by Igor Metcheka [5].

1.1 Using computing clusters from *Compute Canada*

For information on the use of computing clusters, please refer to the official wiki of *Compute Canada*. https://docs.computecanada.ca/wiki/Compute_Canada_Documentation. The wiki includes information on connecting to compute clusters, data storage, job launch, and interactive node allocation.

For even greater ease of use with compute clusters, it's best to be comfortable with a command-line editor such as vim, emacs or nano, and to be able to split a terminal into several terminals, for example, using tmux https://docs.computecanada.ca/wiki/Tmux (all this is installed by default on compute clusters).

In the following, the examples are given on the assumption that the user is connected to one of *Compute Canada*'s computing clusters. Still, the code can be downloaded to any system. To compile and run the code, however, you'll need to have the various libraries installed and compiled correctly.

1.2 Version management with Git

Git software [1] is used to manage code versions. A tutorial is available on the official website, and another on OpenClassrooms. It simply shows you how to clone the code folder and keep it up to date with any changes made to it. Of course, Git allows you to do many more things, such as go back into the history of files or allow people to work on different branches of the same code.

To clone the project on GRAHAM in the user's scratch space:

```
# On Graham
cd ~/scratch
module load git-lfs

# Generate a cluster authentication key with
ssh-keygen -t rsa -b 4096

# View the key with
cat ~/.ssh/id_rsa.pub
```

The system will then ask you to create a password associated with the key. You'll need to enter it to clone the Git repo. The key will be stored in ***home/UserName/.ssh/id_rsa.pub*** or similar.

Then copy this key to GitHub here https://github.com/settings/keys. The git-clone will then look like this:

```
# On Graham
cd ~/scratch
git clone git@github.com:ETS-GRANIT/CuteFlow
```

A CuteFlow/ folder containing important code files will then be created, and the operation may take several minutes.

If the user inadvertently destroys some of the code files, he can always re-synchronize his local folder with GitHub by doing,

```
# Reset folder to last download status
git reset --hard

# Download changes from source folder on GitHub
git pull
```

Warning: in this way, all modifications made by the user to files managed by Git will be overwritten. This will have no impact on new files and folders that the user may have created. There are other ways of preserving part of the modifications, see https://git-scm.com/docs/gittutorial.

2 Mesh files

2.1 Converting mesh files to CGNS format

Various types of mesh files can be converted to CGNS format using Python, for example. Code to convert older mesh files used by CuteFlow to CGNS format can be found in the CuteFlow/src/cute_to_cgns folder.

With the code cute_to_cgns.py, we read the old mesh files, in particular, node coordinates, element connectivity and input and output nodes, and use the *pyCGNS* library to write a file in CGNS format. The code cute_to_cgns.py is accompanied by a file Readme.txt which explains how to create the Python virtual environment that runs the code.

```
# Once the virtual environment has been created and activated
python3 cute_to_cgns.py mesh.txt multi_entree
```

where, *mesh.txt* is a file using the old mesh file format, and *multi_entree* takes the value 0 or 1 depending on whether the mesh file contains multiple entries. Following code execution, a mesh file ***mesh.cgns*** will be generated.

The code cute_to_cgns.py was used, for example, to convert Mille_Iles_mesh_743968_elts.txt to the new format Mille_Iles_mesh_743968_elts.cgns.

```
 Table de coordonnees
      374619
           1     271700.8750    5042630.5000         30.2758          0.0220
           2     271702.9062    5042625.0000         30.2134          0.0220
           3     271704.5625    5042636.0000         30.3013          0.0220
....
....
      374617     280227.0938    5050397.0000         35.0776          0.0300
      374618     280227.9688    5050384.0000         35.1949          0.0300
      374619     280229.4375    5050392.0000         35.1007          0.0300
 Table de connectivités
      743968
           1           1           2           6          0.0220
           2           1           6           3          0.0220
           3           2           4           6          0.0220
           4           3           6           5          0.0220
....
....
      743965      374611      374616      374618          0.0300
      743966      374611      374618      374613          0.0300
      743967      374613      374618      374619          0.0300
      743968      374613      374619      374617          0.0300
 Noeuds a l entree
         266
           2
....
....
       23926
           1
 Noeuds a la sortie
         342
      374619
....
....
      285251
      284912
 Noeuds aux murs
        4708
       24071
....
....
      139796
      139719
|
```

Figure 1: Mille_Iles_mesh_743968_elts.txt using the old *.txt format read by CuteFlow

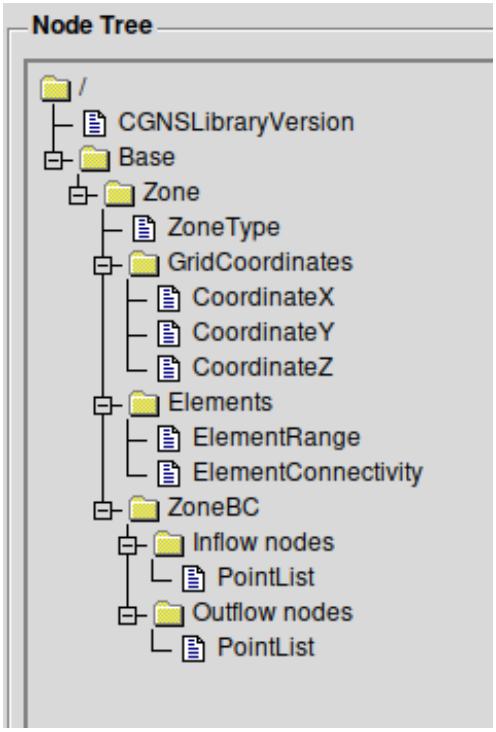


Figure 2: Structure of the new file Mille_Iles_mesh_743968_elts.cgns in *.cgns format

In the old file format shown in Figure 1, we distinguished several sections: A coordinate table containing nodes numbered starting at 1 with their x, y, z coordinates and a Manning number which is now obsolete (a Manning number defined on a zone must now be added as presented in section 2.4); A connectivity table where elements are numbered starting at 1 and where for each (triangular) element the vertices of the element are listed together with a Manning number which is also obsolete; A list of input nodes; A list of output nodes; A list of wall nodes which can be omitted. For each section, the first line corresponds to the number of subsequent lines to be read. In the file Mille_Iles_mesh_743968_elts.txt we can see that there are 374619 nodes, 743968 elements, 266 input nodes, 342 output nodes and, 4708 wall nodes. This file will be converted by the code cute_to_cgns.py to the file Mille_Iles_mesh_743968_elts.cgns whose structure is shown in Figure 2.

The code cute_to_cgns.py will need to be modified and should serve as the basis for converting various types of mesh files to the CGNS format readable by *CuteFlow*.

2.2 Creating a mesh file in Python

CGNS mesh files can be created in Python using the *pyCGNS* library. An example code is given in the CuteFlow/src/-gen_mesh/ folder. The *gen_mesh.py* code is accompanied by a *Readme.txt* file which explains how to create a virtual environment to run the code.

In the *gen_mesh.py* code, we start by creating a list of points in a rectangular domain and use *scipy.Delaunay* to generate the triangulation. You can then use the *pyCGNS* library to create the structure of the CGNS file, starting by creating a base and then a zone, and adding node coordinates and element connectivity before saving the file in CGNS format. Full documentation for *pyCGNS* can be found at https://pycgns.github.io/index.html, in particular the MAP and PAT modules.

Once the virtual environment has been created and activated, you can run the gen_mesh.py code as follows :

```
# Once the virtual environment has been created and activated
python3 gen_mesh.py
```

An output mesh file will then be generated according to the parameters entered in the gen_mesh.py file. The values $x0$, $x1$, $y0$ and $y1$ define the spatial boundaries of the domain, while ny corresponds to the number of points along the y axis. The mesh is generated as homogeneously as possible using equilateral triangles. Examples of meshes generated in this way can be found in the folder CuteFlow/meshes/rect_mesh_convergence/.

2.3 Creating a mesh file with *GMSH*

This section explains how to create a simple mesh file using *gmsh* https://gmsh.info/. You'll need the latest version of *gmsh* to generate files in CGNS format for reading with *CuteFlow*. The CGNSview software https://cgns.github.io/CGNS_docs_current/cgnstools/cgnsview/index.html may be required to edit mesh files by hand if necessary.

In this section, we create a simple mesh of two rectangular floors connected by a channel. We start by adding points to the mesh contour in the "Geometry->Elemental Entities->Add->Points" section. For this example, we add points with coordinates $(0, 0, 10)$, $(0, 10, 10)$, $(10, 10, 10)$,$(10, 0, 10)$, $(10, 7, 10)$, $(10, 3, 10)$, and $(20, 0, 0)$, $(20, 3, 0)$, $(20, 7, 0)$, $(20, 10, 0)$, $(30, 10, 0)$, $(30, 0, 0)$.

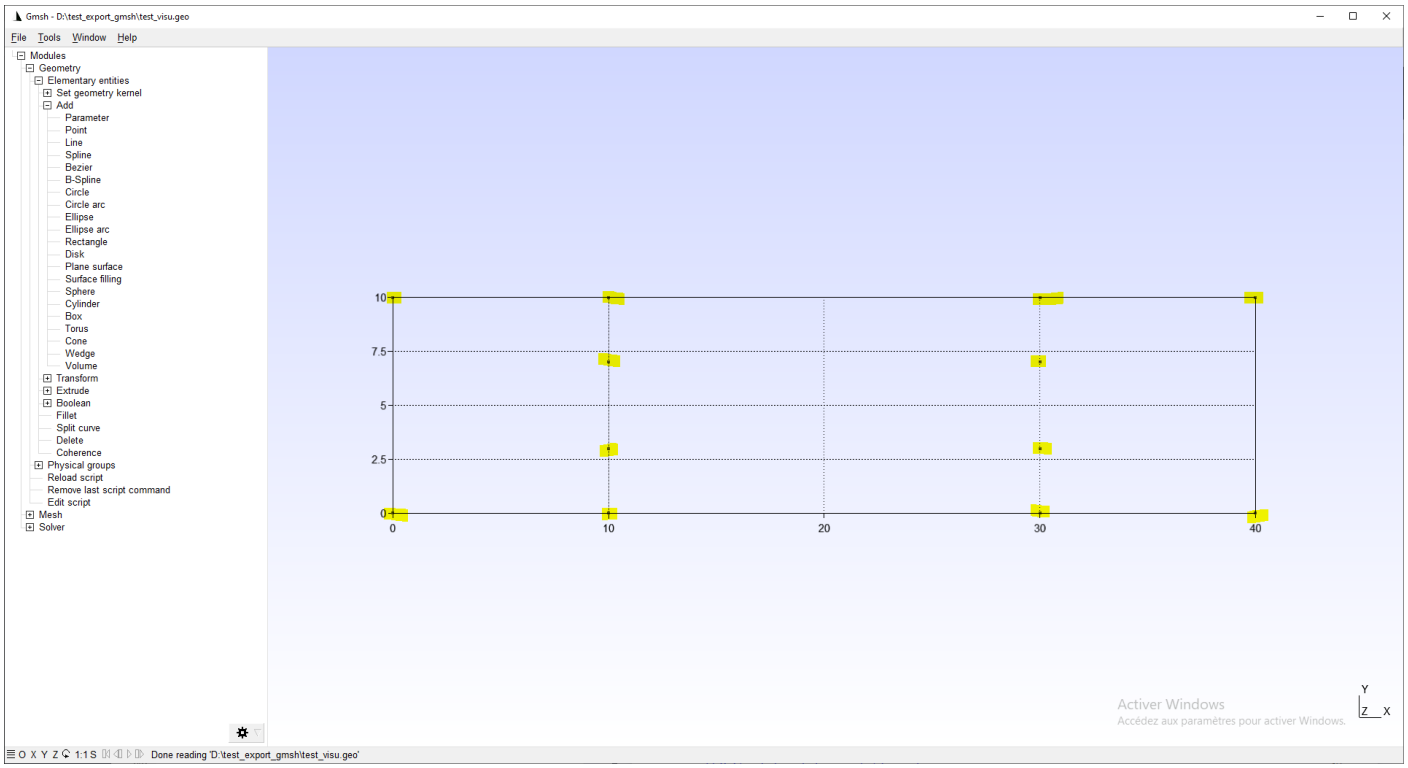


Figure 3: Adding points to the domain

You can then connect the points with lines by going to "Geometry->Elemental Entities->Add->Lines".

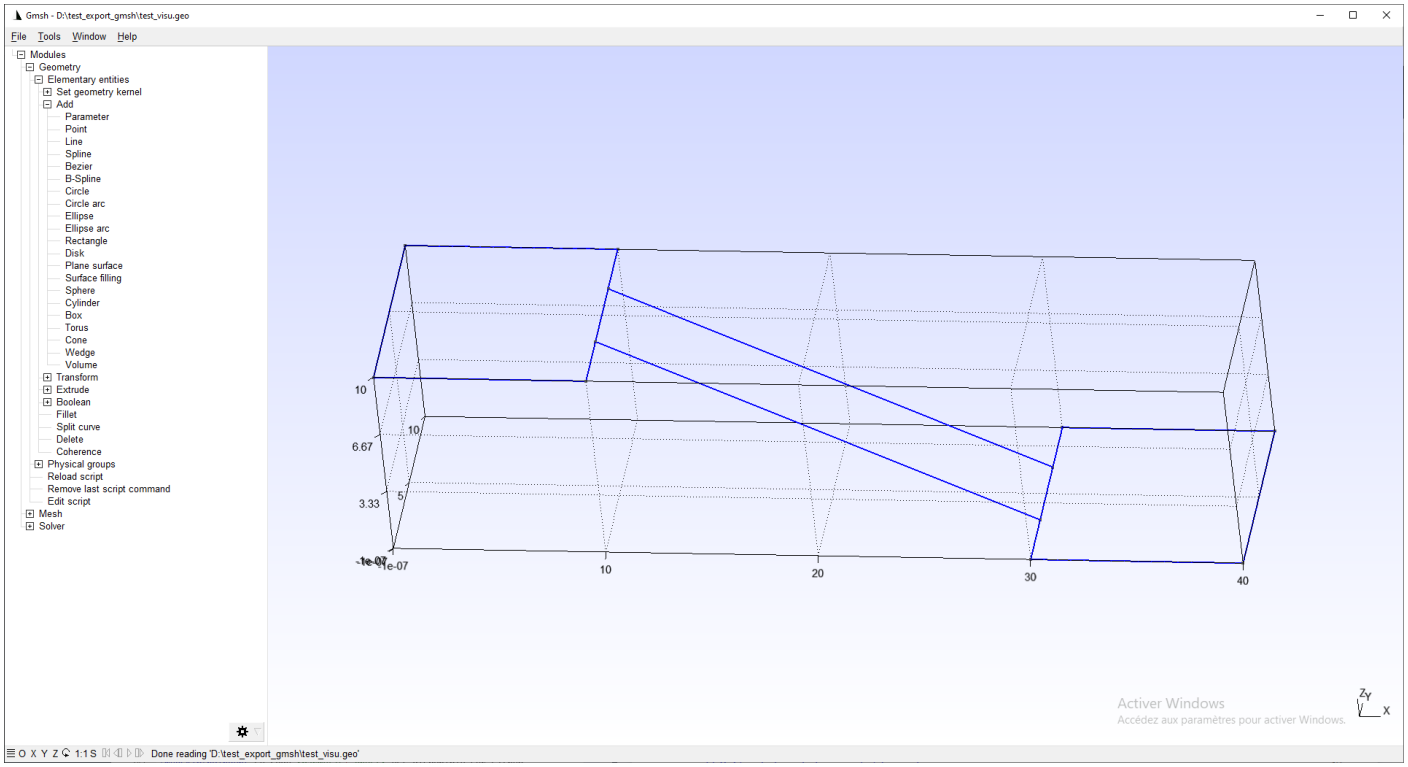


Figure 4: Adding lines to the domain

You can then create plane surfaces on each section of the mesh by going to "Geometry->Elemental Entities->Add->Plane Surfaces".

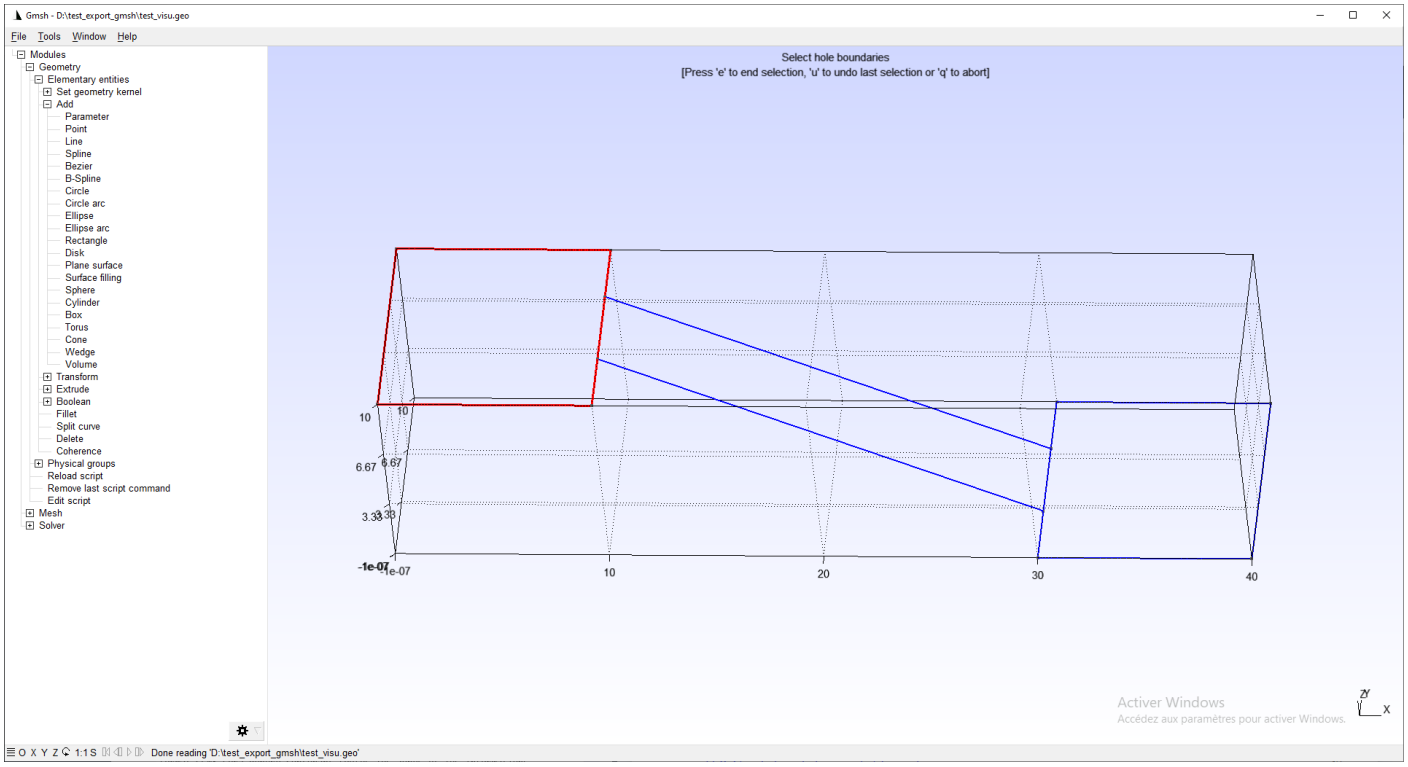


Figure 5: Adding surface 1

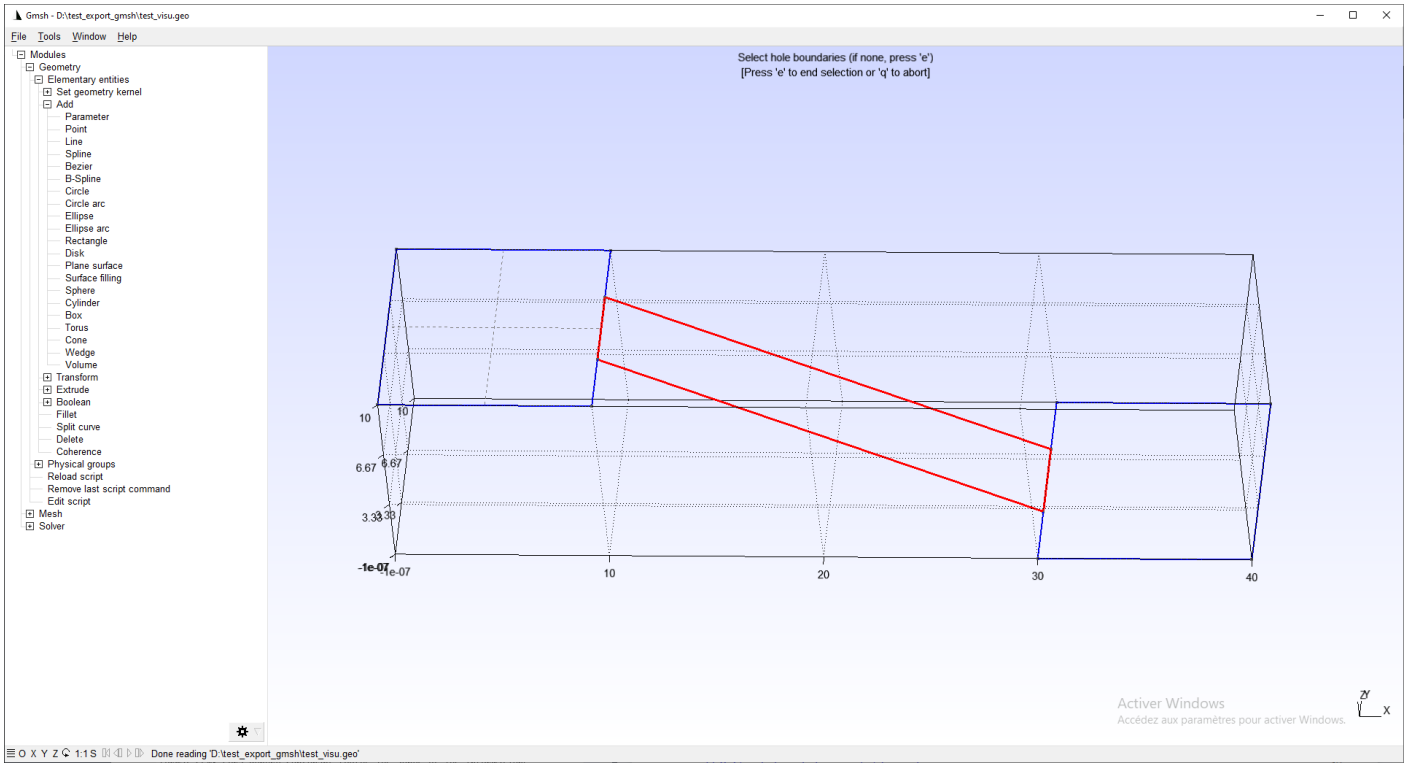


Figure 6: Adding surface 2

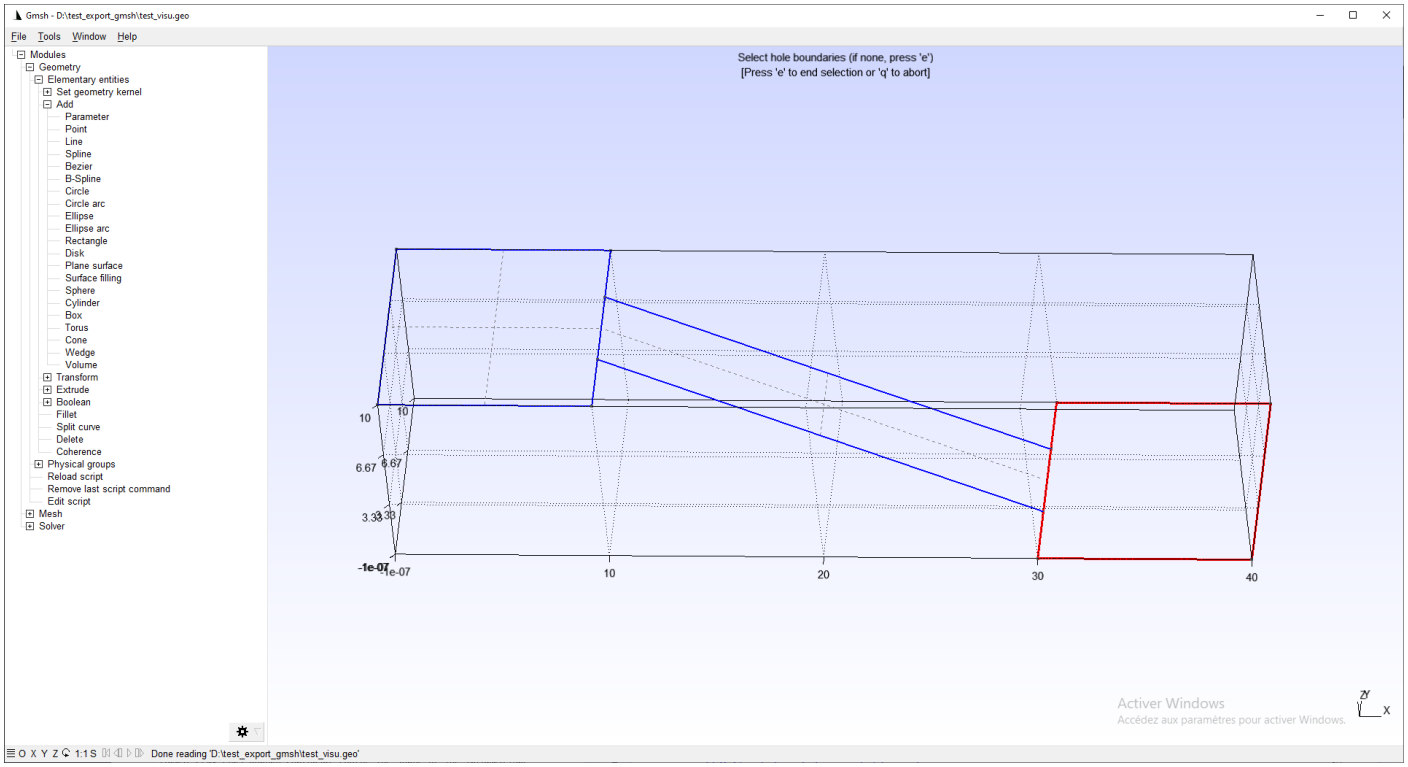


Figure 7: Adding surface 3

If we now go to "Mesh->2D", a separate mesh of the three surfaces will be created. This is a problem because we want to combine the three surfaces into a single one. To do this, go to "Mesh->Define->Comound->Surface" and select the three surfaces to be combined.

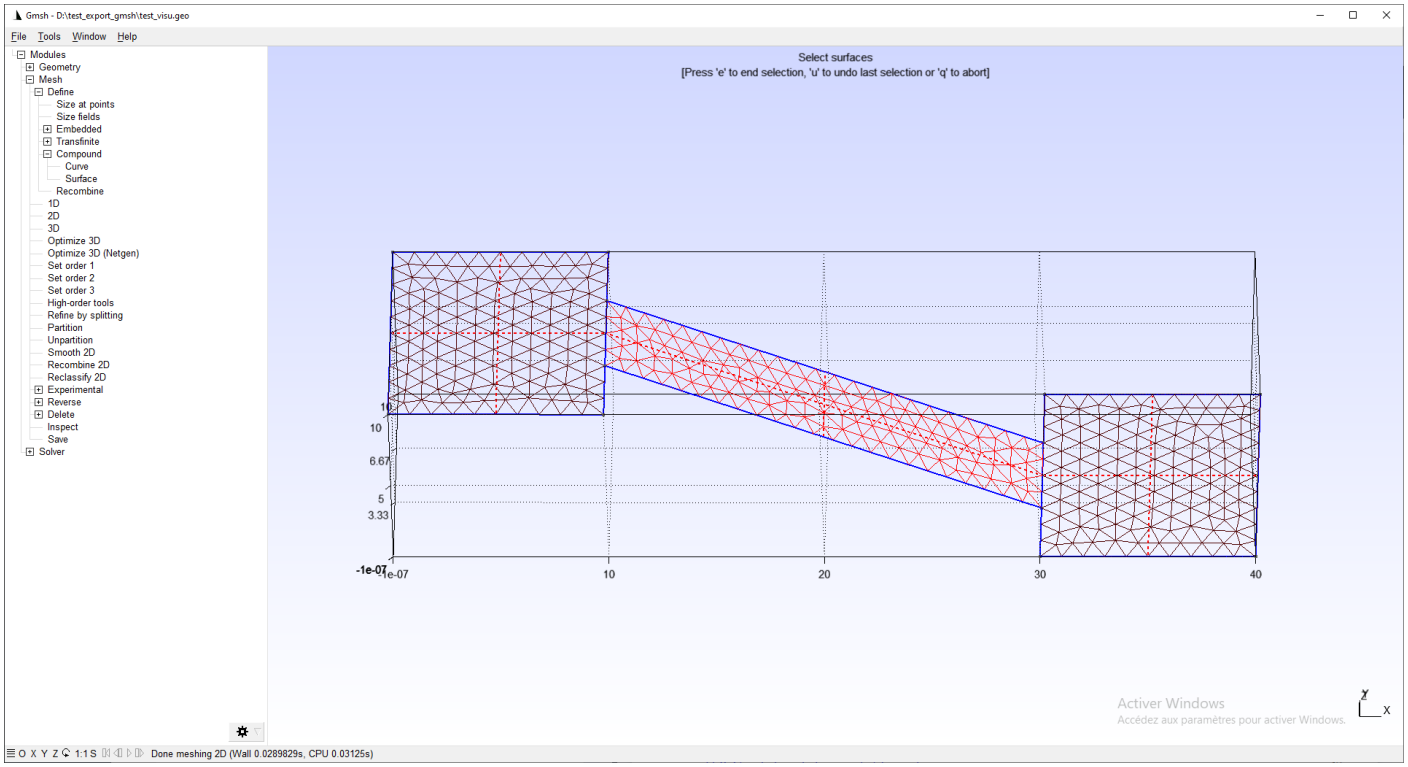


Figure 8: Creating a common surface

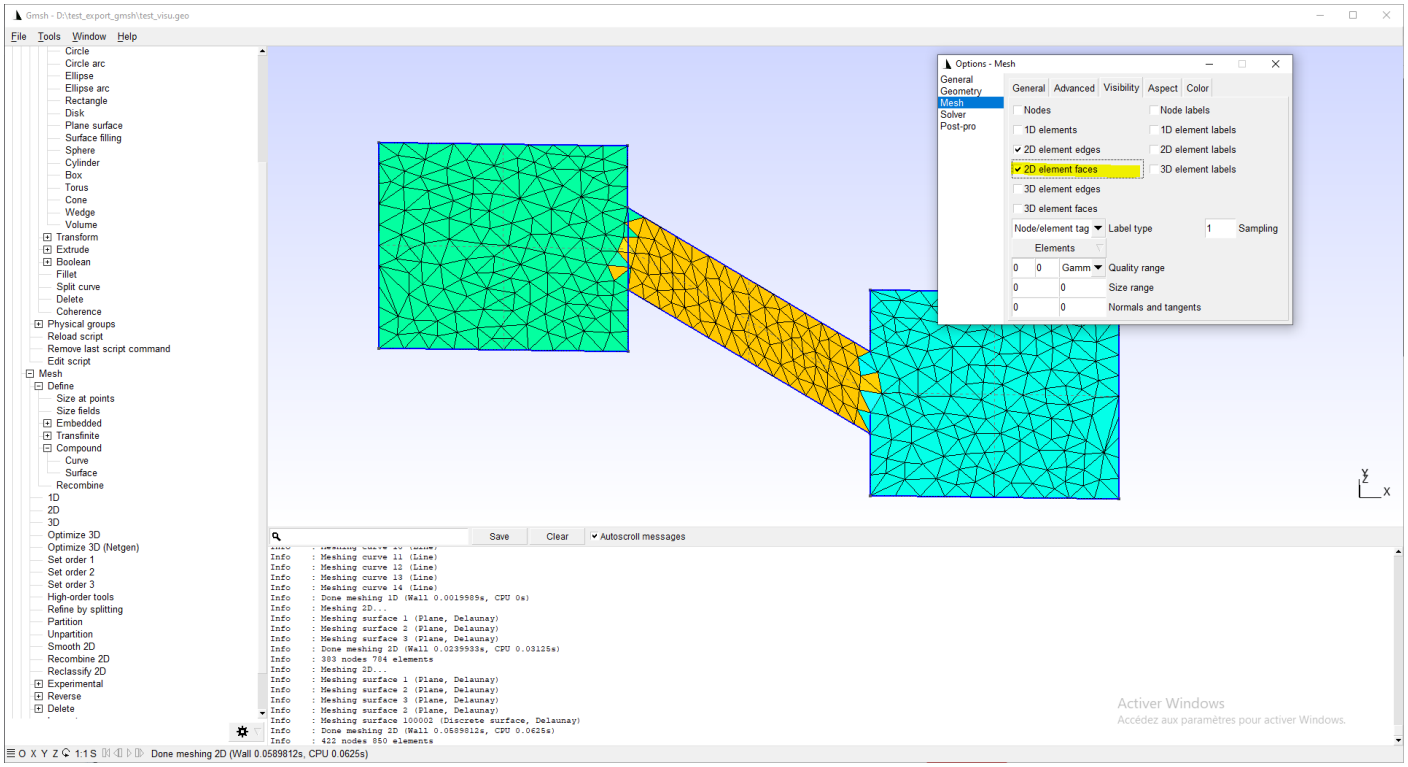


Figure 9: Surface visualization

As you can see, the three surfaces are combined into one, but the surface doesn't pass exactly through the edges of the two floors. To fix this, go to "Mesh->Define->Embeded->Curve" and integrate the interior lines into the surface.

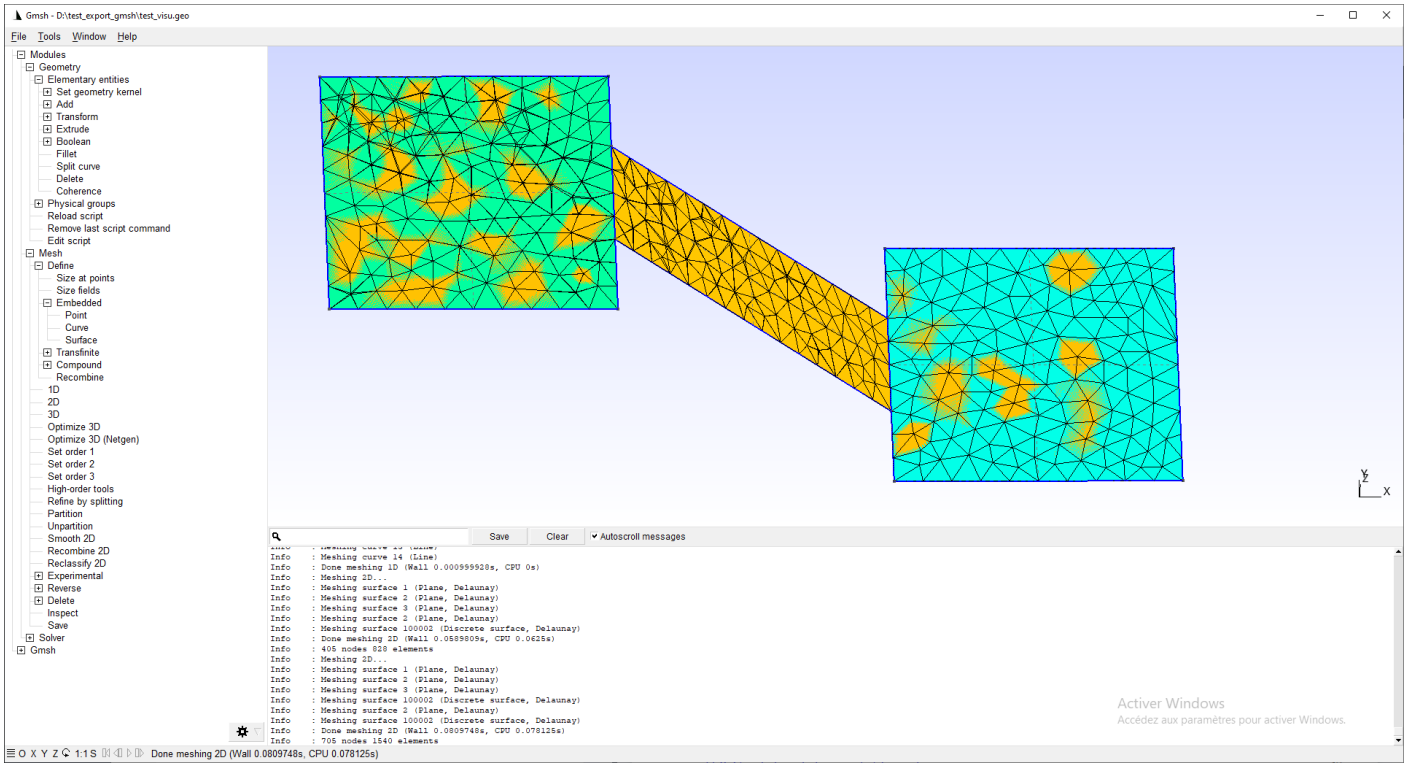


Figure 10: Surface correction to pass through inner lines

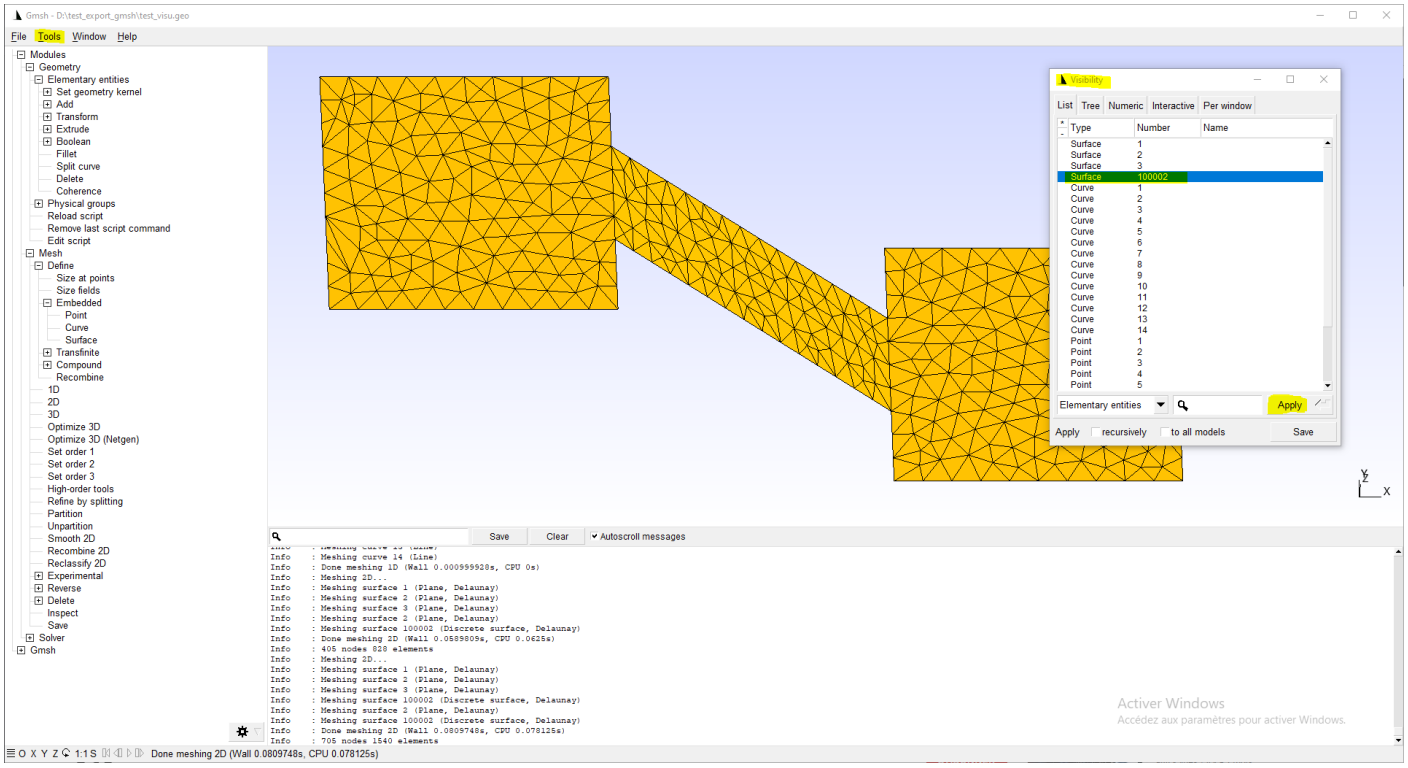


Figure 11: Visualization of the recombined surface only

At this point, the mesh is ready to be exported in CGNS format and read either by the domain decomposition code or by *CuteFlow*.

2.4 Adding a Manning number to each mesh in the mesh file

Most of the time, the Manning number is defined as constant, using the *is_ override_ manning* parameter set to 1. If the user wishes to define different Manning numbers for each mesh, the approach is more complex. A Python code is used to write a Manning number to each mesh in the mesh file. This code can be found in the CuteFlow/src/gen_ manning/

folder. A file Readme.txt briefly explains the procedure for creating a virtual environment, which we detail here.

We create the virtual environment in which we can run the python code as follows:

```
# Creating the virtual environment

module load gcc cgns python/3.8.10
virtualenv ENV
source ENV/bin/activate
pip install --no-index --upgrade pip
pip install openpyxl numpy pandas scipy matplotlib pyCGNS xlrd pyDOE sobol-seq
```

After creating the virtual environment, we update the installed libraries with the upgrade_bib.py code:

```
# Updating installed libraries
python3 upgrade_bib.py
```

```
# To get out of the environment
deactivate

# To re-enter the environment
module load gcc cgns python/3.8.10
source ENV/bin/activate
```

Once the environment has been activated, the code is used as follows:

```
# Create mesh.vtk and mesh.cgns files
python3 gen_manning.py mesh.cgns
```

The *mesh.vtk* file can be opened in Paraview to visualize Manning's numbers. Once the *mesh.cgns* file has been generated, it can be used with *CuteFlow*.

Note: Changing the Manning number can make simulation with *CuteFlow* difficult by causing instabilities. Manning numbers below 0 or above 0.5 should not be used. To reduce instabilities, you can reduce the CFL number or increase the dry/wet tolerance *tolisec*.

In the code gen_manning.py, we specify the indexes (numbers) of the domain meshes with the Manning numbers we wish to assign to them. The mesh numbers are defined in the list ***cell_index*** and the Manning coefficient values in the list ***manning_value***. These data are read into the code from a text file, *Entree_gen_manning.txt*, whose structure and creation procedure are detailed below.
The ***average_manning*** variable is used to define the number of Manning preeminent in the domain, or the number of Manning to assign to cells not identified in the *Entree_gen_manning.txt* file.

2.4.1 Creation of the file "Entree_gen_manning.txt" preliminary to the addition of Manning's numbers

. The *Entree_gen_manning.txt* file contains information on cell numbers and their respective Manning coefficient values. It serves as the input file for the gen_manning.py code, and looks as follows:

```
Cells IDs
71,72,78,80,90,91,92,93,125
Manning_values
0.024,0.024,0.024,0.030,0.030,0.030,0.014,0.014,0.014
```

Figure 12: Structure of file *Entree_gen_manning.txt*

When this information is already available, for example in the case of small study areas, the file can be edited directly using a text editor such as Notepad. In the case of a real mesh, the identifiers of the cells of interest in the domain can be retrieved using Paraview.

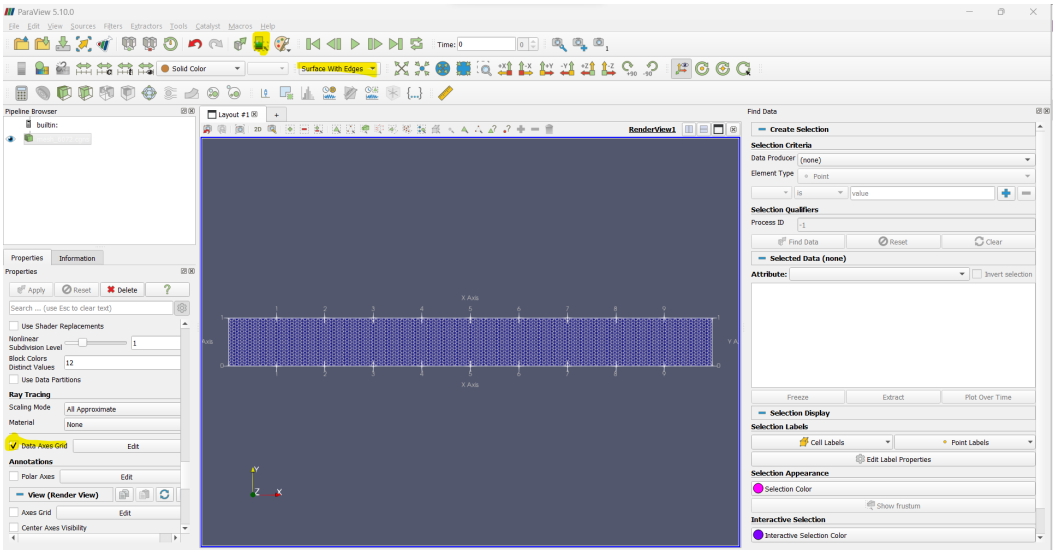


Figure 15: Open the data selection window

The following is an example of how to create the file *Entree_gen_manning.txt* for the mesh mesh_6072.cgns. The aim here is to define 4 zones based on Manning coefficient values.

The first step is to identify the different zones of the mesh and define a Manning number for each of them. The figure below illustrates an example of the Manning mapping we wish to obtain for the mesh under consideration.

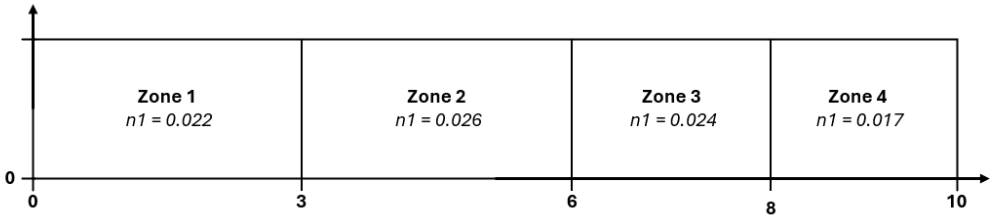


Figure 13: Mapping (zoning) of mesh_6072.cgns according to Manning coefficient values

Next, open the mesh file in paraview, select the cells according to the established mapping and extract the cell identifiers by defining the zones as follows:

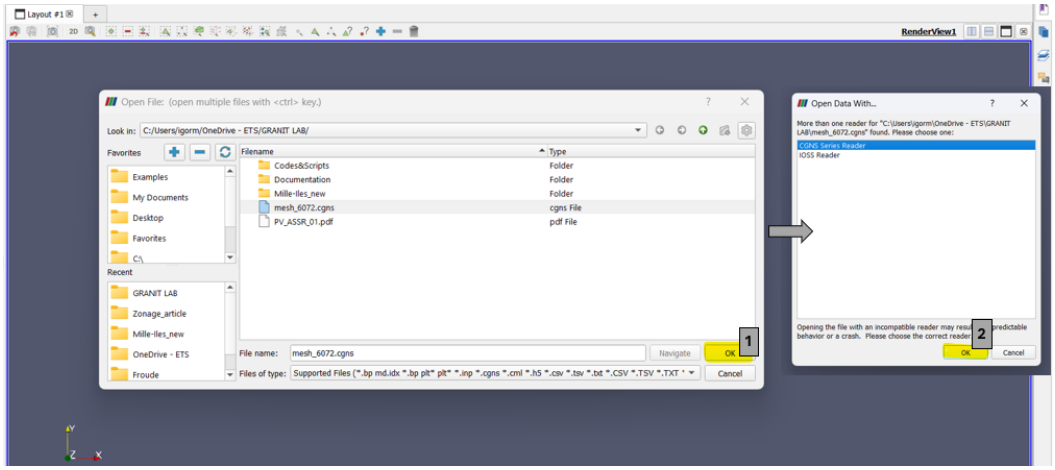


Figure 14: Open the mesh_6072.cgns file and select the file reader (cgns series reader).

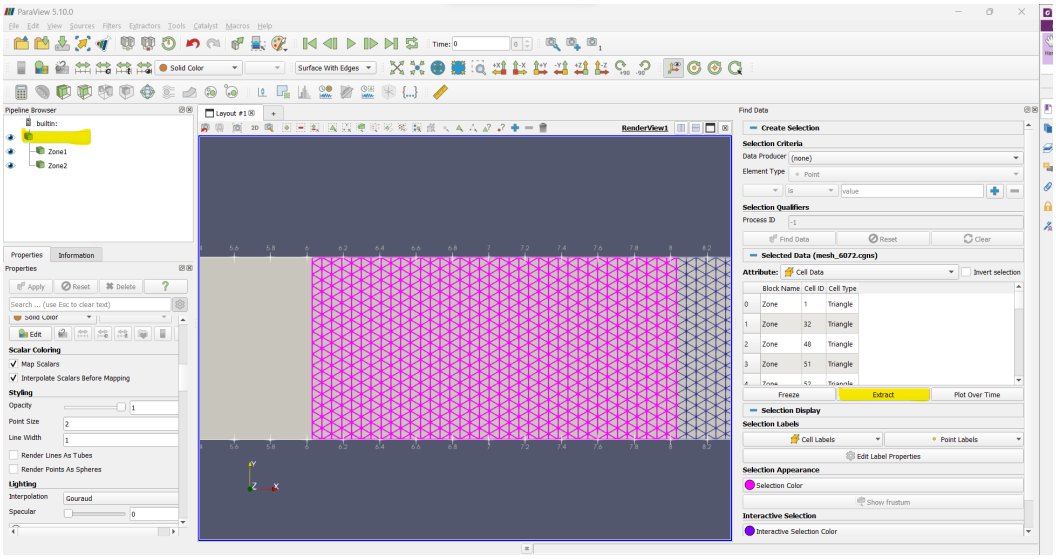


Figure 16: Selecting and extracting mesh cells using the selection tools

After this step, we export the mesh cell IDs to Excel files for each defined zone. To do this, open a new window in SpreadSheetView mode, then export the data as follows:

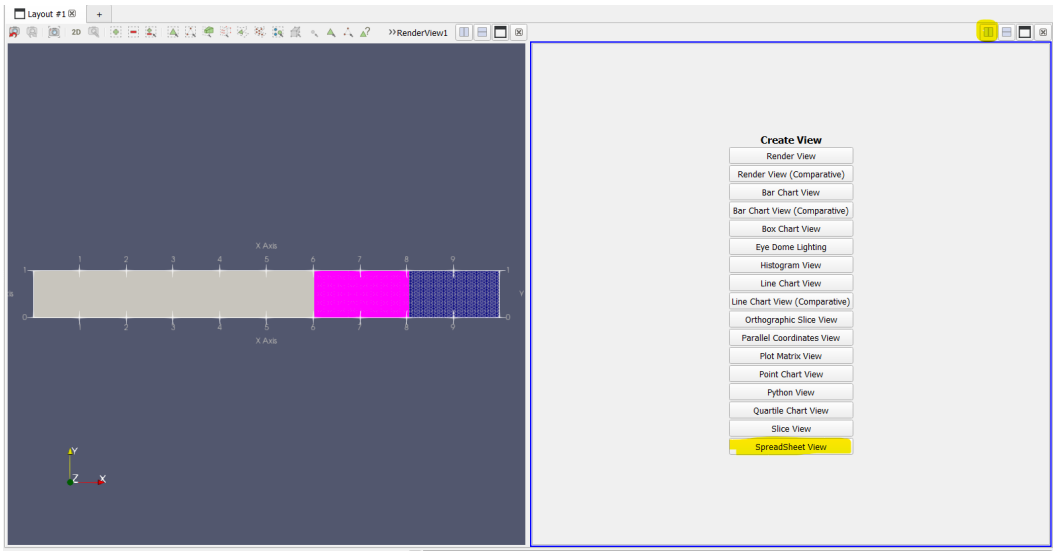


Figure 17: Open window *SpreadSheetView*

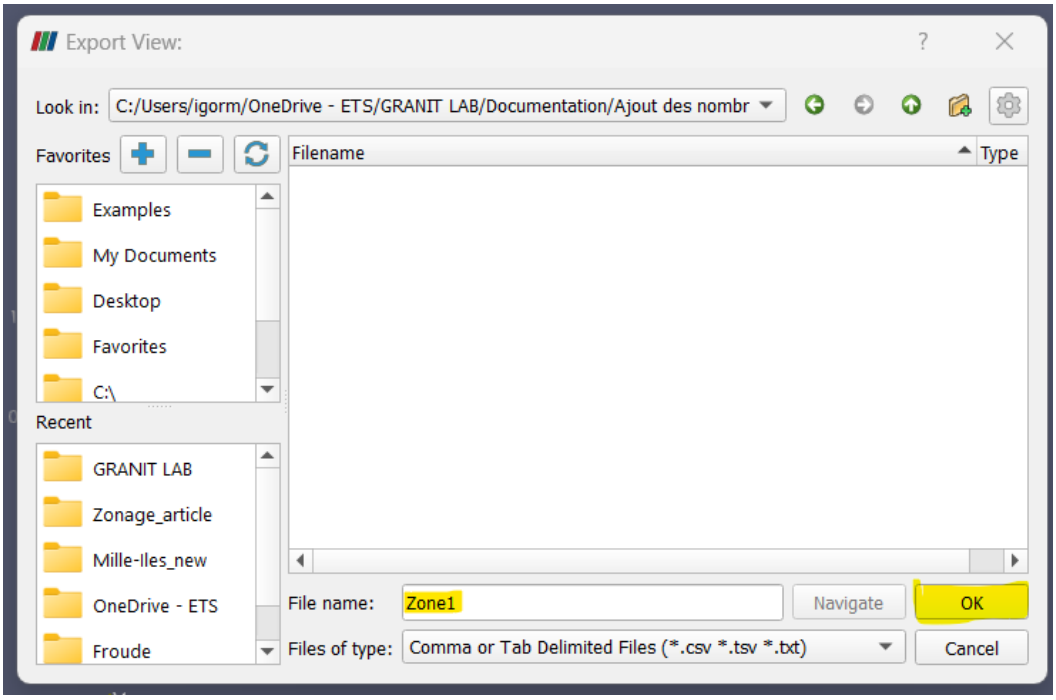
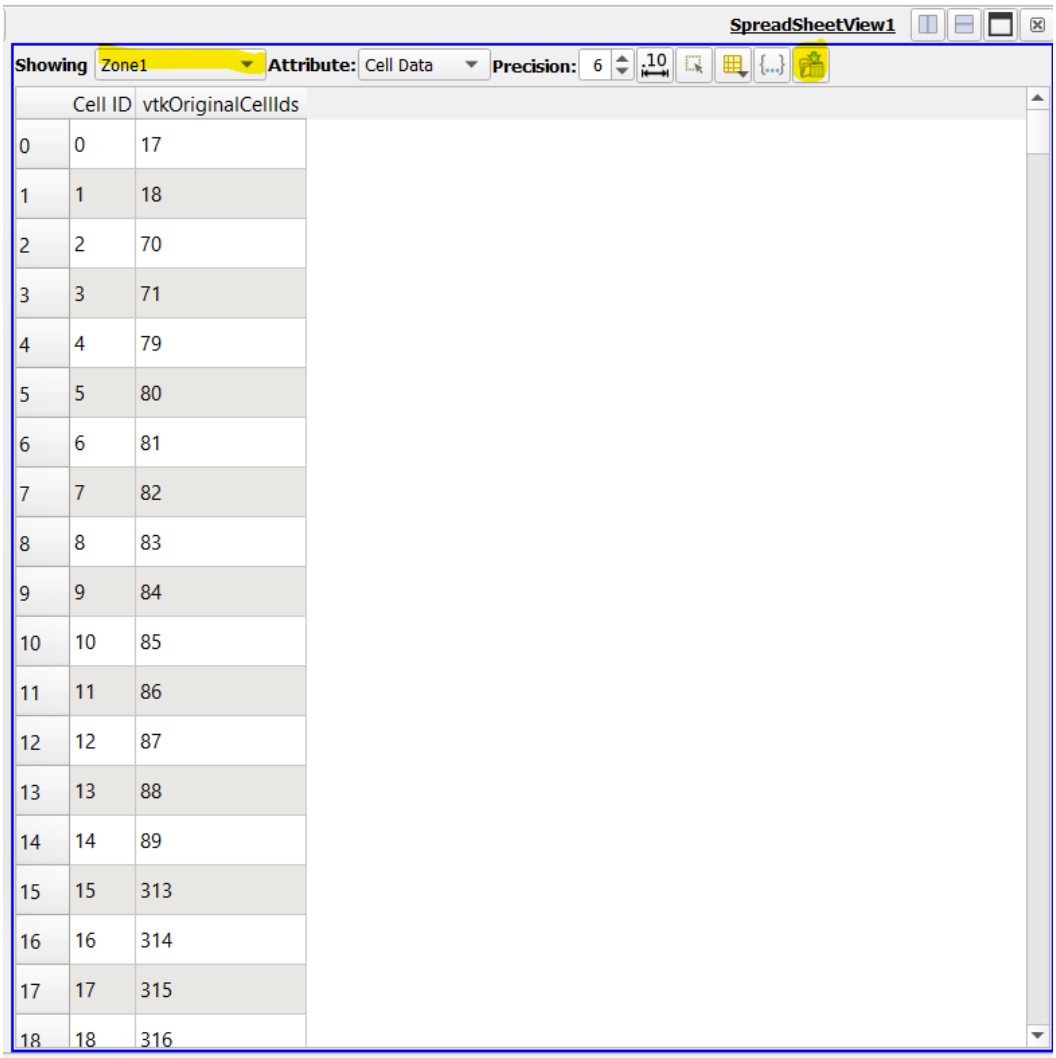


Figure 19: Save the files containing the Cells_IDs for each zone in a folder of your choice. (Example for zone1)



	Cell ID	vtkOriginalCellIds
0	0	17
1	1	18
2	2	70
3	3	71
4	4	79
5	5	80
6	6	81
7	7	82
8	8	83
9	9	84
10	10	85
11	11	86
12	12	87
13	13	88
14	14	89
15	15	313
16	16	314
17	17	315
18	18	316

Figure 18: Select the area and information to be exported, then click on the **export** button.

Once all the zones have been defined and the corresponding Cells_IDs extracted, we assemble this information in a single

Excel file, respecting the following structure:

	A	B	C	D	E	F
1	Zone	Cells IDs				
2	Zone1	3				
3	Zone1	4				
4	Zone1	7				
5	Zone1	8				
6	Zone2	75				
7	Zone2	2				
8	Zone2	68				
9	Zone2	42				
10	Zone3	12				
11	Zone3	34				
12	Zone3	19				
13	Zone3	16				

Figure 20: Data structure in the Zoning Excel file

In the case of mesh_6072.cgns, we've saved this file under the name **"Zones_mesh6072.xlsx"**. Once the zoning file in excel format has been created, we can import it into our Compute Canada cluster workspace and use the code *gen_Entree_gen_manning.py* to generate the text file Entree_gen_manning.txt.

We'll create this example in the CuteFlow/Calibration_manning/Example1 folder.

```
# From the CuteFlow folder
cd Calibration_manning

# Create folder example1
mkdir example1
cd example1

# Download the file Zones_mesh6072.xlsx in the current folder example1
# Copy mesh file to the current folder
cp ../../meshes/rect_mesh_convergence/mesh_6072.cgns .

# From the example1 folder, copy the necessary Python code
cp ../code/Zonage.py .
cp ../code/gen_Entree_gen_manning/gen_Entree_gen_manning.py .

# Create the Manning.txt file containing, in order (as defined in the Excel zoning file), the respective Manning values
    ↪ for the extracted zones.
nano Manning.txt
0.022,0.026,0.024 #n1, n2, n3 respectively

# Once the virtual environment is active, add Manning's values to the zoning Excel file
python3 Zonage.py Zones_mesh6072.xlsx

# Create the Entree_gen_manning.txt file
python3 gen_Entree_gen_manning.py Output_zonage.xlsx

# From the example1 folder
cp ../../src/gen_manning/gen_manning.py .

# Add Manning values to meshes in the mesh file
python3 gen_manning.py mesh_6072.cgns
```

You can check that the mesh has been modified by opening the file *mannings.vtk*:

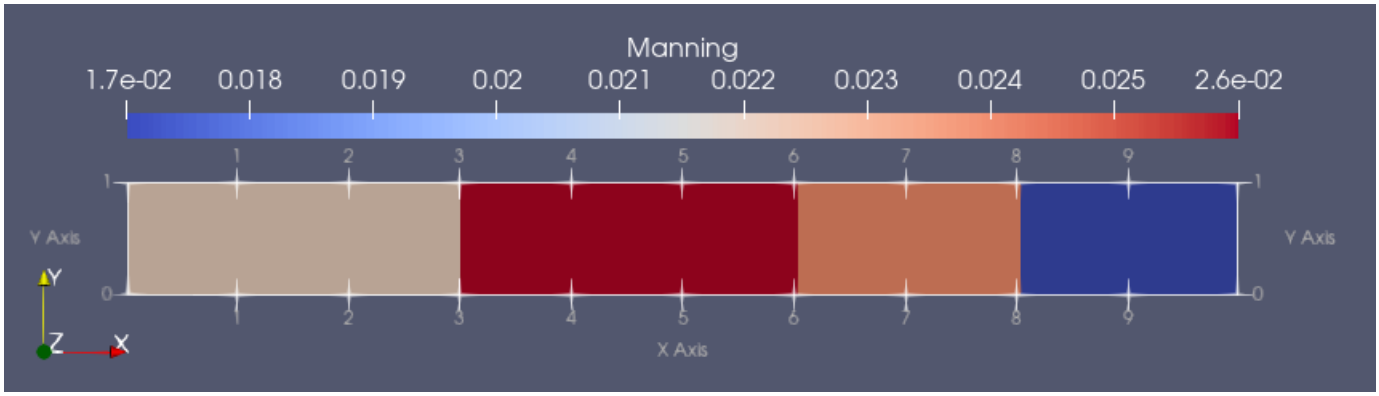


Figure 21: Manning numbers modified on the mesh_6072.cgns file

Note :

- The values exported to the zoning Excel file to build the new mesh, are the vtkOriginalCellIds extracted from Paraview
- For **m** desired manning zones, it's sufficient to select and export the data from the cells on **m-1** zones ; the value of the Manning coefficient for the last zone being defined in the variable **average_manning** of the gen_manning script (average_manning = 0.017 in our example).
- The file **Manning.txt** is essential to the operation of the zoning script **Zonage.py**.

2.5 Breaking down the mesh into sub-domains: parmetis_ghost

To be able to use *CuteFlow* in parallel, it is necessary to decompose the mesh files into as many sub-domains as you want to use GPUs to do the solving. In addition, it is necessary to add one or more layers of ghost meshes to each sub-domain to ensure the quality of the solution across the sub-domains. To this end, a parallel pre-processing code has been developed to decompose a CGNS mesh into several sub-domains and add multiple layers of ghost meshes. [4]. All you need to do is give this program a file in CGNS format, the number of sub-domains you want, and the number of ghost mesh layers you want (one layer for a 1st-order method and two layers for a 2nd-order method). As output, a single file in CGNS format will be generated, containing all the fields and information required for memory exchange in *CuteFlow*.

The code parmetis_ghost is available on GitHub https://github.com/ETS-GRANIT/parmetis-ghostlayers. It can be cloned into Graham's scratch as follows:

```
# On Graham
cd ~/scratch
git clone https://github.com/ETS-GRANIT/parmetis-ghostlayers
```

You can then create a build folder and compile the code there, after loading the right modules as follows,

```
cd parmetis-ghostlayers
mkdir build
cd build

# Loading modules required for compilation
module load StdEnv/2020 gcc/9.3.0 metis/5.1.0 parmetis/4.0.3 cgns/4.1.2

# Creating the makefile with cmake
cmake ..

# Code compilation
make
```

If you want to compile the *parmetis-ghostlayers* code on another system, you'll need to make sure you've installed the *metis*, *parmetis* and *cgns* libraries, with particular support for parallel file writing for the *cgns* library (PCGNS CGNS-Github).

At the end of compilation, an executable file *main* is created. To run this code, you must use MPI with two processors (a prerequisite for using *parmetis*) as follows,

```
mpirun -n NP ./main NSD FICHIER_MAILLAGE.CGNS MODE NC
```

where, NP is the number of processors used (at least 2), FICHIER_MAILLAGE. CGNS is a mesh file in CGNS format containing an area to be decomposed into several sub-areas, NSD is the number of sub-areas required, MODE takes either the value 0 or 1 depending on whether a neighboring mesh is calculated with one side in common (MODE=0 classical choice) or one vertex in common (MODE=1), and, NC is the number of layers of ghost meshes required (1 for a method of order 1 and 2 for a method of order 2). An example of use is therefore,

```
mpirun -n 2 ./main 8 mille.cgns 0 2
```

which will decompose the mille.cgns file into 8 sub-domains with 2 layers of phantom meshes calculated via common sides using 2 processors. Note that NP must be less than or equal to NSD.

We now present an example of the decomposition of a mesh file that will be used in section 3.3. We'll perform this decomposition in the **CuteFlow/examples/example1** folder, using the *main* executable file created earlier.

```
From the Cuteflow folder
cd examples/example1

# Copy of parmetis-ghost executable (decomposition code)
cp ~/scratch/parmetis-ghostlayers/build/main .

# Copy of the mesh file to be decomposed
cp ../../meshes/mille/Mille_Iles_mesh_743968_elts.cgns .

# If the mesh file is large, you will need to request a calculation node.
salloc --time=0:20:0 --ntasks=4 --account=def-soulaima --mem-per-cpu=4000M

# Once the node has been allocated, start domain decomposition
# 4 MPI processes, 4 sub-domains, 2 ghost mesh layers
mpirun -n 4 ./main 4 Mille_Iles_mesh_743968_elts.cgns 0 2

# You can then rename the output file, for example with
mv Mesh_Output_pcgns_ch.cgns Mille_Iles_mesh_743968_elts_4_2.cgns
```

Following this procedure, the file *Mille_Iles_mesh_*743968*_elts_*4*_*2*.cgns* contains the 4-subdomain decomposition with 2 ghost mesh layers of the original mesh file and can be used with the *CuteFlow* code with 4 GPUs (see section 3.3).

3 *CuteFlow*

3.1 Source files and compilation

CuteFlow source files are located in the CuteFlow/src/cuteflow/ folder. A *makefile* is present in the CuteFlow/ folder and can be copied to the CuteFlow/build/ folder for compilation.

Note: To use the multi-CPU version, you need to run "git checkout multi-cpu-new" to get onto the multi-CPU branch of the code, then compile in the same way.

```
# To compile CuteFlow
cd build/
cp ../makefile .
make
```

The executable file *cuteflow* will then be created in the folder CuteFlow/bin/.

The user can modify the compilation options in the *makefile* file, which has just been copied to the Cuteflow/build/ folder.

To compile the *cuteflow* code, you need to specify the target *Compute Capability* of the GPUs you're compiling for. The default *Compute Capability* in the makefile is 6.0, which works for compiling on CEDAR and GRAHAM with NVIDIA P100 GPUs. For BELUGA and the use of NVIDIA V100 GPUs in general, you need to specify a *Compute Capability* of 7.0. This can be achieved by doing,

```
make CC=70
```

If you want to be sure that the code is recompiled, you can force recompilation of all files with the "-B" option, as follows:

```
# For NVIDIA P100 GPUs
make -B CC=70
```

Note: When requesting computing resources, if you don't specify which type of GPU to use, the choice may be made automatically and GPUs that don't correspond to the chosen Compute Capability may be used. It is therefore strongly recommended to specify exactly which type of GPU to use when launching a simulation with *–gres=gpu[[:type]:number]* (see https://docs.alliancecan.ca/wiki/Using_GPUs_with_Slurm).

The makefile file will load the modules needed to compile the code on the *Compute Canada* calculation clusters with the command :

```
module load StdEnv/2020 nvhpc/20.7 cuda/11.0 openmpi/4 cgns/4.1.2
```

To be able to compile the code on another system, you'll need to install and compile the various versions of the *nvhcp*, *cuda*, *openmpi* and *cgns* libraries. In particular, make sure that the *openmpi* library is compiled with CUDA support (CUDA-Aware OpenMPI) and that the CGNS library is compiled for parallel mesh file management (PCGNS-doc CGNS-Github).

List of libraries used to compile *CuteFlow* code in multi-GPU version ("master" branch):

- nvhcp/20.7
- cuda/11.0
- openmpi/4
- cgns/4.1.2

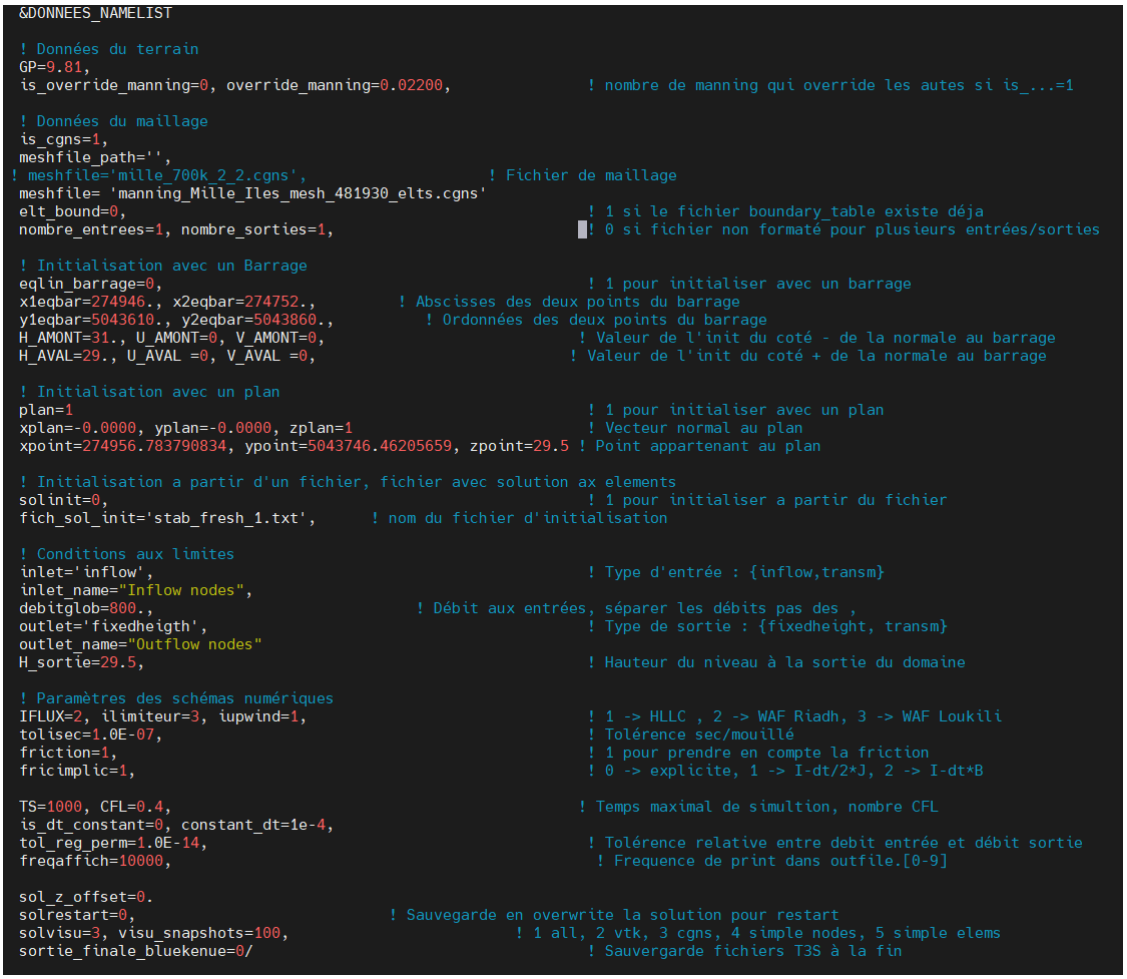
List of libraries used to compile *CuteFlow* code in multi-CPU version ("multi-cpu-new" branch):

- gcc/9.3.0
- openmpi/4.0.3
- cgns/4.1.2

3.2 File description donnees.f

During a simulation, the *CuteFlow* code reads the simulation parameters from the **donnees.f** file in the current folder. This section describes each parameter in this file, which is previewed in Figure 22. An example can be found in the file CuteFlow/src/cuteflow/donnees.f.

1. GP (real): The gravitational constant (9.81 $m.s^-2$)
2. is_override_manning (0/1): Used to force a specific Manning number (friction coefficient) rather than reading that from the mesh files.
3. override_manning (real): Manning number that applies if is_override_manning is 1.
4. is_cgns (0/1): Takes the value 1 if the mesh used is in CGNS format, and the value 0 if we use the old mesh formats.
5. meshfile_path (string): Path to the mesh file. If the mesh file is in the parent folder, meshfile_path="../" .
6. meshfile (string): Name of the mesh file.
7. elt_bound (0/1): Used to avoid having to re-calculate the connectivity of the elements of a mesh file. If a previous simulation has already been launched, and the $*boundary*$ files have been created then elt_bound can take the value of 1 to marginally speed up the launch of the simulations. If we use a mesh broken down into several subdomains, the time saving will be negligible and the value 0 can be systematically adopted without problem.
8. number_entries (integer): Number of mesh entries.
9. number_outputs (integer): Number of outputs from the mesh.
10. eqlin_dam (0/1): Used to specify if the initial condition is a Riemann problem around a line. If the value is 1 then the coordinates of two points on the line will be read in the following parameters.
11. x1eqbar (real): Position x of the first point on the line.
12. y1eqbar (real): Position y of the first point on the line.
13. x2eqbar (real): Position x of the second point on the line.



```
&DONNEES_NAMELIST

! Données du terrain
GP=9.81,
is_override_manning=0, override_manning=0.02200,              ! nombre de manning qui override les autes si is_...=1

! Données du maillage
is_cgns=1,
meshfile_path='',
! meshfile='mille_700k_2_2.cgns',                  ! Fichier de maillage
meshfile= 'manning_Mille_Iles_mesh_481930_elts.cgns'
elt_bound=0,                                                  ! 1 si le fichier boundary_table existe déja
nombre_entrees=1, nombre_sorties=1,                           ! 0 si fichier non formaté pour plusieurs entrées/sorties

! Initialisation avec un Barrage
eqlin_barrage=0,                                              ! 1 pour initialiser avec un barrage
x1eqbar=274946., x2eqbar=274752.,         ! Abscisses des deux points du barrage
y1eqbar=5043610., y2eqbar=5043860.,          ! Ordonnées des deux points du barrage
H_AMONT=31., U_AMONT=0, V_AMONT=0,                             ! Valeur de l'init du coté - de la normale au barrage
H_AVAL=29., U_AVAL =0, V_AVAL =0,                             ! Valeur de l'init du coté + de la normale au barrage

! Initialisation avec un plan
plan=1                                                        ! 1 pour initialiser avec un plan
xplan=-0.0000, yplan=-0.0000, zplan=1                         ! Vecteur normal au plan
xpoint=274956.783790834, ypoint=5043746.46205659, zpoint=29.5 ! Point appartenant au plan

! Initialisation a partir d'un fichier, fichier avec solution ax elements
solinit=0,                                                    ! 1 pour initialiser a partir du fichier
fich_sol_init='stab_fresh_1.txt',      ! nom du fichier d'initialisation

! Conditions aux limites
inlet='inflow',                                               ! Type d'entrée : {inflow,transm}
inlet_name="Inflow nodes",
debitglob=800.,                              ! Débit aux entrées, séparer les débits pas des ,
outlet='fixedheigth',                                         ! Type de sortie : {fixedheight, transm}
outlet_name="Outflow nodes"
H_sortie=29.5,                                                ! Hauteur du niveau à la sortie du domaine

! Paramètres des schémas numériques
IFLUX=2, ilimiteur=3, iupwind=1,                              ! 1 -> HLLC , 2 -> WAF Riadh, 3 -> WAF Loukili
tolisec=1.0E-07,                                              ! Tolérence sec/mouillé
friction=1,                                                   ! 1 pour prendre en compte la friction
fricimplic=1,                                                 ! 0 -> explicite, 1 -> I-dt/2*J, 2 -> I-dt*B

TS=1000, CFL=0.4,                                            ! Temps maximal de simultion, nombre CFL
is_dt_constant=0, constant_dt=1e-4,
tol_reg_perm=1.0E-14,                                         ! Tolérence relative entre debit entrée et débit sortie
freqaffich=10000,                                             ! Frequence de print dans outfile.[0-9]

sol_z_offset=0.
solrestart=0,                           ! Sauvegarde en overwrite la solution pour restart
solvisu=3, visu_snapshots=100,                        ! 1 all, 2 vtk, 3 cgns, 4 simple nodes, 5 simple elems
sortie_finale_bluekenue=0/                                    ! Sauvergarde fichiers T3S à la fin
```

Figure 22: Preview of the **donnees.f** file edited with nano

14. y2eqbar (real): Position y of the second point on the line.
15. H_UPstream (real): Height of the **free surface** upstream of the right.
16. H_AVAL (real): Height of the **free surface** downstream of the right.
17. U_UPstream (real): Speed according to x upstream of the right.
18. U_AVAL (real): Speed according to x downstream of the right.
19. V_UPstream (real): Speed according to y upstream of the right.
20. V_AVAL (real): Speed according to y downstream of the right.
21. plan (0/1): Used to specify the initialization of the solution according to a plan. The coordinates of a point belonging to the plane and of a vector orthogonal to the plane are read in the following. If $xplan = 0.$, $yplan = 0.$, $zplan = 1.$, $xpoint = 0.$, $ypoint = 0.$, and, $zpoint = 10.$, we have an orthogonal plane to the axis z which passes through the point $(0, 0, 10)$.
22. xplan (real): Projection according to x of a vector orthogonal to the plan
23. yplan (real): Projection according to y of a vector orthogonal to the plane
24. zplan (real): Projection according to z of a vector orthogonal to the plane
25. xpoint (real): Coordinate x of a point on the plane.
26. ypoint (real): Coordinate y of a point on the plane.
27. zpoint (real): Coordinate z of a point on the plane.
28. solinit (0/1): Used to specify whether to initialize from an already generated solution file.
29. file_sol_init (string): Name of the pre-existing solution file.
30. inlet (inflow/transm): Specifies the type of inlet boundary condition. $inflow$ for an incoming flow, $transm$ for a transmissive condition.
31. inlet_name (string): Name of input boundary conditions separated by commas. Used with CGNS files when multiple entries are present so that each process is aware of all entries that exist. (In the Montreal archipelago mesh files, the name of the entries is simply $InflownodesX$ or X is a number from 1 to 7, which gives, $inlet\_name =$ "$Inflownodes0$","$Inflownodes1$","$Inflownodes2$","$Inflownodes3$","$Inflownodes4$","$Inflownodes5$","$Inflownodes6$
32. debiglob (real): Debits of domain entries separated by commas.
33. outlet(fixedheight/transm): Specifies the type of outlet boundary condition. $fixedheight$ for a fixed output height, $transm$ for a transmissive condition.
34. outlet_name (string): Name of the outlet boundary conditions separated by commas.
35. H_output (real): Heights of the **free surface** at the outputs of the domain separated by commas (not tested for several outputs).
36. IFLUX (1/2/3/4): Choice of digital scheme used. 1 for classic HLLC, 2 for HLLC riadh style (best for dry/wet management), 3 for WAF Loukili style, 4 MUSCL Toro style. For choices 3 and 4, 2 layers of ghost meshes are necessary when decomposing into subdomains.
37. illimiter (1/2/3/4): Choice of limiter in the WAF and MUSCL methods. 1 Superbee style, 2 Leer Style, 3 Minmod-/Albada Style. Limiter 3 is often the one which is the most stable, limiter 1 is the one which gives the best results on simple test cases but (like limiter 2) it can create speeds on complex bathymetries (hydrostatic reconstruction of order 2 necessary).
38. iupwind (1/2): Way to search for upwind meshes for order 2. Must take the value 1 to have correct results, the value 2 is a test in progress which does not give stable results.
39. tolisec (real): Dry/wet tolerance value. A value between $10e-4$ and $10e-8$ is usual. A value that is too large or too small can cause code discrepancies.
40. friction (0/1): Activates or not the management of friction in the code (via Manning).
41. fircimplicit (0/1): Activates the semi-implicitness of friction terms. Leave at 1.
42. TS (real): Final time of the simulation.
43. CFL (real): CFL number between 0 and 1. The closer the value is to 1, the more discrepancies may appear. (The characteristic distance is 1.8 times the value of the radius of the circle inscribed in the smallest mesh of the mesh).
44. is_dt_constant (0/1): Used to set a fixed value of the time step (Not recommended).

45. constant_dt (real): If *is_dt_constant* is 1, specifies the value of the time step for the simulation.
46. tol_reg_perm (real): Output tolerance linked to steady state. If the relative difference between the output flow and the input flow is smaller than the tolerance for a certain time, then the simulation will stop. A value of $1e-15$ allows this exit condition to never be reached.
47. freqdisplay (integer): Number of iterations between two displays of the simulation state in the output file.
48. sol_z_offset (real): Formerly used to artificially enhance code output solutions. A value of 0 allows you to have no impact.
49. solrestart (0/1): If a value of 1 is adopted, the code will generate a final solution from which it will be possible to restart.
50. solvisu (0/1/2/3/4/5): Specifies the output file type. 0 for no output, 1 for all outputs, 2 for output in vtk format, 3 for output in cgns format, 4 for solution output on each node of the mesh, 5 for solution on each element of the mesh (at from which we can restart).
51. output_finale_bluekenue (0/1): Output in T3S format readable by Bluekenue.

3.3 Launching a simulation

In this section, we present how to launch a simulation from the **exemples/example2** folder. This simulation will use 4 GPUs and will be performed on the mesh files that the user has already decomposed in the **exemples/example1** folder (see section 2.5).

```
# From the Cuteflow folder
cd examples/example2

# Copying the mesh file to the current folder
cp ../example1/Mille_Iles_mesh_48930_elts_4_2.txt .

# Copy the data file to the current folder
cp ../../src/cuteflow/donnees.f .

# Copy of the CUTEFLOW executable to the current folder
cp ../../bin/cuteflow .

# Copy the code launch file to the current folder
cp ../../scripts/4_gpu.sh .
```

At this point, the output of the *ls* command in the **example2** folder should be as follows:

```
Mille_Iles_mesh_481930_elts_4_2.txt
4_gpu.sh
donnees.f
cuteflow
```

The user must then modify the simulation parameters in the **donnees.f** file by referring to the section 3.2.

There are then two ways to launch a simulation: by launching a deferred job or by requesting an interactive node. We can request an interactive node in the following way,

```
salloc --time=0-01:00 --nodes=2 --ntasks-per-node=2 --gres=gpu:p100:2 --mem-per-cpu=4000M --account=rrg-soulaima-ac
```

With this command, we request two calculation nodes with 2 CPUs and 2 GPUs (P100) for 1 hour. Here, we specify as group *rrg-soulaima-ac* only when we are on GRAHAM to use the resource allocation that was given to the research group. We can also use the group *def-soulaima* but we will have a lower priority and a lower quantity of allocated resources.

We may have to wait a little while for the compute node to be allocated to us. Once the calculation node has been allocated to us, we can launch the simulation with the following commands,

```
# Launching the simulation
mpirun --mca pml ob1 -n 4 ./cuteflow
```

This way, all MPI processes will output directly to the terminal, this is often not what we want, we can then replace the last command with:

```
mpirun --mca pml ob1 -n 4 sh -c './cuteflow > outfile.$OMPI_COMM_WORLD_RANK'
```

This way, no process will output to the console, they will each output to the **outfile.X** files with X their process number. You can monitor the progress of the simulation in one of these files with the command:

```
tail -f outfile.0
```

which will allow you to see all the additions that are made to the files interactively.

If we request an interactive node, we should see nothing in the terminal once the command

```
mpirun --mca pml ob1 -n 4 sh -c './cuteflow > outfile.$OMPI_COMM_WORLD_RANK'
```

launched, if errors are present, they will be seen in the *outfile.X* files.

Once the simulation is finished, you can look at the file **outfile.0** for example with a text editor to make sure that everything went well. In this case, with the data file shown above, we obtain for the file **outfile.0** in the figure 23.

Alternatively, you can submit the code to the scheduler, after modifying the **4_gpu.sh** file to match your needs, with the following command:

```
# While in the examples/example2/ folder
sbatch 4_gpu.sh
```

We can see if the code is launched or when it will be launched with the command

```
squeue -u $USER
```

In the same way as before, we can follow the progress by monitoring the outfile.[0-9] files which serve as standard output for each MPI process.

Once the simulation is completed, depending on the parameters of the *donnees.f* file, several result files will be generated and can be viewed using Paraview (see section 4).

3.3.1 Launching a simulation from an already generated solution

A simulation can be restarted from a solution saved on the elements. Simply set the *solrestart* parameter to 1 in the *data.f* file to trigger the saving of the solution in the appropriate files at the end of the simulation. A solution file **_solution_elements_restart.txt* will be saved for each subdomain and will be used to restart the simulation.

Once the **_solution_elements_restart.txt* files have been generated, if we want to restart the simulation on these files we will have to rename them then put their base name as a parameter in $file\_sol\_init$ in the file *donnees.f*.

In the case of example 2, we can for example put $solrestart = 1$ in the file *donnees.f*. Once the first simulation is finished, the **_solution_elements_restart.txt* files will be generated. If we want to be able to re-save the solution on the elements again, we must rename the generated files, for example by doing

```
mv 0_solution_elements_restart.txt 0_sol_example2.txt
mv 1_solution_elements_restart.txt 1_sol_example2.txt
mv 2_solution_elements_restart.txt 2_sol_example2.txt
mv 3_solution_elements_restart.txt 3_sol_example2.txt
```

It will then be necessary to set the parameter $solinit = 1$ and $fich\_sol\_init = ' sol\_example2.txt'$ to initialize the solution with these files.

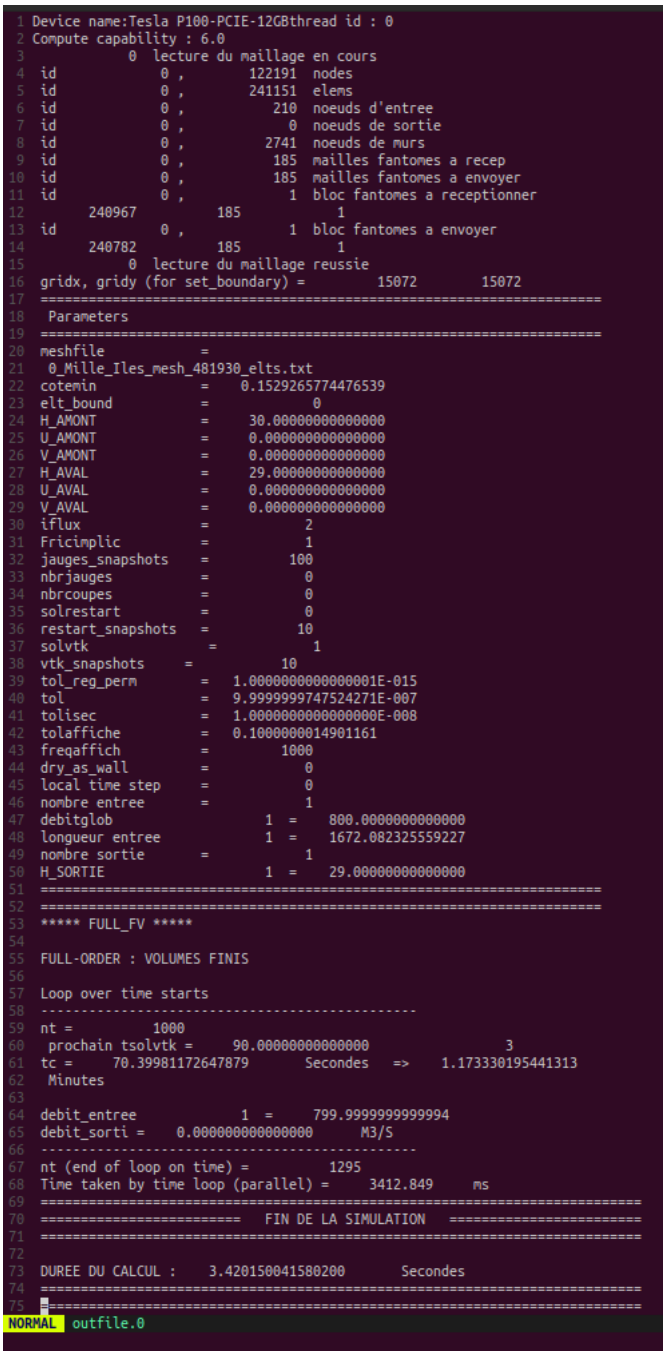


Figure 23: Example 2 : file **outfile.0**

3.4 Launching several simulations simultaneously

This section presents how you can run several simulations simultaneously using *job_arrays*. The objective here is to have a bash script which will create a folder for each simulation that we want to launch by copying the corresponding parameter file there. We can then use a *job_arrays* to launch all the simulations simultaneously.

In the rest of the section, we take as an example the launch of several simulations in the folder **examples/example3** using the mesh files which have been divided into section 2.5.

3.4.1 Generation of data files

We need a way to generate data files automatically from debitglob, hamont, haval and Manning values. This is the purpose of the *gen_donnees* script which is used inside the *multi* script. We don't normally have to use it alone, but we still describe its simple use.

To generate a data file we do,

```
# For debit_global=1000 hamont=31 haval=30 manning=0.04
./gen_data 1000 31 30 0.04
```

The data file *donnees.f* will then be created and it will contain the parameters passed as arguments. Please note, in the current configuration, the value of *haval* will also be used as *houtput*. This script can be found in the folder scripts and the user will surely have to modify it for the use that he wants to do it. This script will be called by the script *multi.sh* with each line of the input file as an argument, we explain this below.

3.4.2 Preparing the base_files

folder Before running the *multi.sh* script, you must prepare the files that the code will need. To do this, you must create a base_files folder in which you will put these files. The necessary files are: the mesh file, the *cuteflow* executable. We will then proceed as follows:

```
# From the CUTEFLOW_CUDA_MPI folder
cd examples/example3

# Copy scripts to current folder
cp ../../scripts/multi.sh .
cp ../../scripts/gen_donnees.sh .

# Creation of the base_files folder and copy of the mesh files
mkdir base_files
cp ../example1/Mille_Iles_mesh_743968_elts_4_2.cgns base_files/
cp ../../bin/cuteflow base_files/
```

Following these operations, the result of the *tree* command launched in the **examples/example3** folder should be consistent with figure 24. (Note that the folder **example3** should already contain the file *INPUT_MONTE_CARLO.dat*)

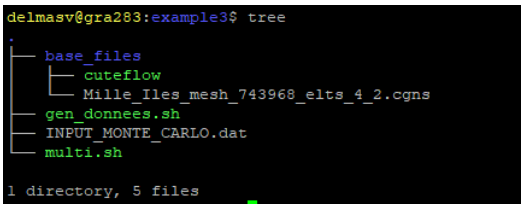


Figure 24: Output of the *tree* command

3.4.3 Launching the script *multi.sh*

The *multi.sh* script reads an input file and creates for each line (for each simulation case) a multi_*[0-9] folder. It copies the files from the base_files folder into this folder and generates the data files using the *gen_donnees* script. The file *INPUT_MONTE_CARLO.dat* serves as an input file to launch several simulations in the folder **examples/example3**.

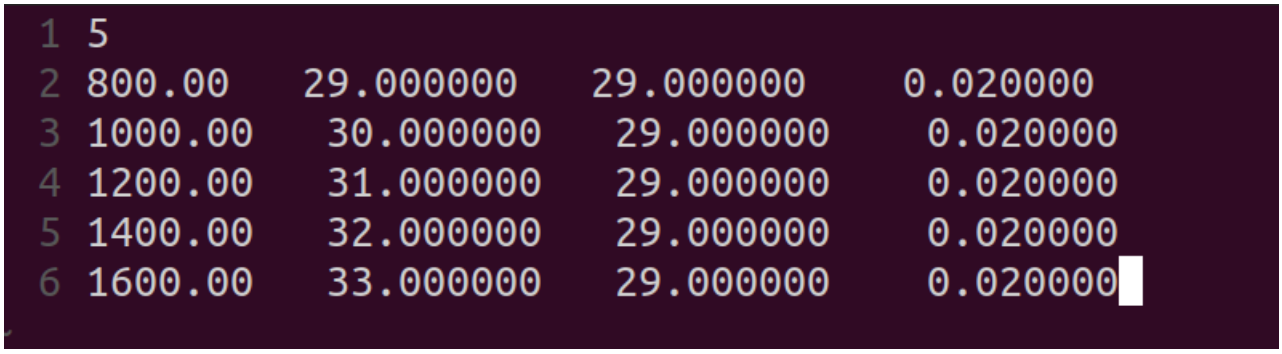


Figure 25: File *INPUT_MONTE_CARLO.dat*

We see in this file that 5 simulations will be launched, to understand what each column corresponds to, we must go to the script *gen_donnees.sh* which generates the file *donnees.f from each line* corresponding. In this example, the first column corresponds to the inlet flow rate, the second to the upstream head, the third to the downstream head and the outlet head, and the last corresponds to the overall Manning number in the domain.

In the **examples/example3** folder, you can run this script with the command

```
./multi.sh INPUT_MONTE_CARLO.dat
```

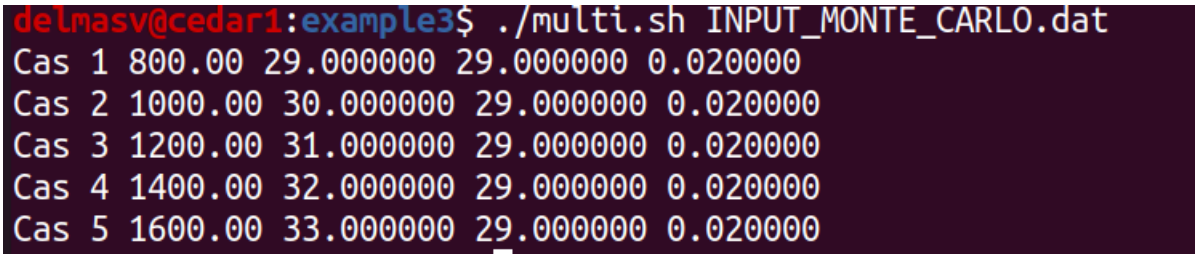


Figure 26: Launching of script *multi.sh*

The different multi_*[0-9] folders will be created one after the other. We can then check the simulation parameters, for example in multi_1. If a parameter needs to be changed, for example the final simulation time, rather than changing it by hand in each of the simulation folders, simply change it in the script *gen_donnees.sh* then restart the script *multi.sh* as indicated previously. This way the modification will apply in all folders.

We can check the correct structure of the folder **examples/example3** following the execution of the script *multi.sh* by executing the command *tree* in this folder, the output should be identical to the figure 27.

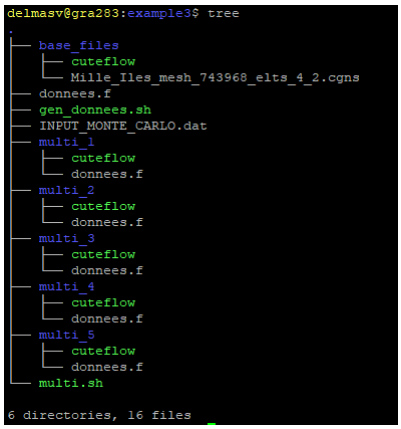


Figure 27: *tree* once the script *multi.sh* is run.

3.4.4 Launching the job array

In the previous part, we created the folder structure to launch all the simulations specified in the file *INPUT_MONTE_CARLO.d* we must now launch all these simulations on the calculation cluster. For this purpose, we use a job array. A job array is a quick way to launch several jobs with the same parameters, the description is given on the Compute Canada wiki https://docs.computecanada.ca/wiki/Job_arrays. The file that we will use in this case is array.sh present in the folder script.

Figure 28: File *array.sh*

Alternatively we can use array_group.sh or the group name has been changed to use allocation of resources that was given to the research group.

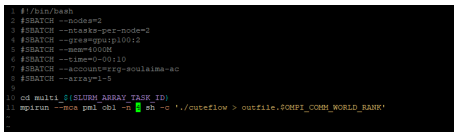


Figure 29: File *array_cedar.sh*

This file describes a job that uses a compute node with 4 GPUs for 10 minutes. An important parameter is the array parameter which describes the number of jobs to launch. In this example, we launch 5 jobs which will be numbered from 1 to 5. We use this number which will be stored in the environment variable SLURM_ARRAY_TASK_ID to move to the appropriate folder before launching the code. To submit this job array to the scheduler, simply do

```
sbatch array.sh
```

To launch the 5 simulations of the folder **examples/example3** you must therefore do,

```
# From the CUTEFLOW_CUDA_MPI folder
cd examples/example3

# Copying the array.sh script
cp ../../scripts/array.sh .

# Launch of the job array
sbatch array.sh
```

The 5 simulations will then be launched on the calculation cluster. You can find out if the simulations are successfully launched with the command *squeue -u $USER*

```
delmasv@cedar1:example3$ sbatch array_cedar.sh
Submitted batch job 49026346
delmasv@cedar1:example3$ squeue -u $USER
             JOBID     USER      ACCOUNT           NAME  ST  TIME_LEFT NODES CPUS TRES_PER_N MIN_MEM NODELIST (REASON)
     49026346_1  delmasv rrg-soulaima array_cedar.sh   R       9:49     1    2 gpu:p100:2   4000M cdr346 (None)
     49026346_2  delmasv rrg-soulaima array_cedar.sh   R       9:49     1    2 gpu:p100:2   4000M cdr385 (None)
     49026346_3  delmasv rrg-soulaima array_cedar.sh   R       9:49     1    2 gpu:p100:2   4000M cdr249 (None)
     49026346_4  delmasv rrg-soulaima array_cedar.sh   R       9:49     1    2 gpu:p100:2   4000M cdr252 (None)
     49026346_5  delmasv rrg-soulaima array_cedar.sh   R       9:49     1    2 gpu:p100:2   4000M cdr258 (None)
delmasv@cedar1:example3$
```

Figure 30: Command *squeue -u $USER*

In the case of the figure 30 the 5 simulations have all been launched and there remains 9 min 49 s of maximum time for their execution. If in the column **ST** (Status) it is indicated **PD** (Pending) it means that the simulations have not yet started and you simply have to wait.

4 Processing of *.vtk and *.cgns files in Paraview

The Paraview software [3] is installed by default on Compute Canada computing clusters and allows client/server use, which makes it possible to visualize very large solutions without having to download the data to your personal computer. A detailed tutorial is available on the official website https://www.paraview.org/Wiki/The_ParaView_Tutorial and use in client/server mode on Compute Canada computing clusters is available on their official wiki url-https://docs.computecanada.ca/wiki/ParaView. If the user prefers to download the solution files locally, they can skip the 4.1 section.

4.1 SSH tunnel for visualization with Paraview on compute clusters using MobaXterm

To do remote viewing with Paraview, you will need to launch a Paraview server on a computing node in the cluster and connect to it from your computer. To connect to it the only way to proceed is to use an SSH tunnel to forward the local port 11111 to the port 11111 of the computing node on which the Paraview server was launched. This way, by connecting Paraview to port 11111 of your local machine, you will connect to the desired computing node.

Before making an SSH tunnel, you must request a computing node on the cluster and launch a Paraview server on it. To do this, two scripts are present in the **scripts**/ folder. The first *request_visu.sh* simply allows you to make a command that requests a computing node with a certain number of CPUs and memory, here 16 CPUs with 12 GB of memory. The script *load_para_cpu.sh* then allows you to launch a Paraview server using the 16 CPUs requested in the previous step (you can request more CPU or memory if necessary).

```
# From folder CUTEFLOW_CUDA_MPI

cd scripts/
./request_visu.sh 2
```

Une fois le nœud de calcul attribué, faire

```
# From folder CUTEFLOW_CUDA_MPI/scripts/

./load_para_cpu.sh
```

This may take a little time but we should have a display close to the following figure,

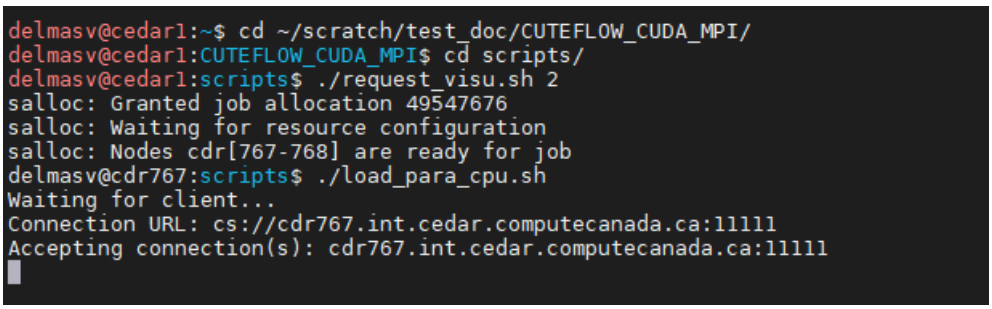


Figure 31: Launching the Paraview server on a compute node

Once arrived at the point in the figure 31 the Paraview server is launched on a computing node of the cluster. The important thing here is to identify the computing node on which the server is launched. We see in the image that the calculation node is called *cdr767*, it is indicated in the right part of the variable PS1 (to the right of the @) and it is also the first part of the URL of connections that we are not going to use in the following. On CEDAR the nodes are called with *cdr* as a prefix, on BELUGA it is *blg* and on GRAHAM it is *gra*. Once the computing node has been located, we can move on to the next step which involves creating the SSH tunnel between our computer and this node on the computing cluster. To do this, we use MobaXterm. MobaXterm offers a visual which shows very well what is happening (see Figure 32).

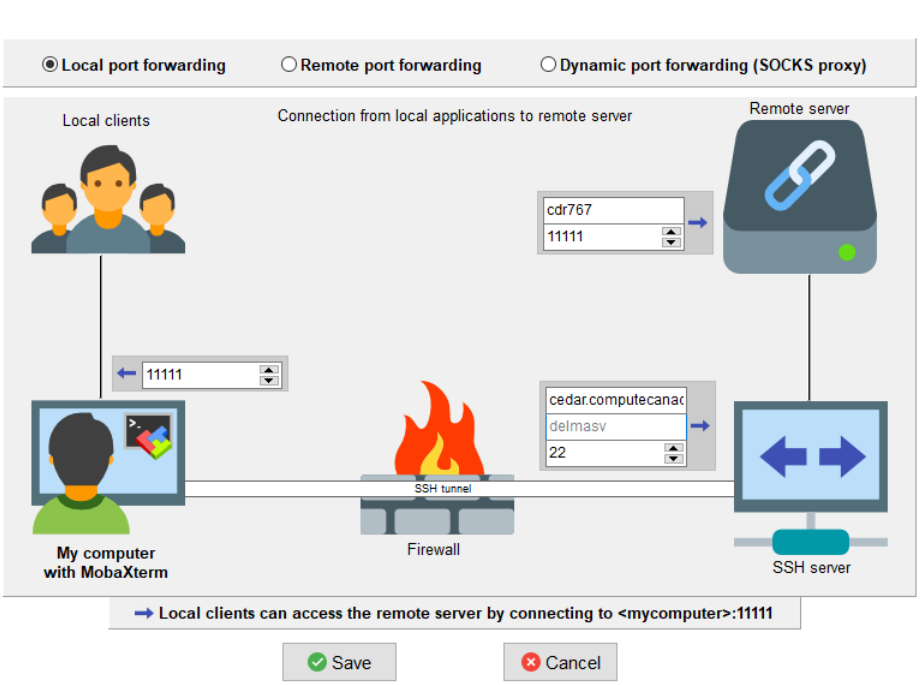


Figure 32: MobaXterm SSH tunnel for Paraview

In this figure, we see that we connect port 11111 of our computer via the SSH server *cedar.computecanada.ca* to port 11111 of the calculation node *cdr767*. You must configure this window as shown in the figure by simply changing the name of the computing node on which the Paraview server was launched. Once these parameters have been entered, simply launch the tunnel by clicking on the read sign in MobaXterm (see following figure).

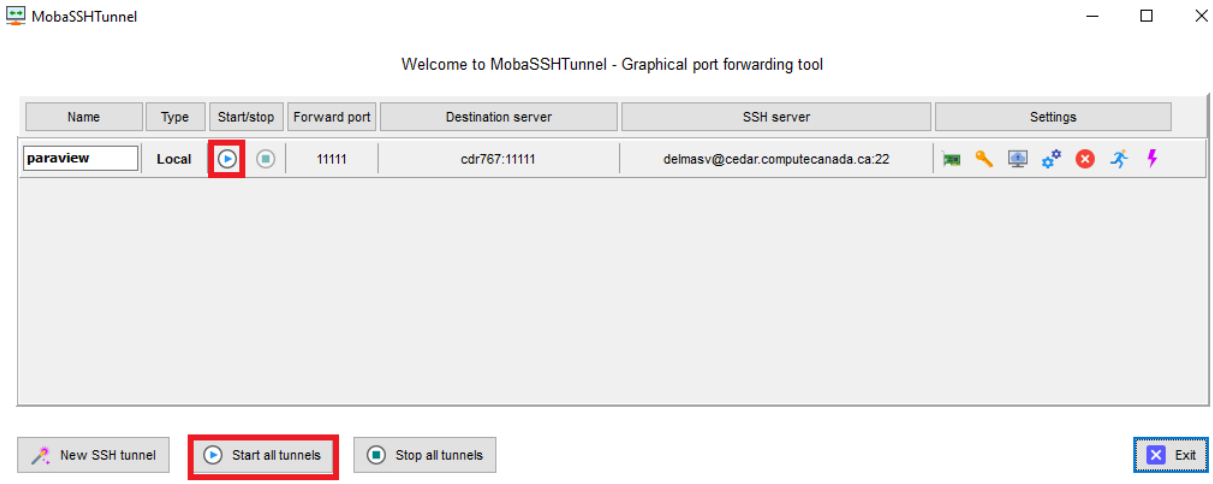


Figure 33: Launching the SSH tunnel in MobaXterm

Once the SSH tunnel has been launched, simply launch Paraview on your computer and connect to port 11111 of your local computer. However, you must make sure to use the same version of Paraview as that present on the calculation clusters (loaded by the script *load_para_cpu.sh*), which is version 5.5.2 of Paraview, downloadable for free. at https://www.paraview.org/download/. Once Paraview is launched, if it is the first time you must create the connection to a server as follows:

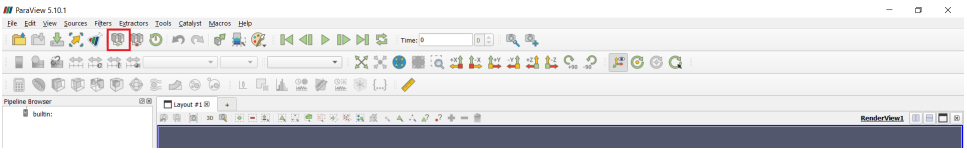


Figure 34: Selecting the menu *connect*

You must then navigate through the windows by following these steps:

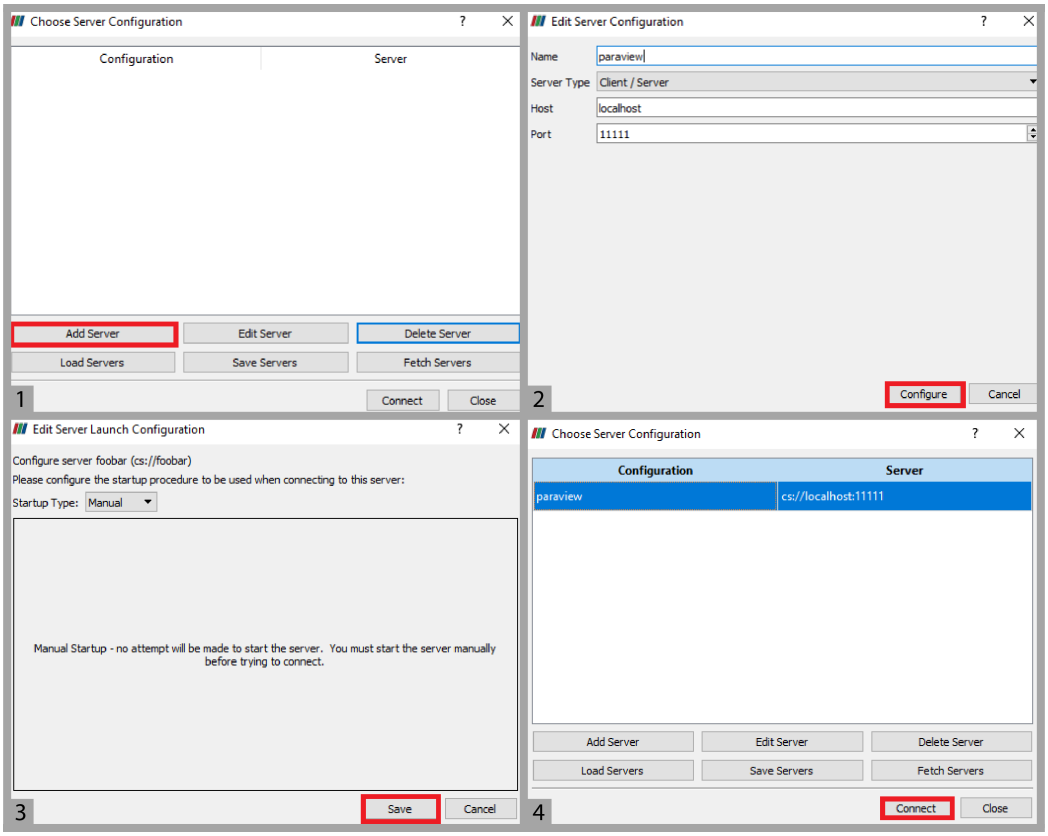


Figure 35: Creating the connection in Paraview

At this point the configuration is finished, simply click on connect to connect to your local port 11111 which is redirected via the SSH tunnel to port 11111 of the computing node on the cluster on which we launched the Paraview server a little earlier.

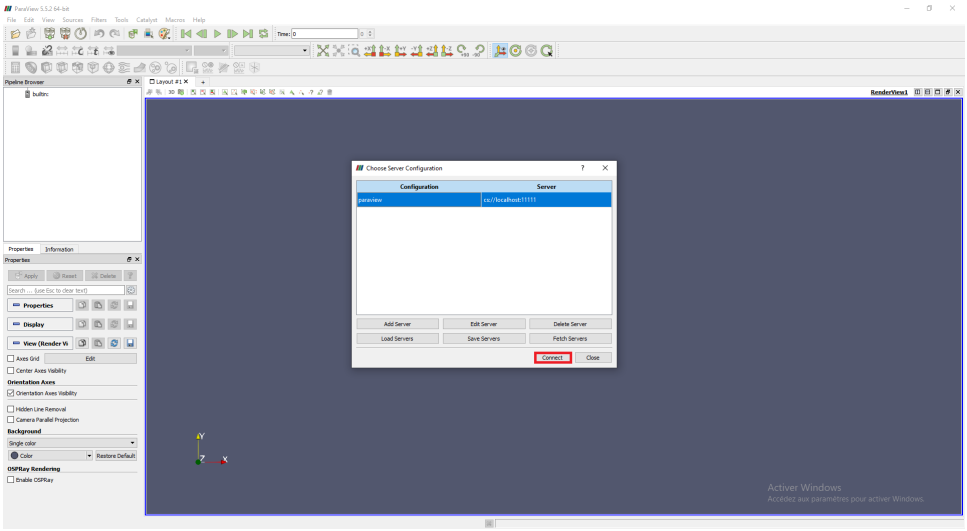


Figure 36: Connection to port 11111 of your machine in Paraview

If all goes well, there should be no errors once you click connect, but the screen may freeze for a few moments. We should then see the display change in the left part of Paraview to match the image 38. We can also check that in the MobaXterm terminal where we launched the Paraview server, it should be indicated *Client Connected*.

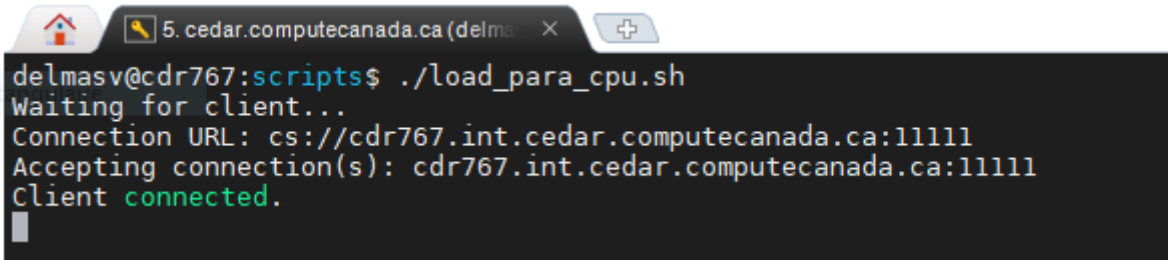


Figure 37: Connection to Paraview server successful

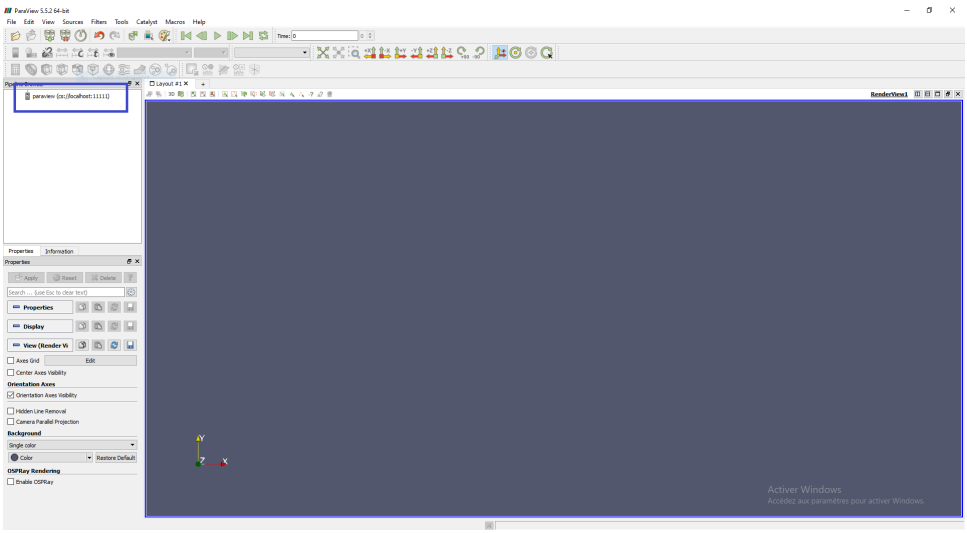


Figure 38: Successful connection to Paraview remote server

To have a concrete example, the section 4.2.2 details the opening of a file on the remote server.

4.2 Viewing result files generated by CUTEFLOW on Paraview

We present here a basic use for visualizing the out_*.cgns files produced by the *CuteFlow* code. It is assumed that the out_*.cgns files have already been downloaded locally or that the user has already connected to the computing cluster via an ssh tunnel. Be careful to read the out_*.cgns files which are the output files of *CuteFlow* and not the *.cgns mesh files which do not contain the temporal solution.

The files that will be used in this section are located in the *Traitement_des_fichiers* folder.

Note: When using Paraview, several tools can be selected by searching through the different menus. For simplicity, it is possible to use the shortcut *Ctrl + space* to open a search box in which all the tools are present. This is for example what is done in figure 41 to search for the tool *Merge Blocks*.

4.2.1 Viewing rect file_6138

We will start by comparing the different orders with an exact solution in the file out _mesh_6138, a rectangular area 10m long and 1m wide.

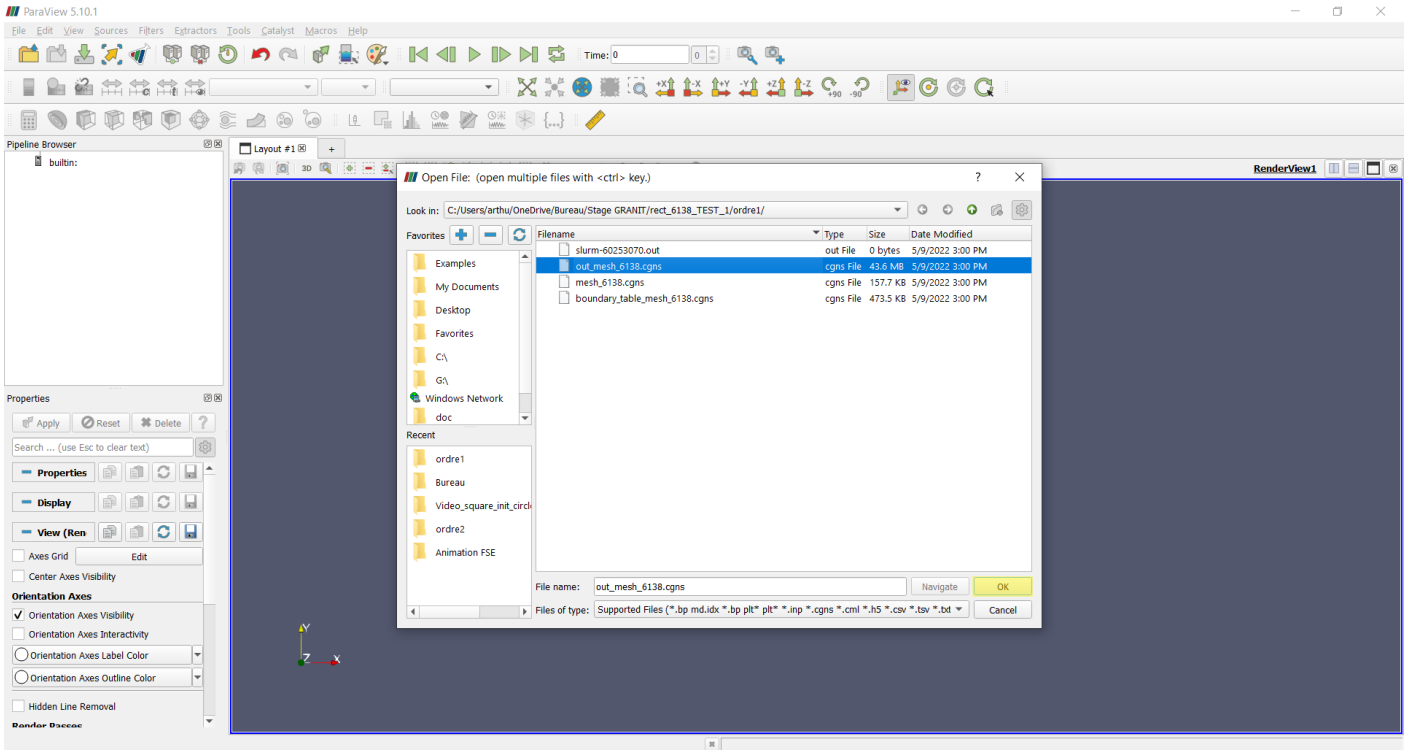


Figure 39: Opening the out_mesh_6138 file of order 1 with *File* and *Open...*

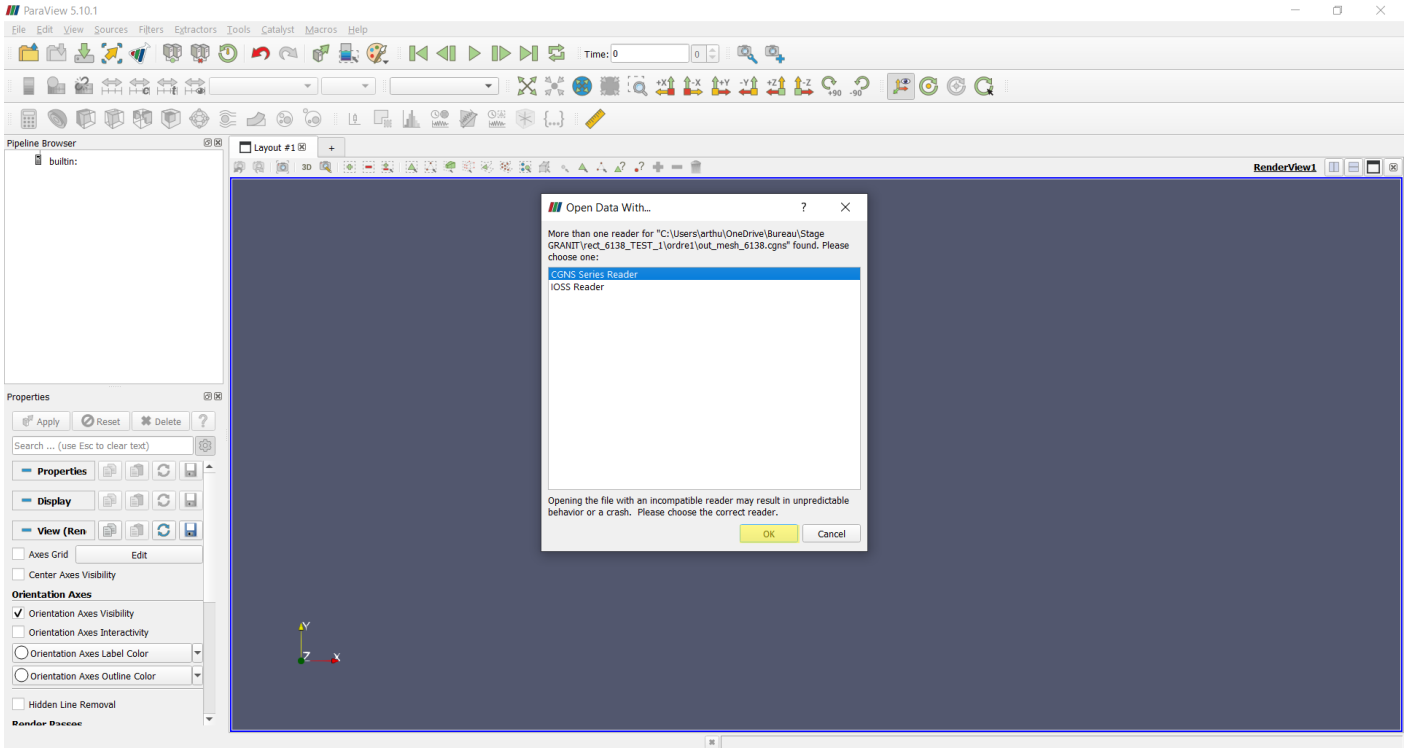


Figure 40: Reader's choice

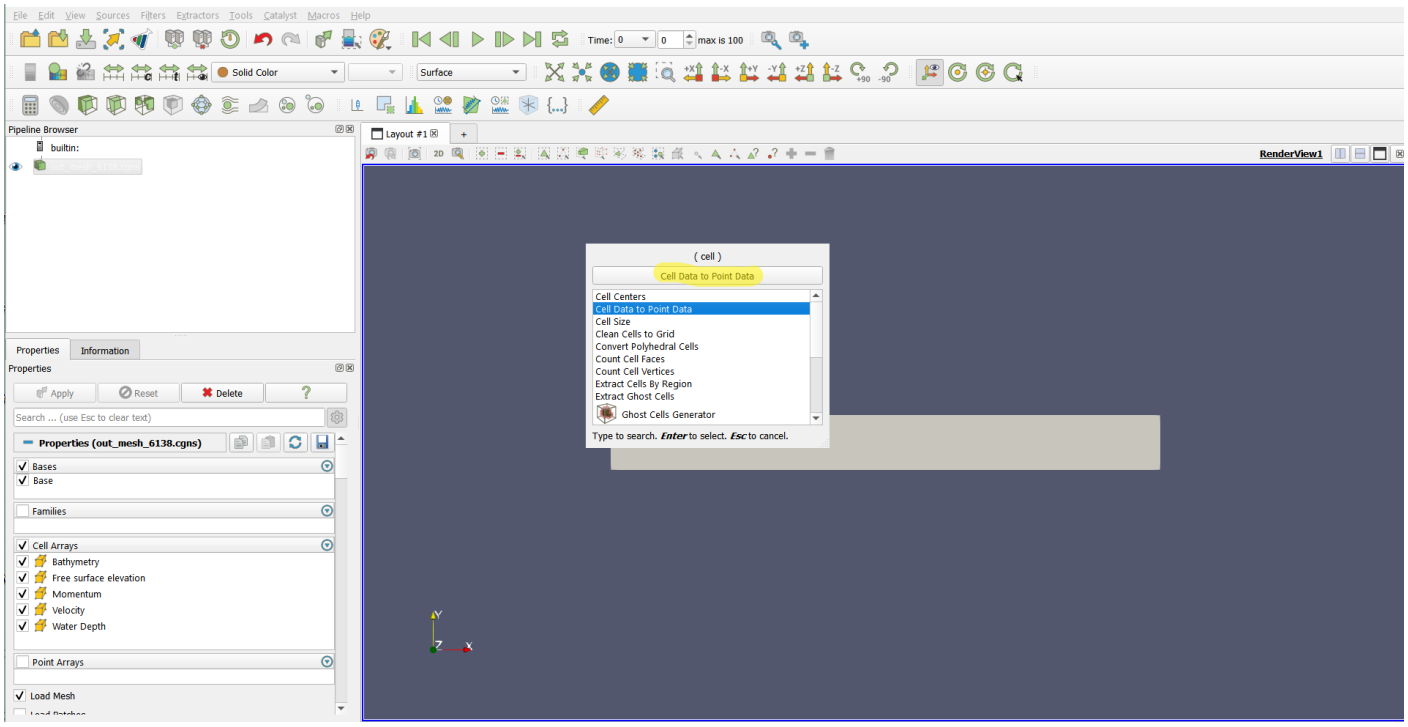


Figure 41: Selection of the *Cell Data to Point Data* tool to avoid aliasing of traced solutions

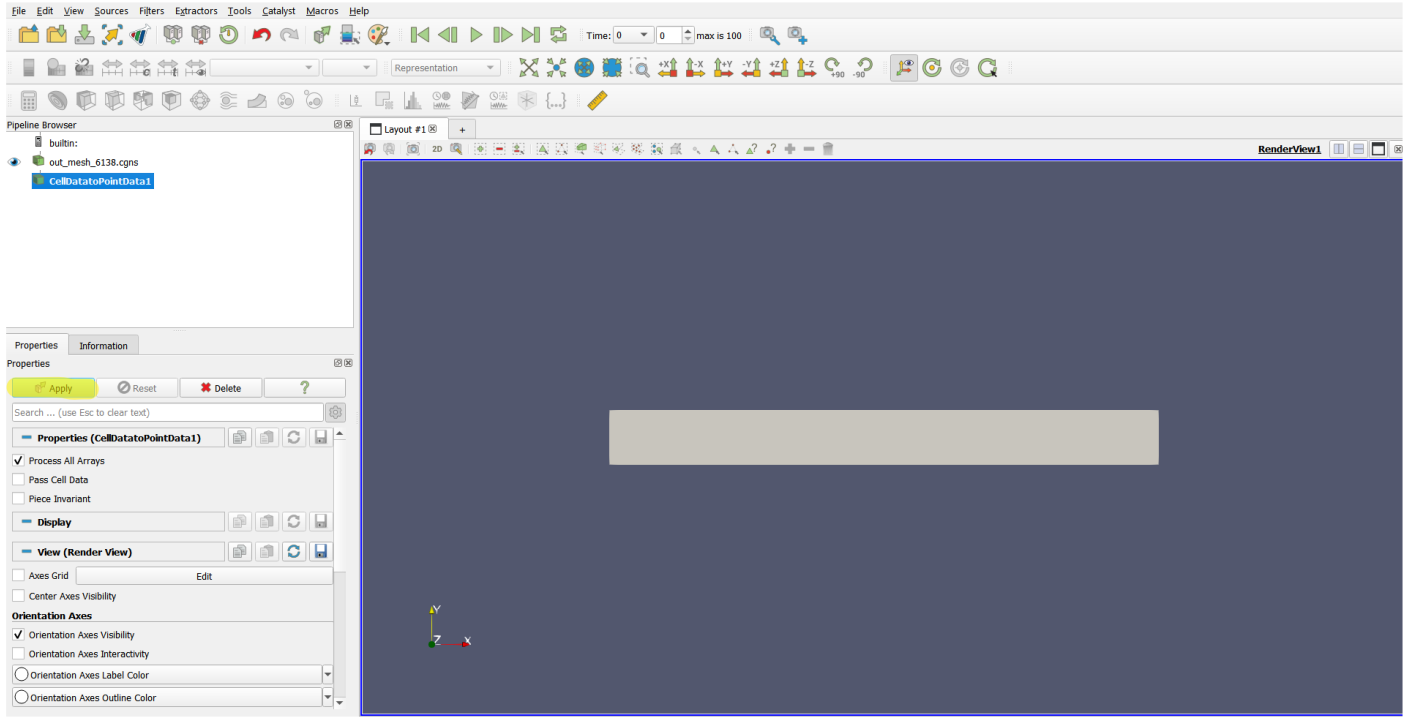


Figure 42: *Apply* to perform the operation

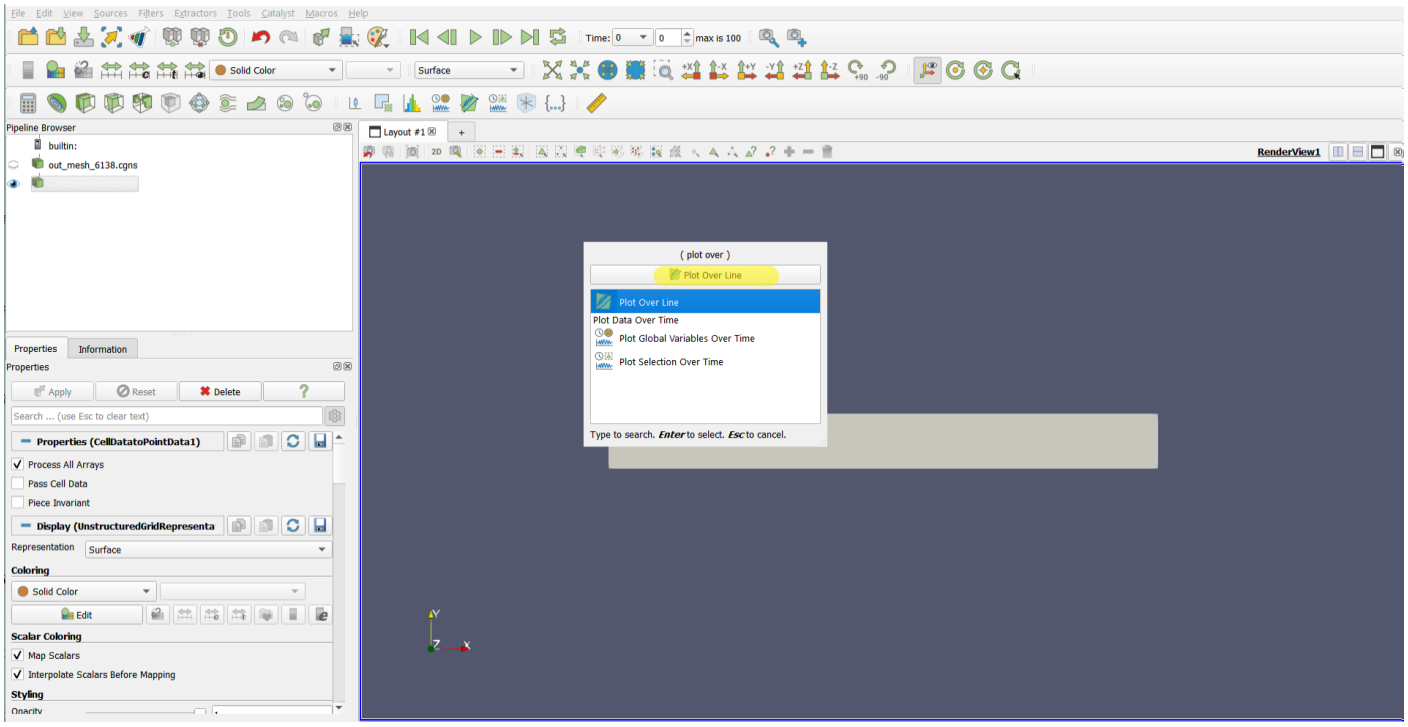


Figure 43: Selecting the tool *Plot Over Line*

Note: From now on, the use of the Ctrl + space shortcut to select the tools will not be mentioned to lighten the guide and facilitate reading but must be done by the user.

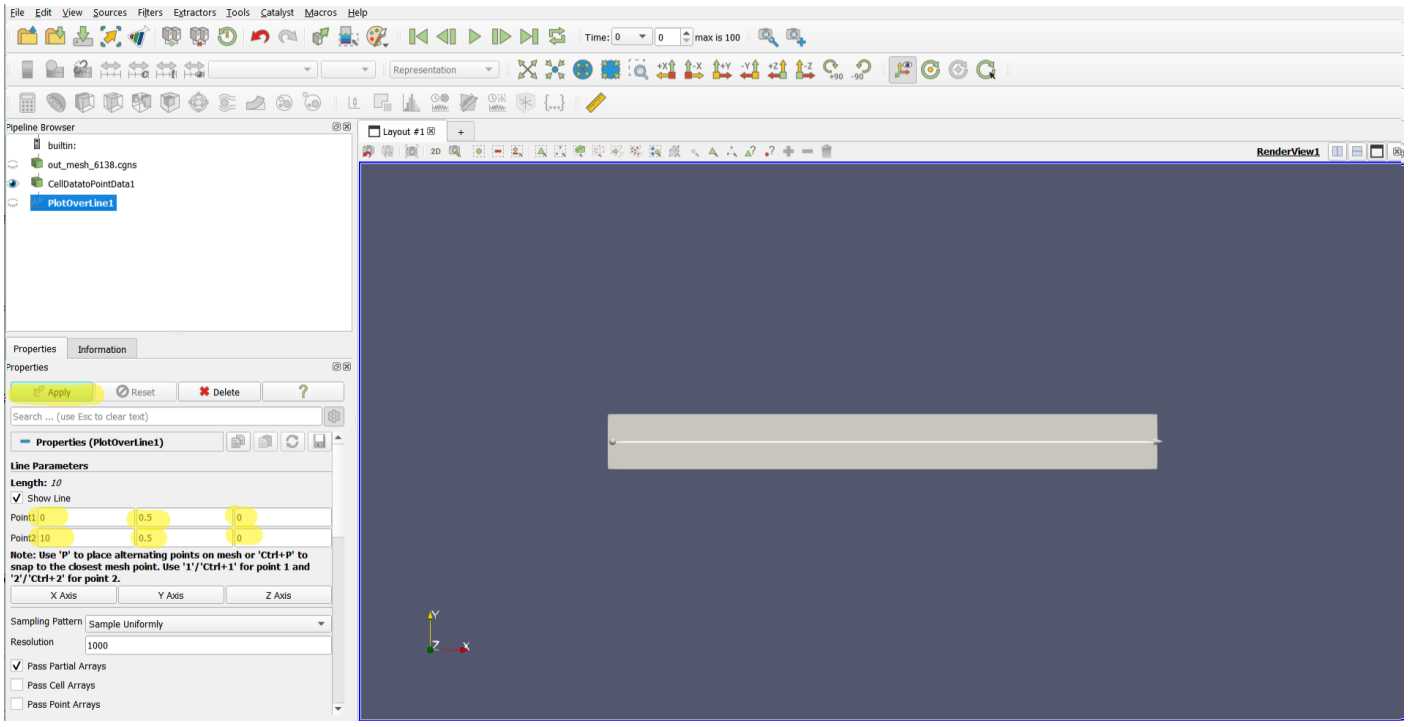


Figure 44: Choice of coordinates of the starting and ending points of the line, then *Apply*.

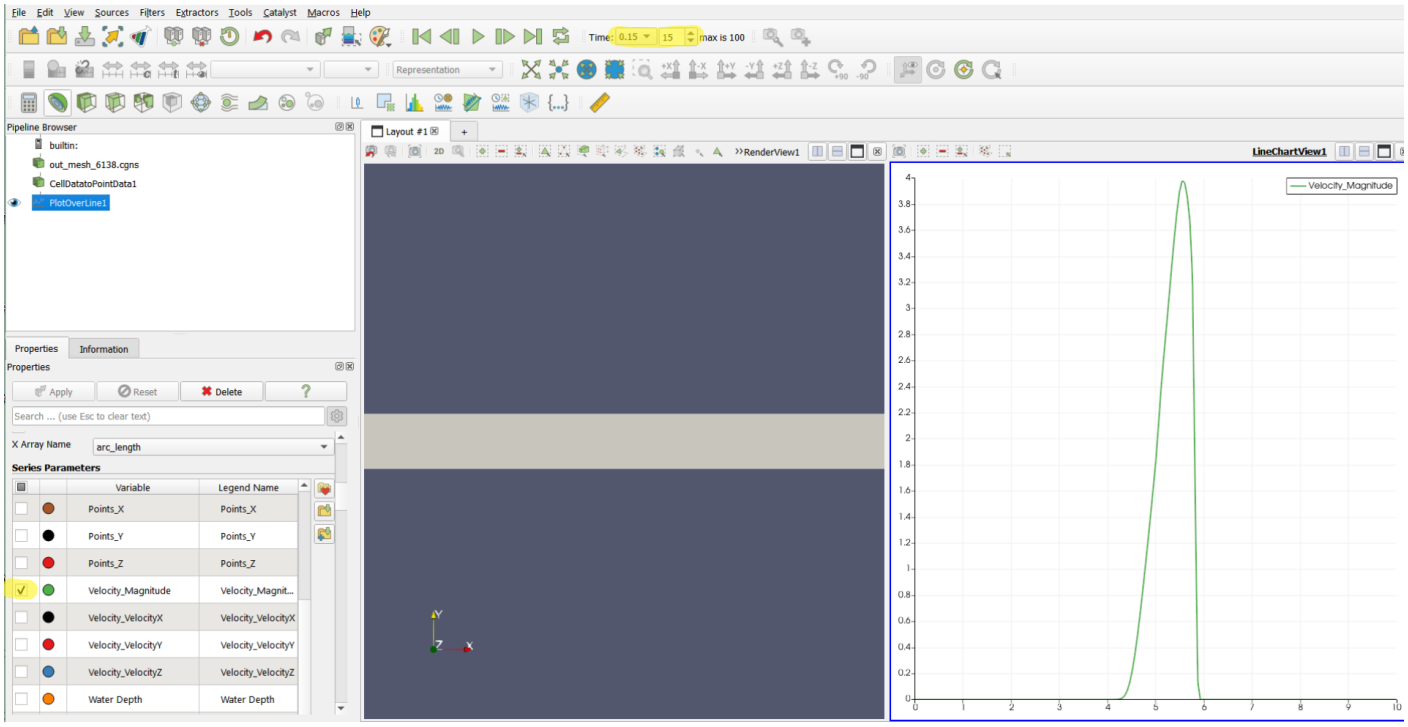


Figure 45: Selection of the attributes of the solution to load and the time of the solution (if the menu on the left is not displayed correctly, left click on the graph on the right).

We will then perform the operation again on the second order file as well as on the exact solution by selecting the file *exact_sol.vtk.series*.

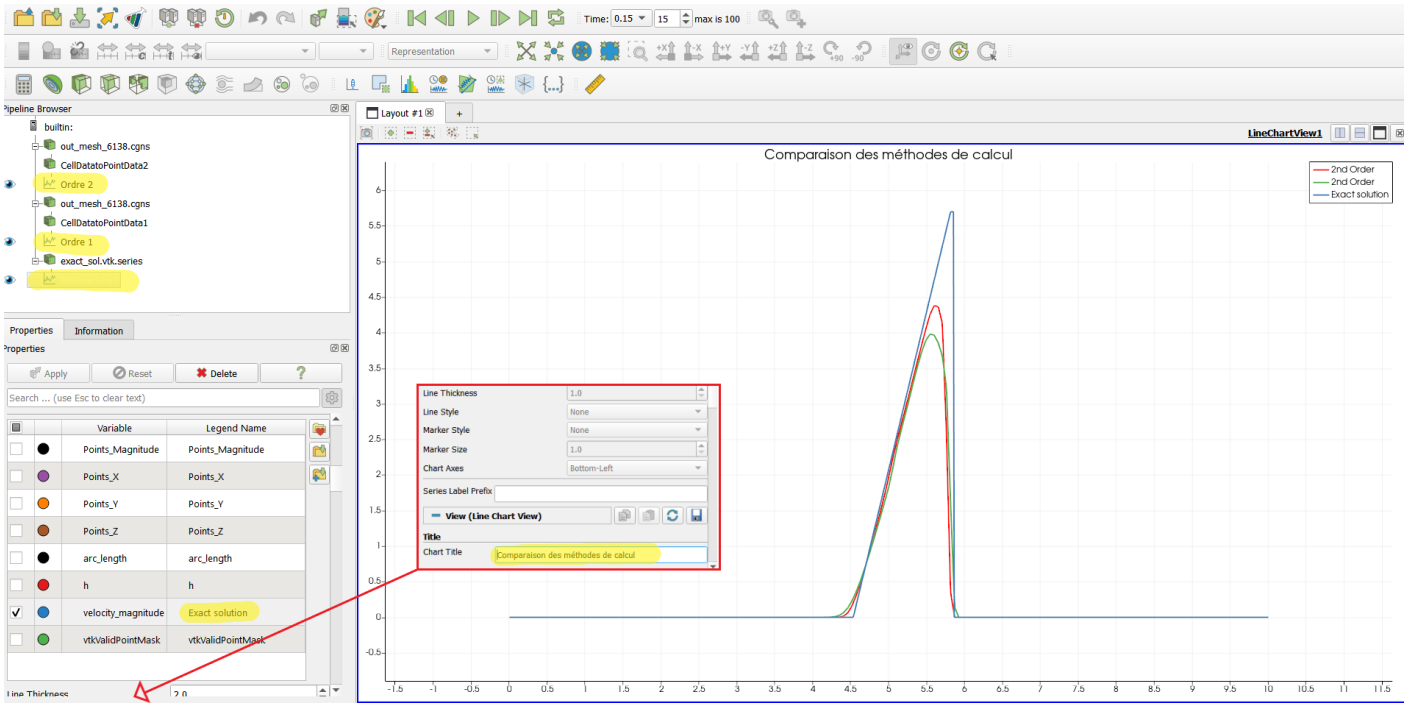


Figure 46: Editing the graph legend and title (located below)

It should be noted that many parameters are available to modify the appearance of the graphs (vertical and horizontal scale, legend, etc.). These modifications are simple and are therefore not all detailed in this guide.

You can then save the state of the job by going to *File* at the top left of the work window, then selecting *Save State* and choosing the reception file. To open the project again, simply do: *File* then *Load State* and select the backup created. Be careful not to move or modify the files in the meantime.

4.2.2 Viewing the rect_6138_bump

file The file out_rect_6138_bump.cgns is similar to the previous one with the difference that the surface is not flat. We will carry out the simulation on the calculation cluster in order to show step by step with a concrete case a simple use of the latter.

We will start by accessing the connection node:

```
# From your Linux terminal emulator (MobaXterm type)
ssh YourUserName@graham.computecanada.ca
Your password

# From ~/scratch/CuteFlow/scripts/
./request_visu.sh 2
./load_para_cpu.sh
```

We will then connect by SSH tunnel to the assigned computing node. For more details, user can refer to section 4.1.

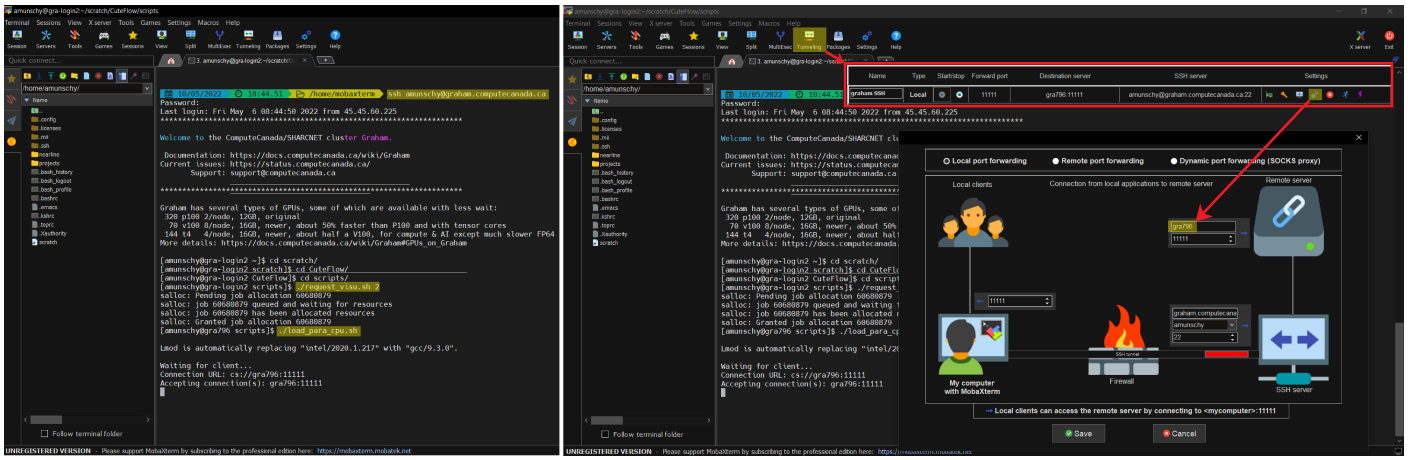


Figure 47: Connexion au nœud de calcul

Once the connection is established, the user launches ParaView locally and then connects via the *Connect* icon. Line *builtin* in the *Pipeline Browser* (left) is then replaced by *Graham_Paraview (cs://localhost:11111)*

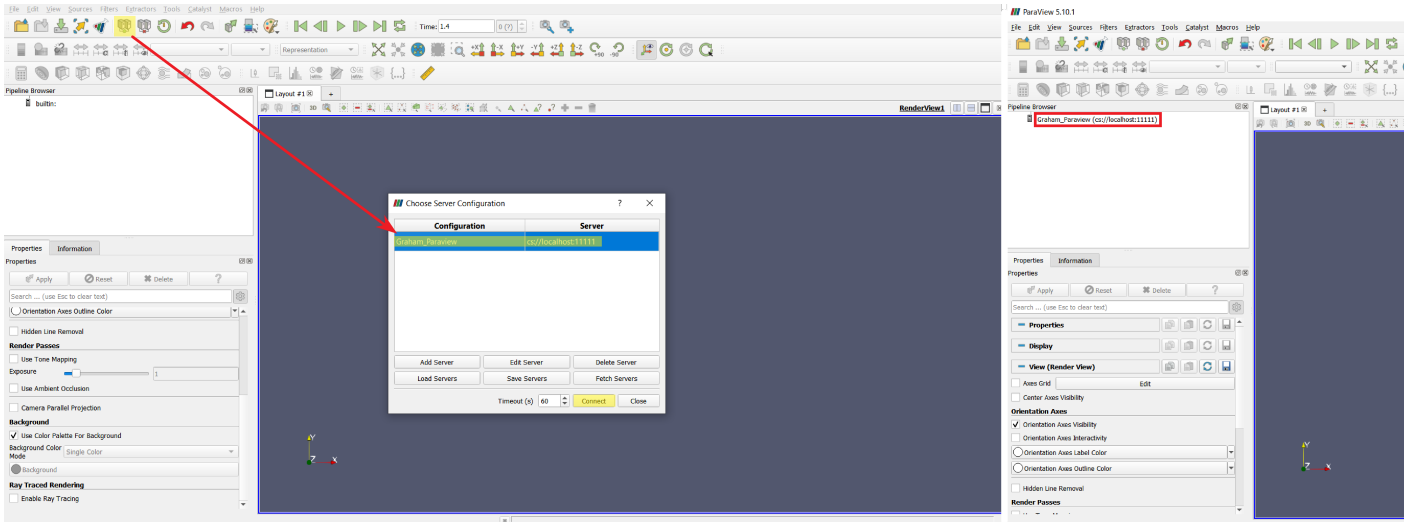


Figure 48: Launching ParaView

The user will now be able to open the file out_rect_6138_bump.cgns on the server. We will then carry out the procedure to draw the desired graph along a line, this procedure is the same as that described in the following section (section 4.2.3). Here is the kind of simple graph that we can draw from this work, comparing here the water heights (FSE: Free Surface Elevation):

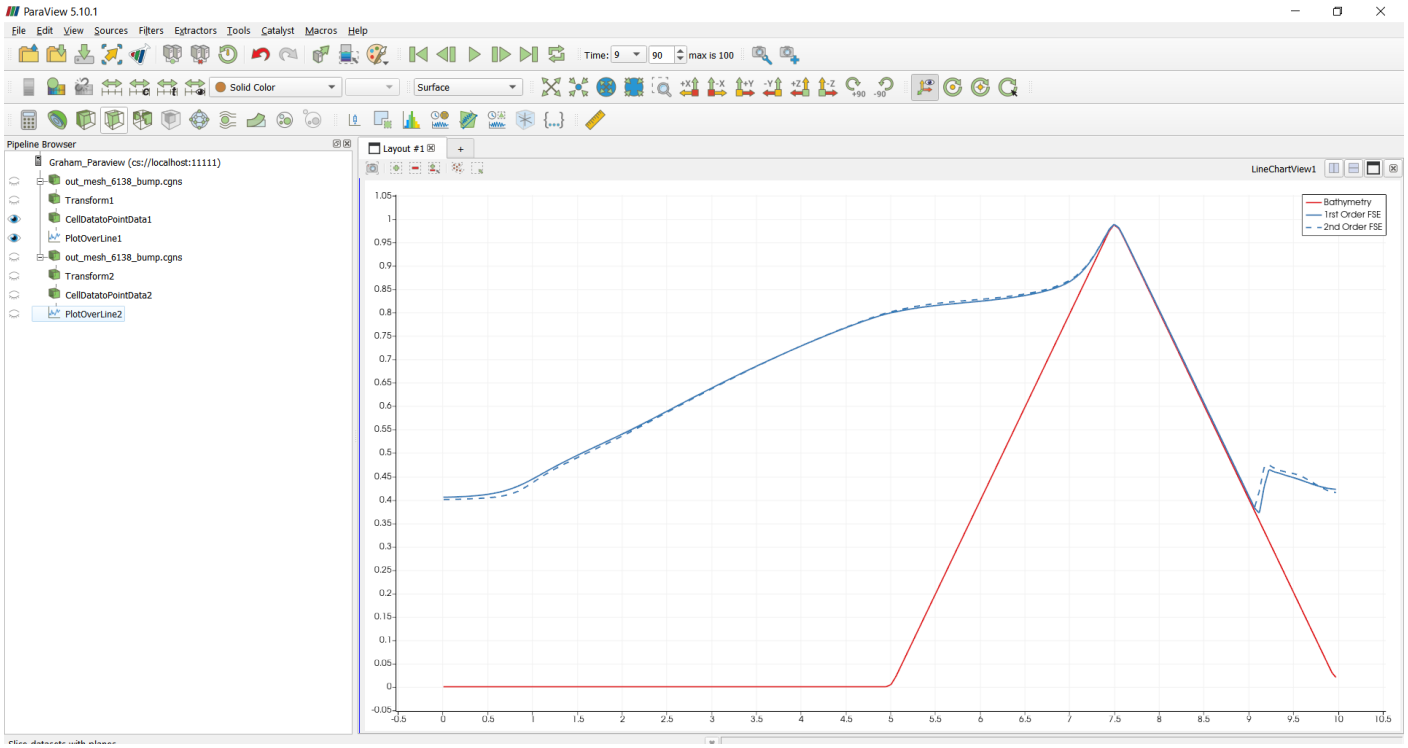


Figure 49: Results of work on **out_mesh_6138_bump.cgns**

4.2.3 Viewing the file out_rect_95459_bump.cgns

In this section we show the procedure to follow to display the solution contained in the file out_rect_95459_bump.cgns. We first seek to display the solution along a line which crosses the domain.

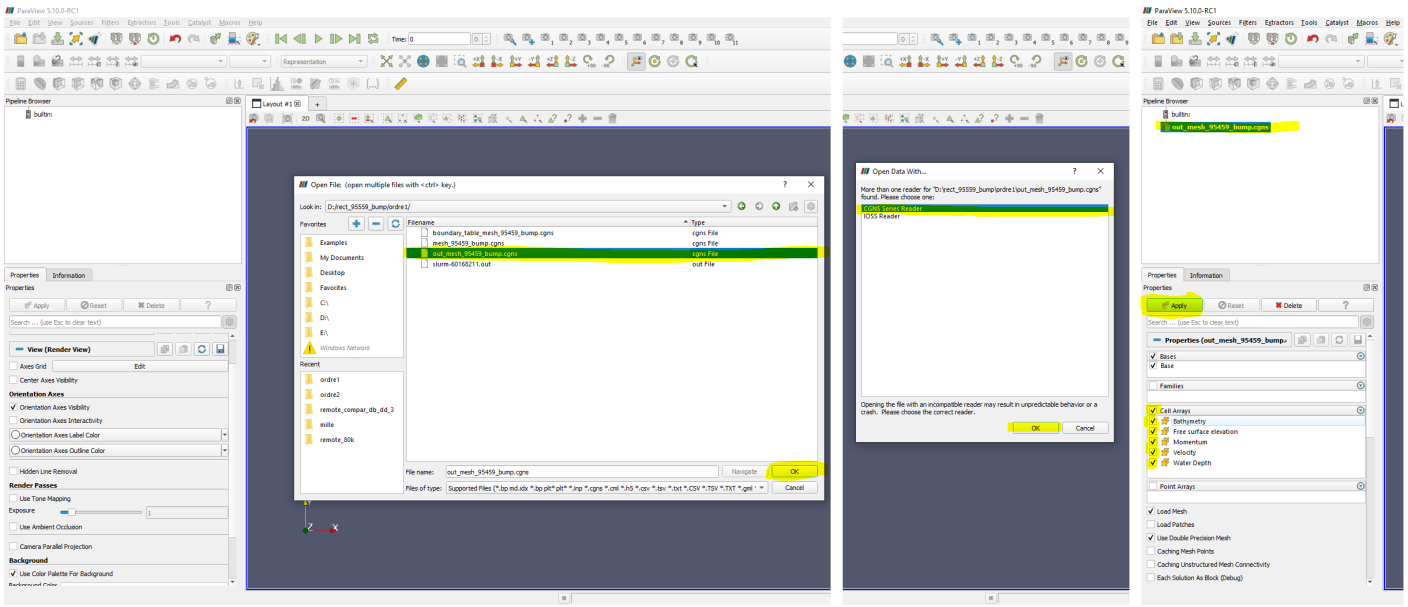


Figure 50: Opening the file, choosing the reader and selecting the attributes of the solution to load

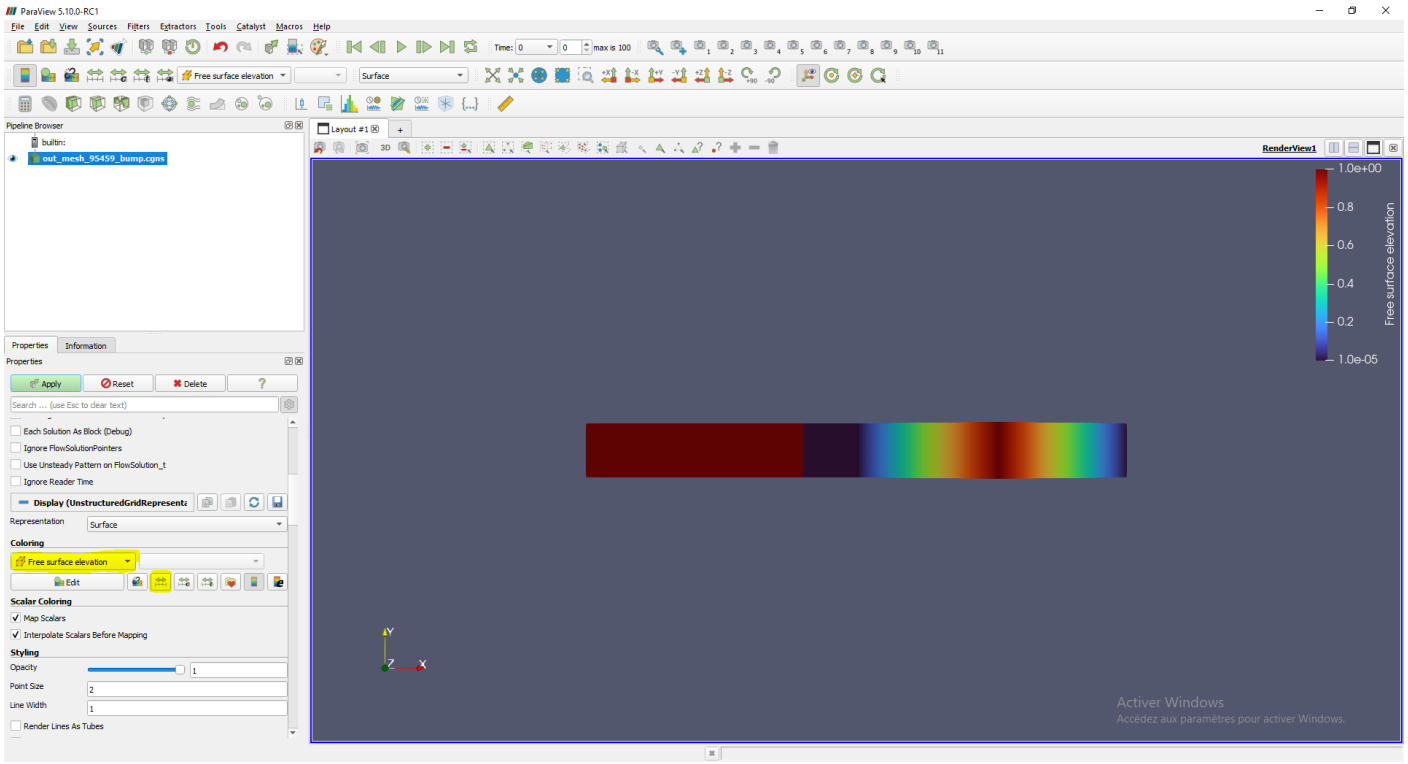


Figure 51: Coloring of the solution depending on the height of the free surface

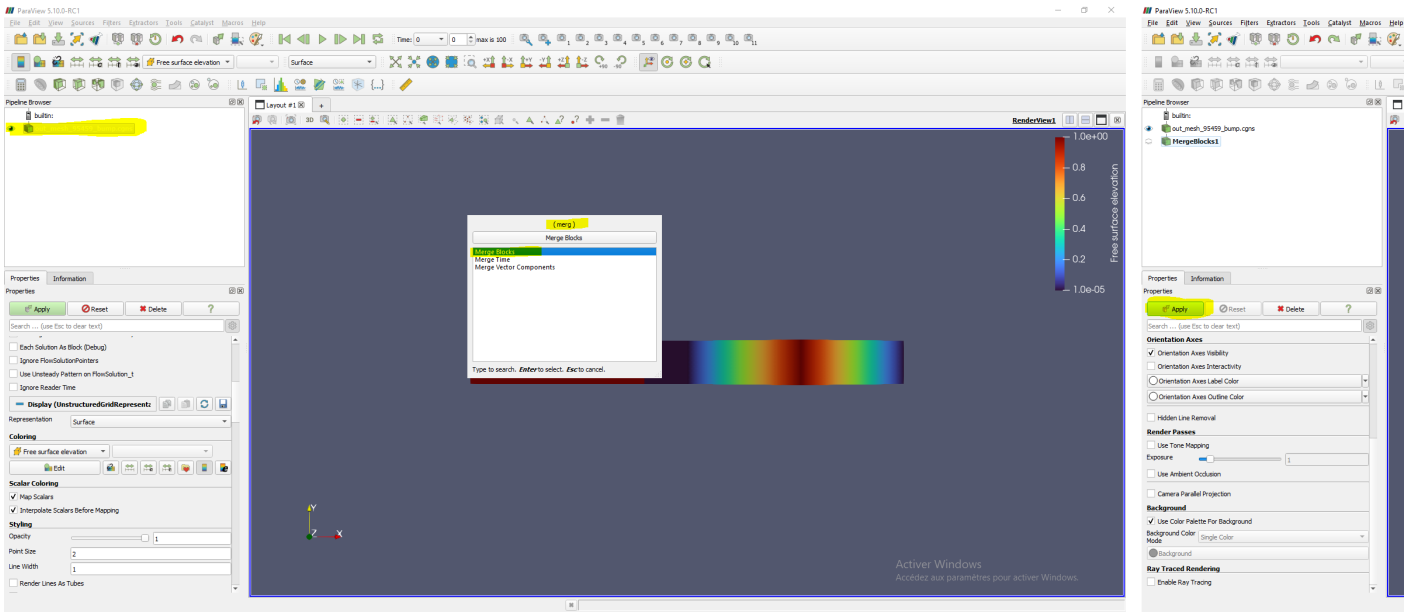


Figure 52: Selection of the *Merge Block* tool to avoid visualization artifacts when several subdomains are present. *Apply* to perform the operation

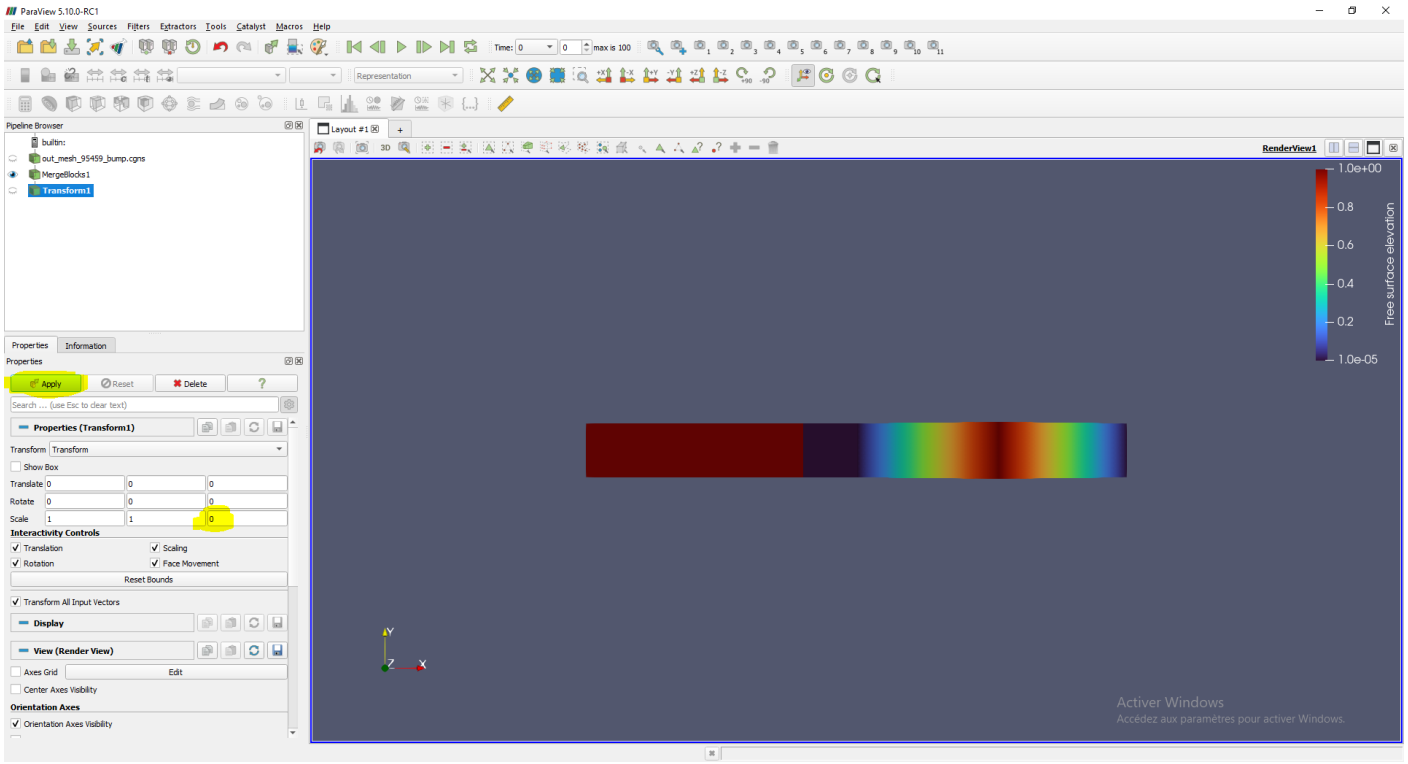


Figure 53: Using *Transform* to flatten the bump. Helps avoid problems when projecting onto a line.

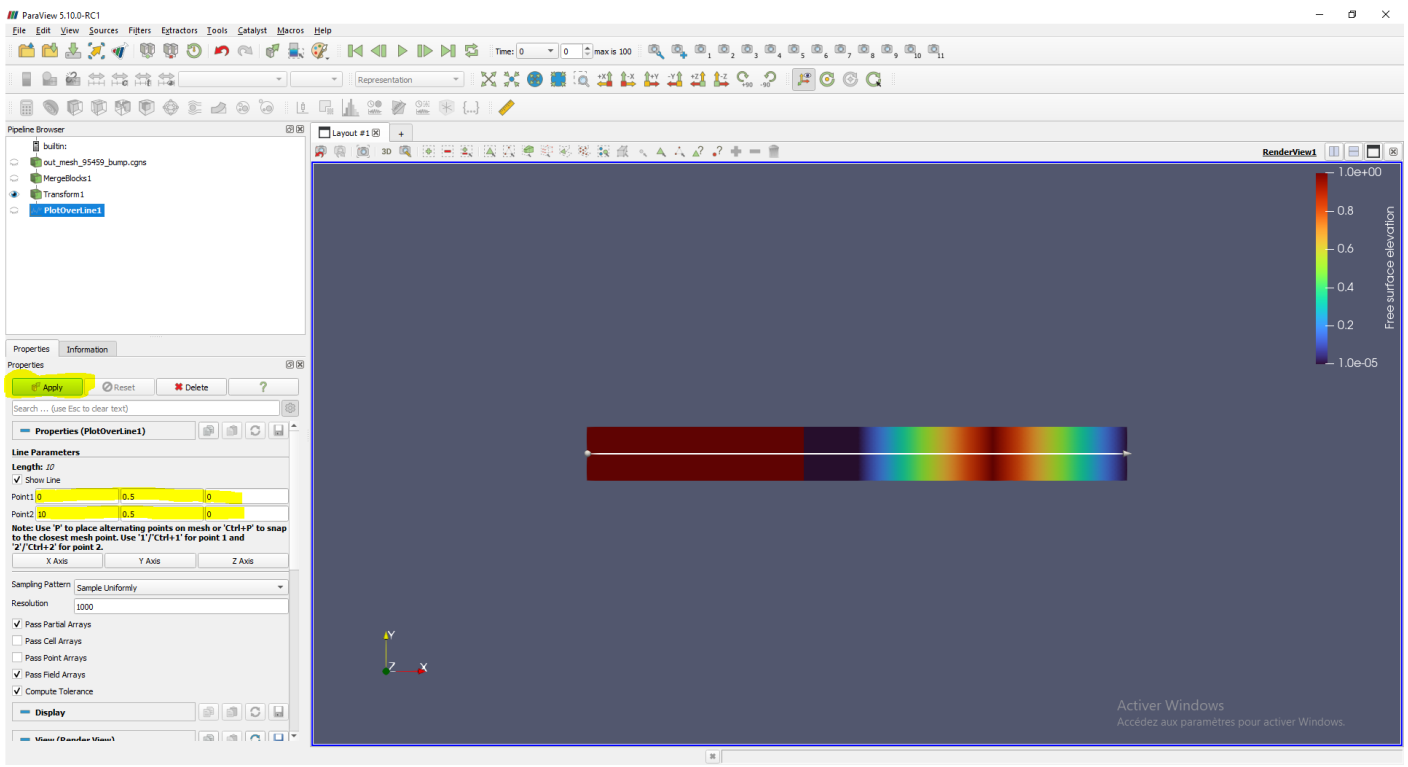


Figure 54: Choosing the coordinates of the starting and ending points of the line with the *Plot over line tool*

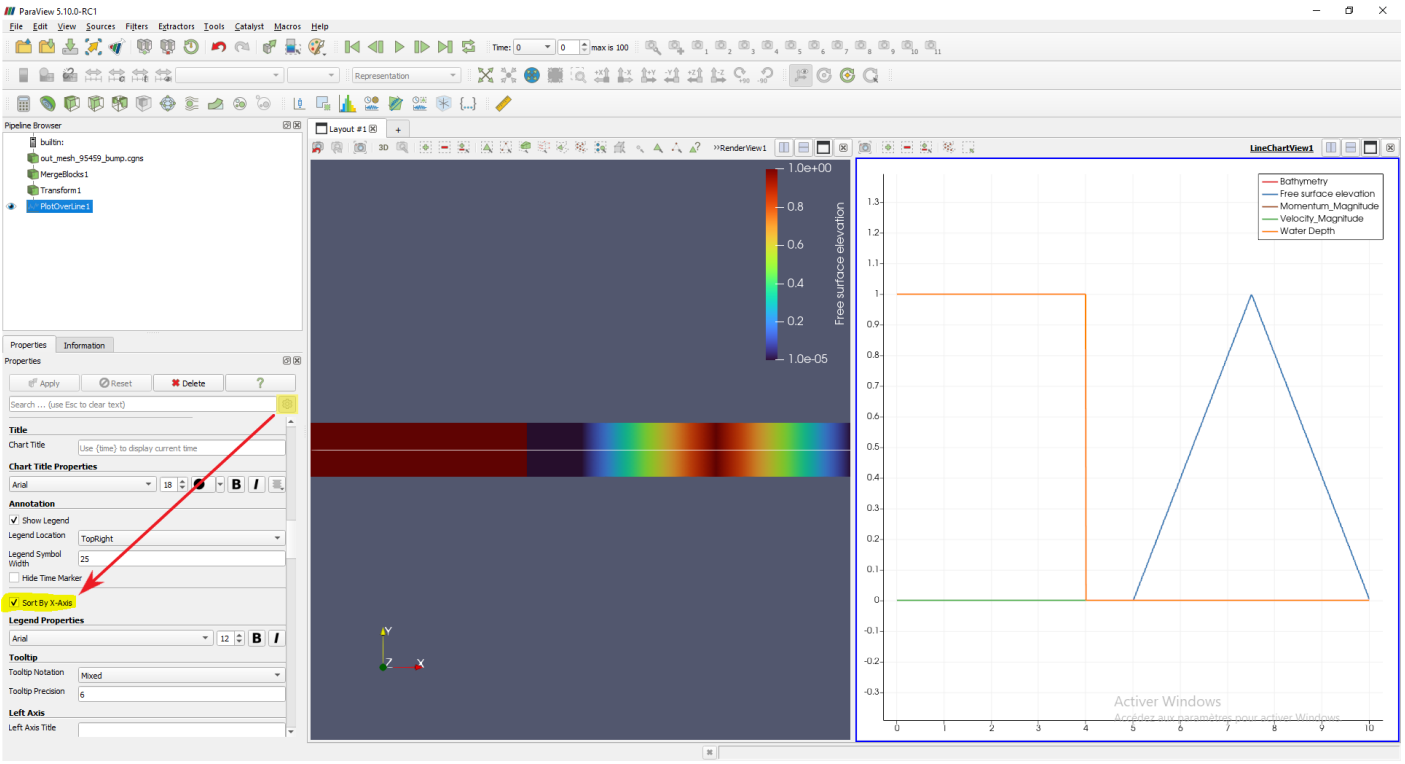


Figure 55: Selection of advanced parameter mode and *Sort by X-axis* parameter to avoid having points in the wrong order (especially useful for real meshes)

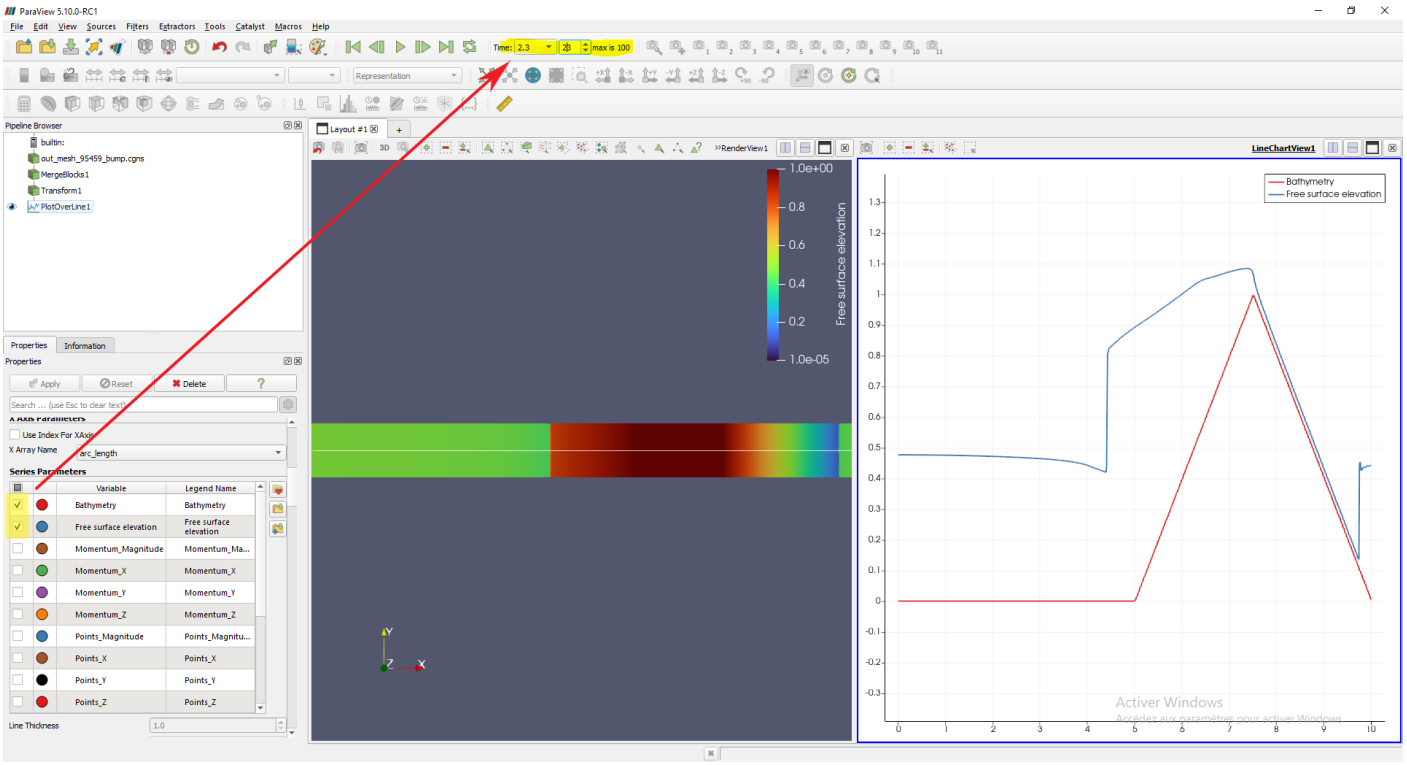


Figure 56: Selection of the attributes of the solution to load and the solution time

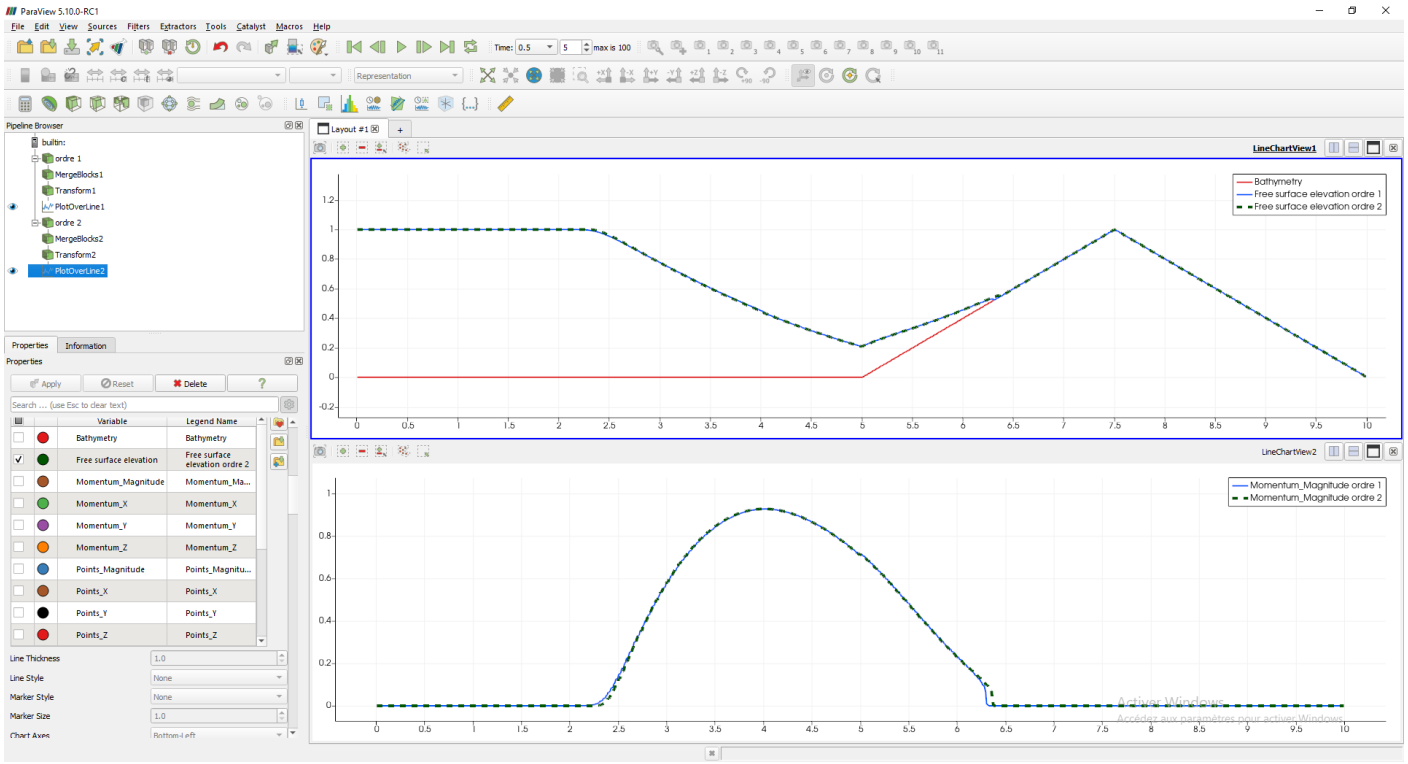


Figure 57: By carrying out the same work for another solution file, here the solution of order 2, we can display the two solutions on the same graph.

4.2.4 Viewing the mille_700k_2_2.cgns

file

We present here how to visualize the solution on a real domain from the mille_700k_2_2.cgns file. In particular, we show how to separate the dry and wet areas and how to display the solution along a broken line initially. Secondly, we will try to display the flood lines.

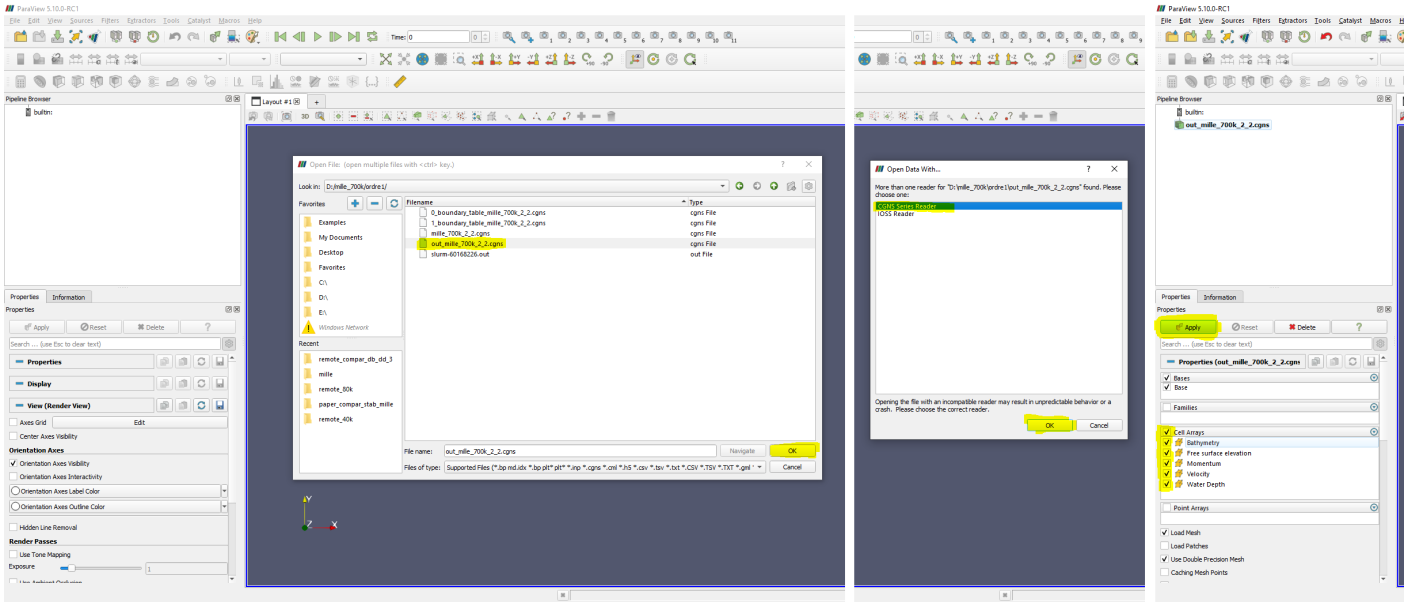


Figure 58: Opening the out_mille_700k.cgns file, choosing the reader and selecting the attributes of the solution to load

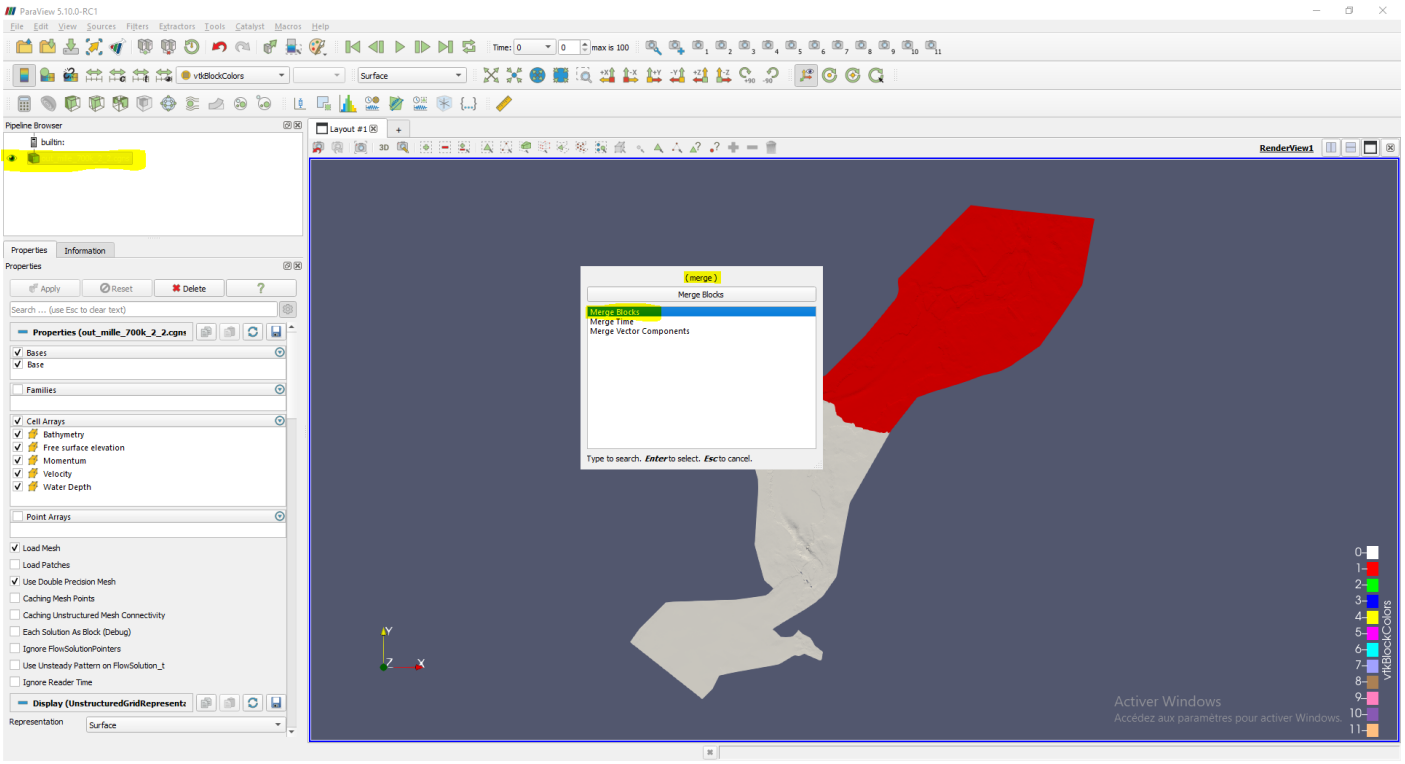


Figure 59: Using the *Merge Blocks* tool to merge subdomains. *Apply* to apply the changes

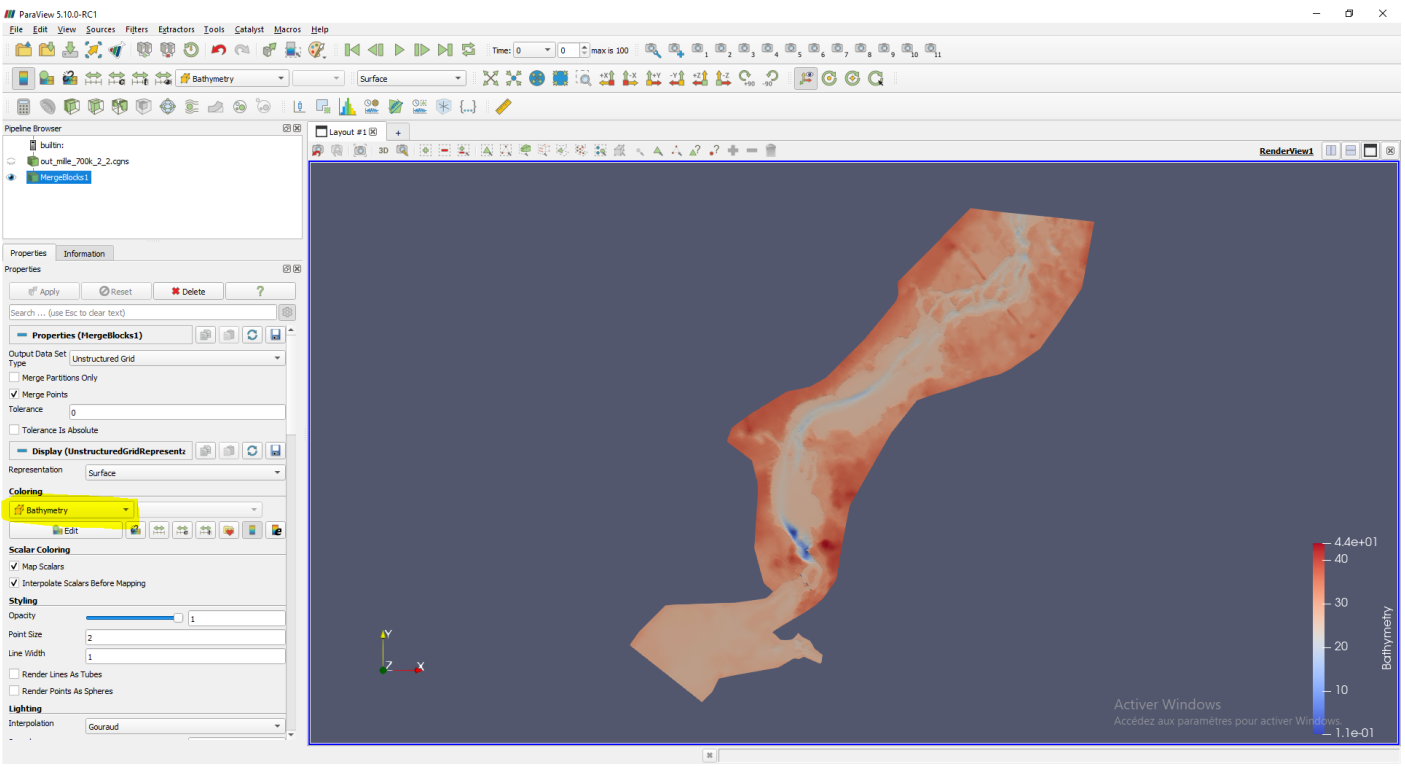


Figure 60: Selecting the solution attribute to use to color the domain

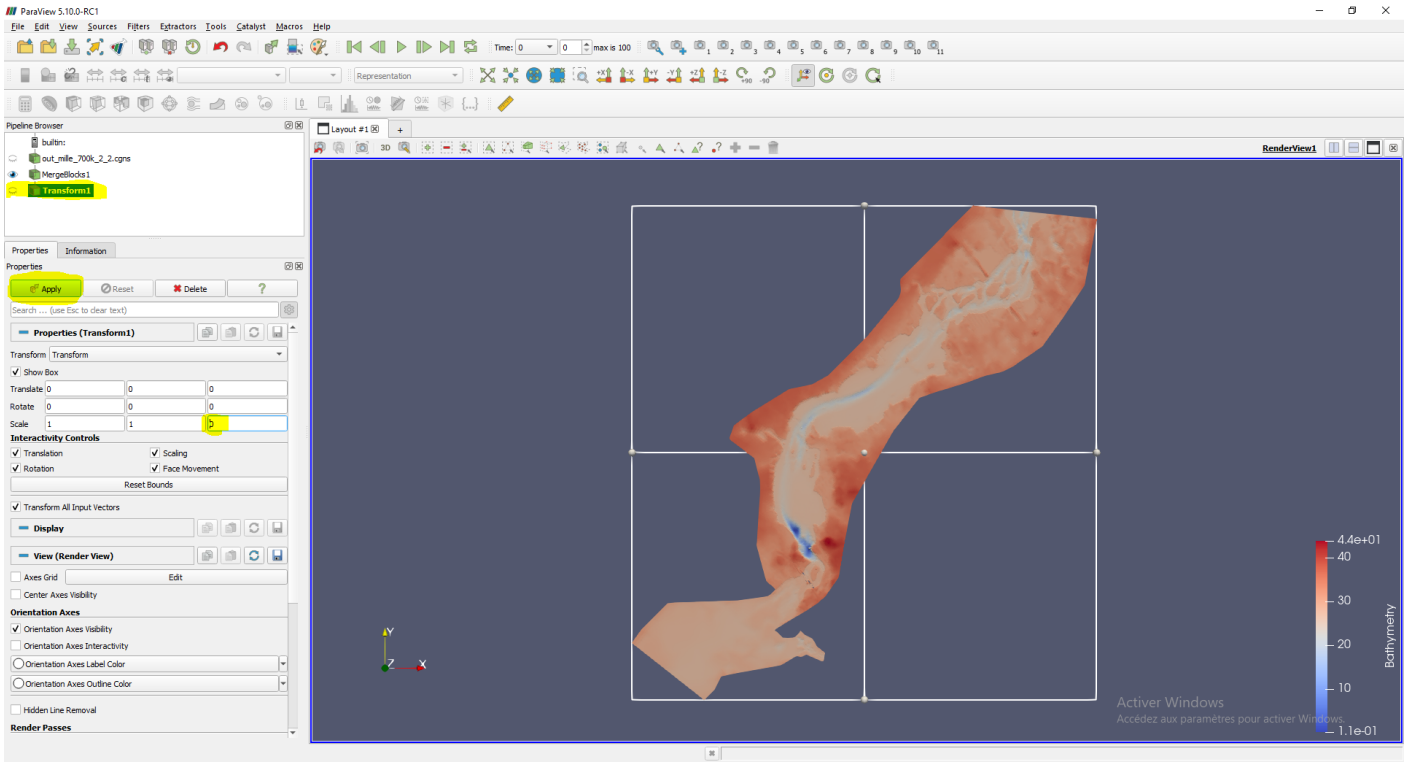


Figure 61: Using *Transform* to flatten the mesh

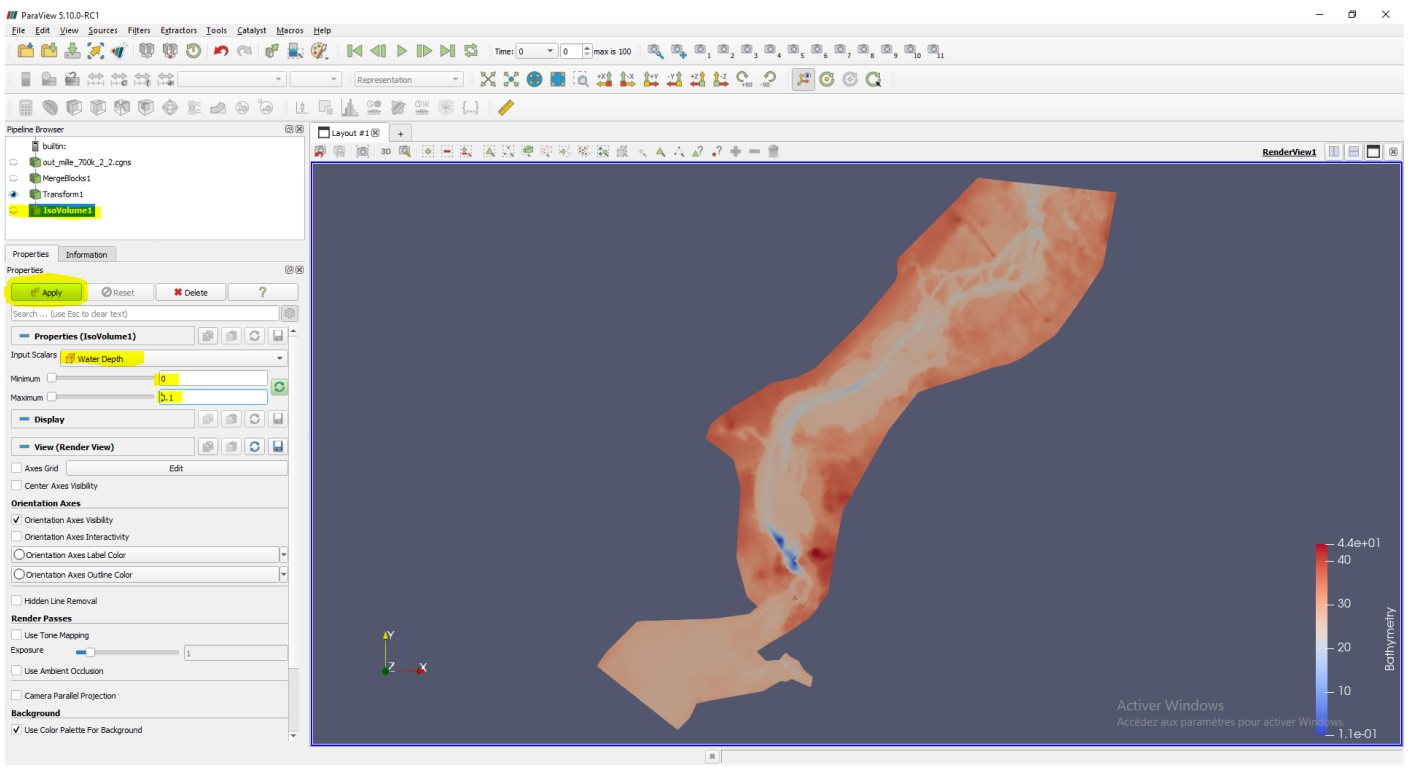


Figure 62: Choice of values to extract from the solution with the *Iso volume* tool. Here the area where the *Water depth* (water depth) between 0 and $0.1m$ will be extracted (dry area).

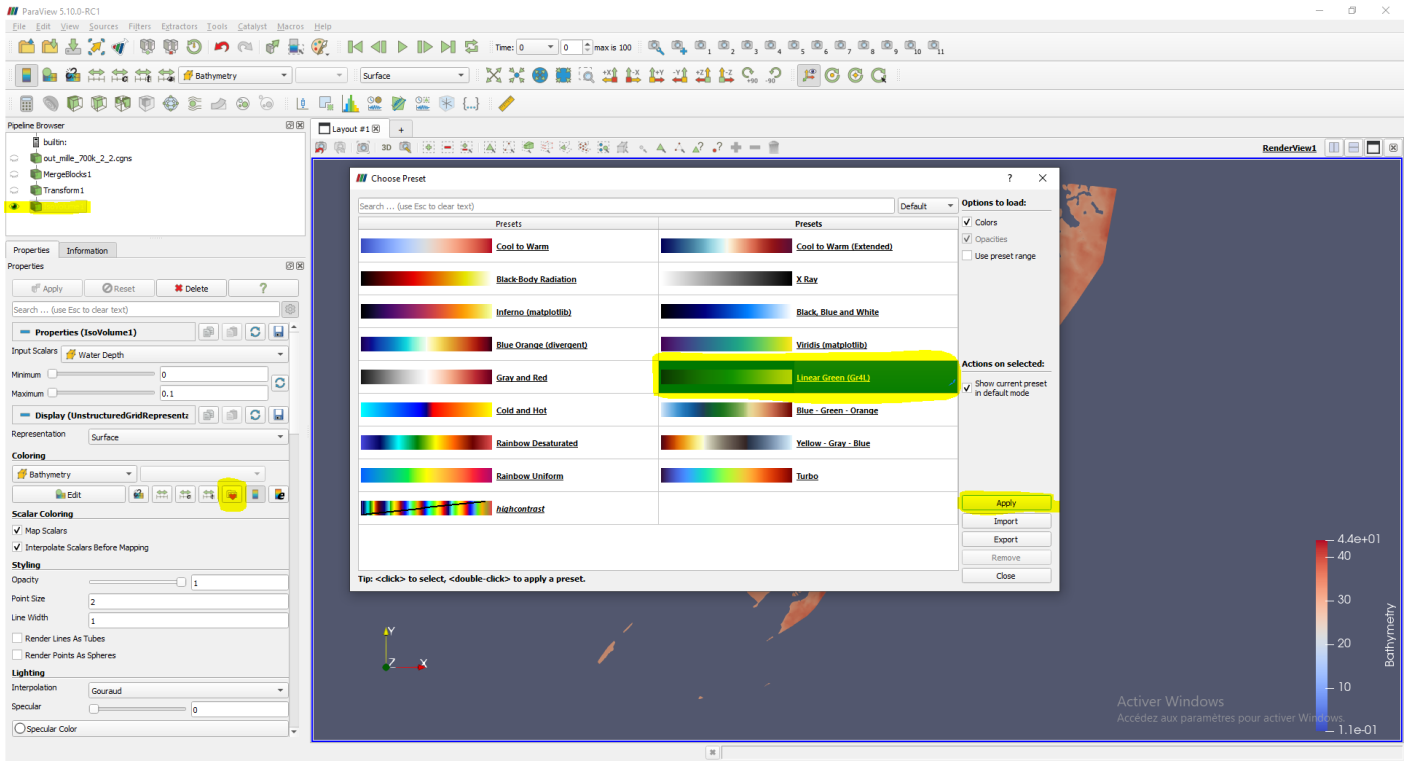


Figure 63: Choosing a color palette for the dry area.

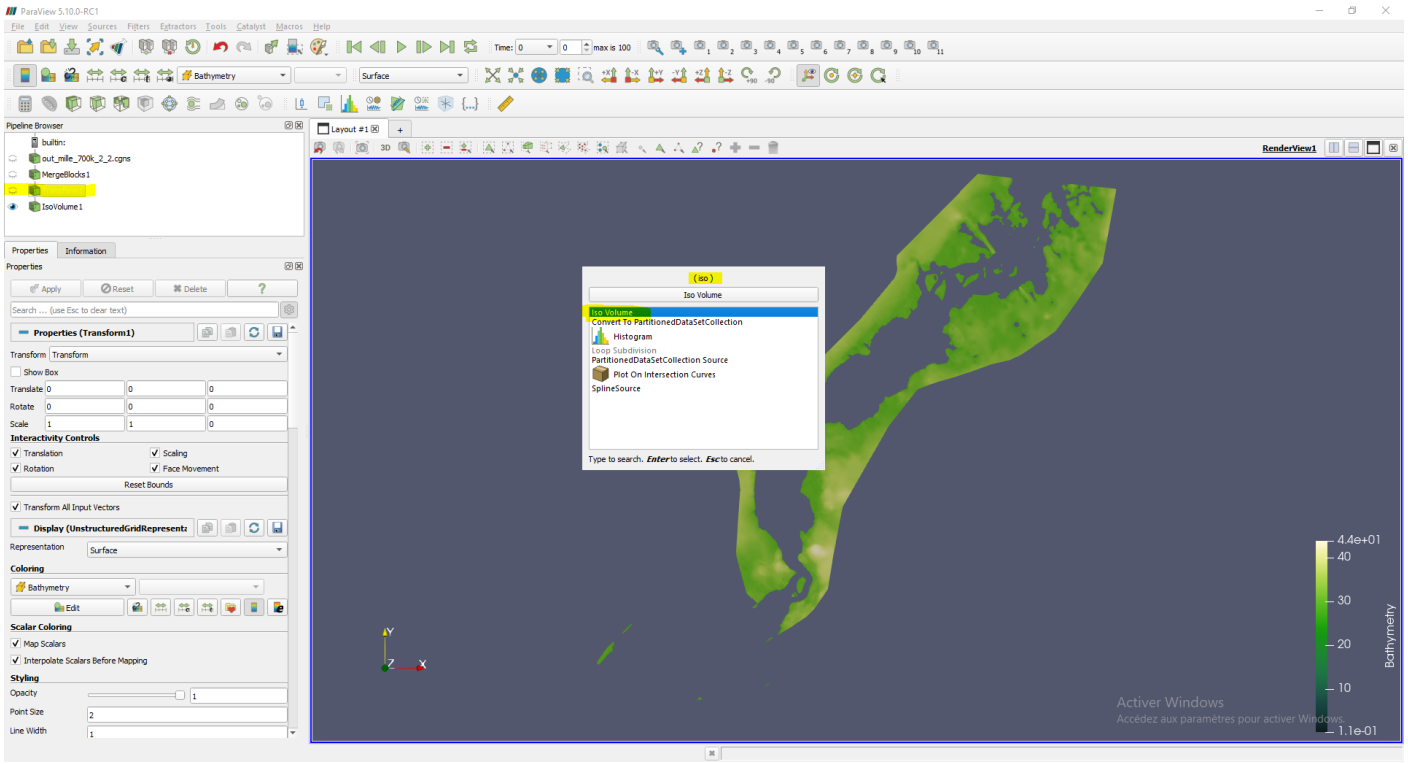


Figure 64: Selecting the *iso volume* tool a second time. Be careful to re-select the *Transform* object in the left window and not *Isovolume1*.

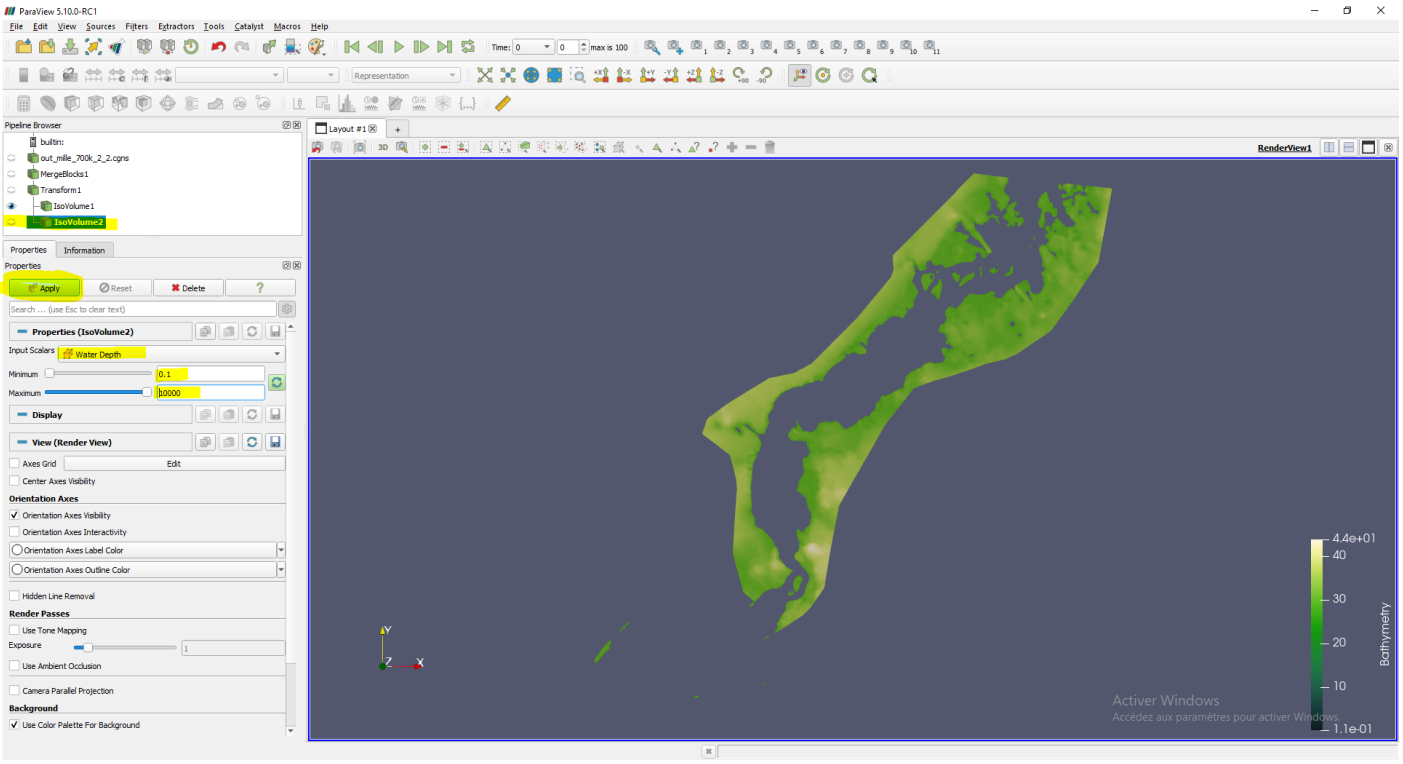


Figure 65: Choice of values of the solution to extract. Here the solution where the *Water depth* is between 0.1 and 10000m will be extracted (wet zone).

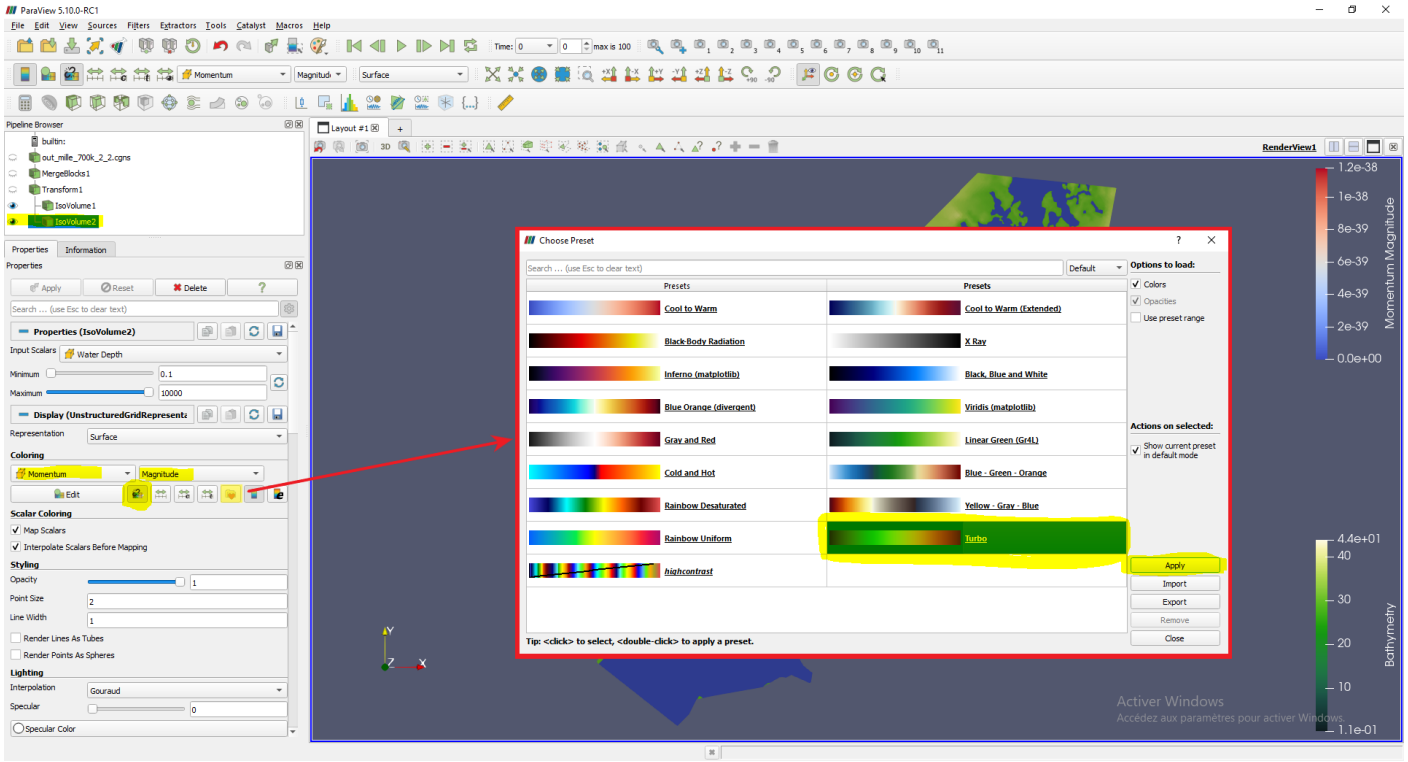


Figure 66: Choosing the attribute of the solution to use for coloring the area and separating the color palettes of the dry and wet area. Next, choice of color palette for the wetted area (see previous figures for more details).

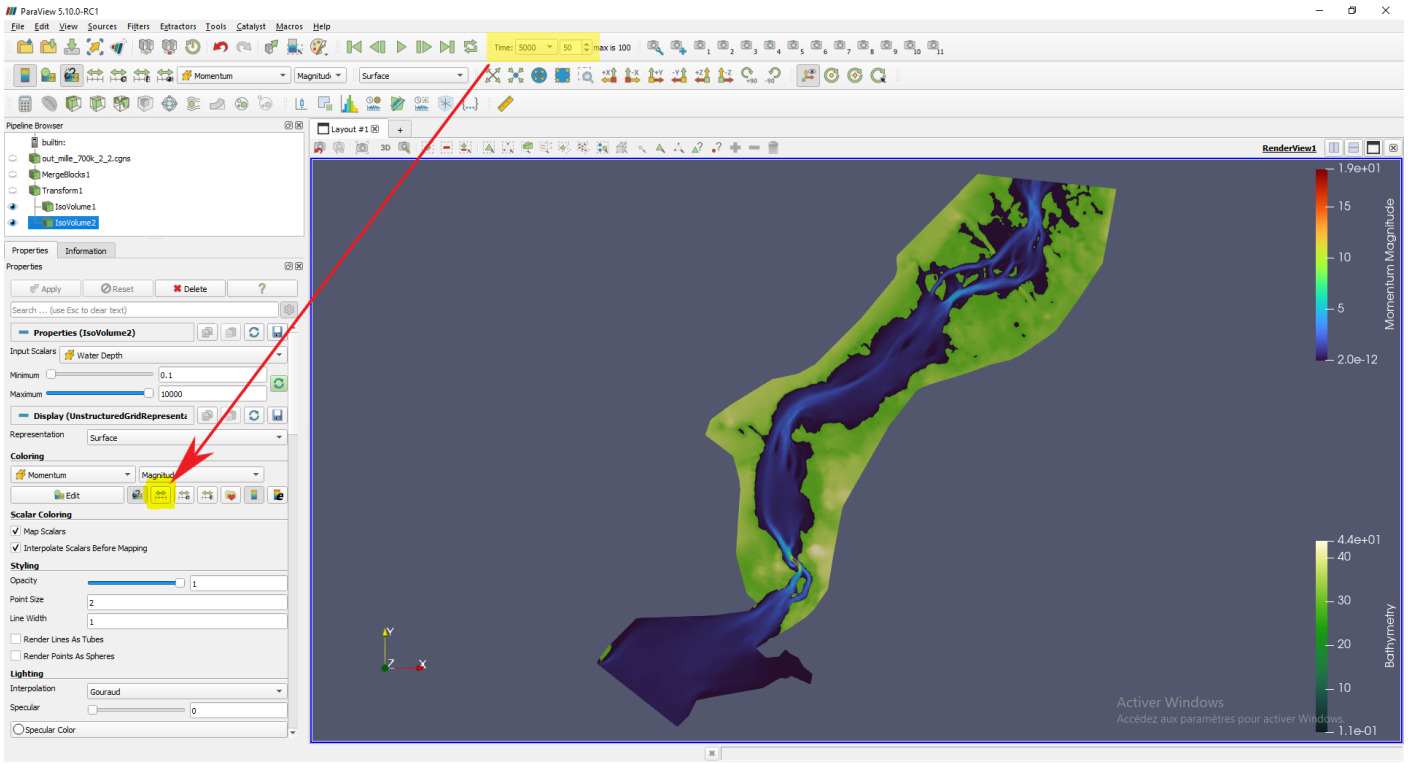


Figure 67: Solution time selection and automatic color scale readjustment.

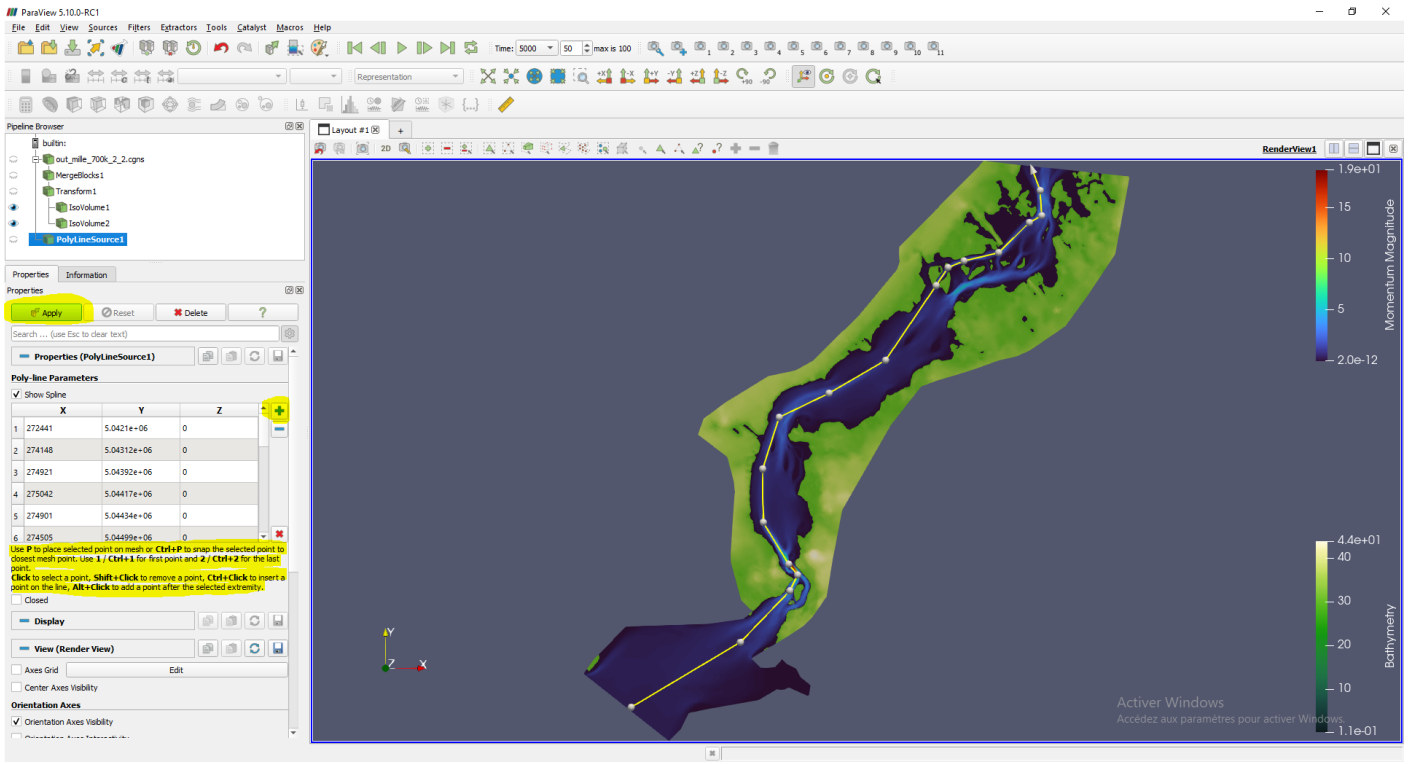


Figure 68: Choice of the *Polyline source* tool and addition of different points of the line.

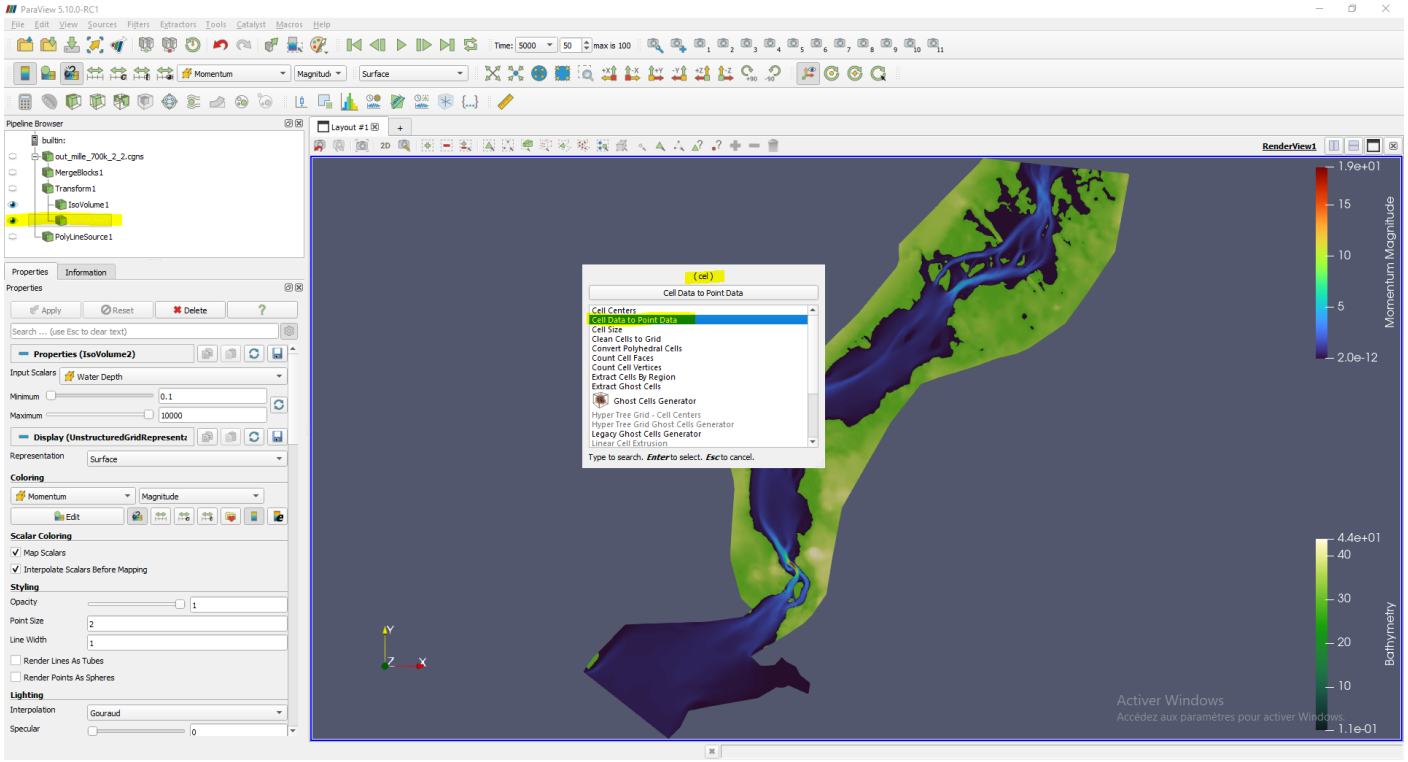


Figure 69: Selection of the *Cell Data to Point Data* tool to pass the solution from the cells to the mesh nodes. *Apply* to apply the changes.

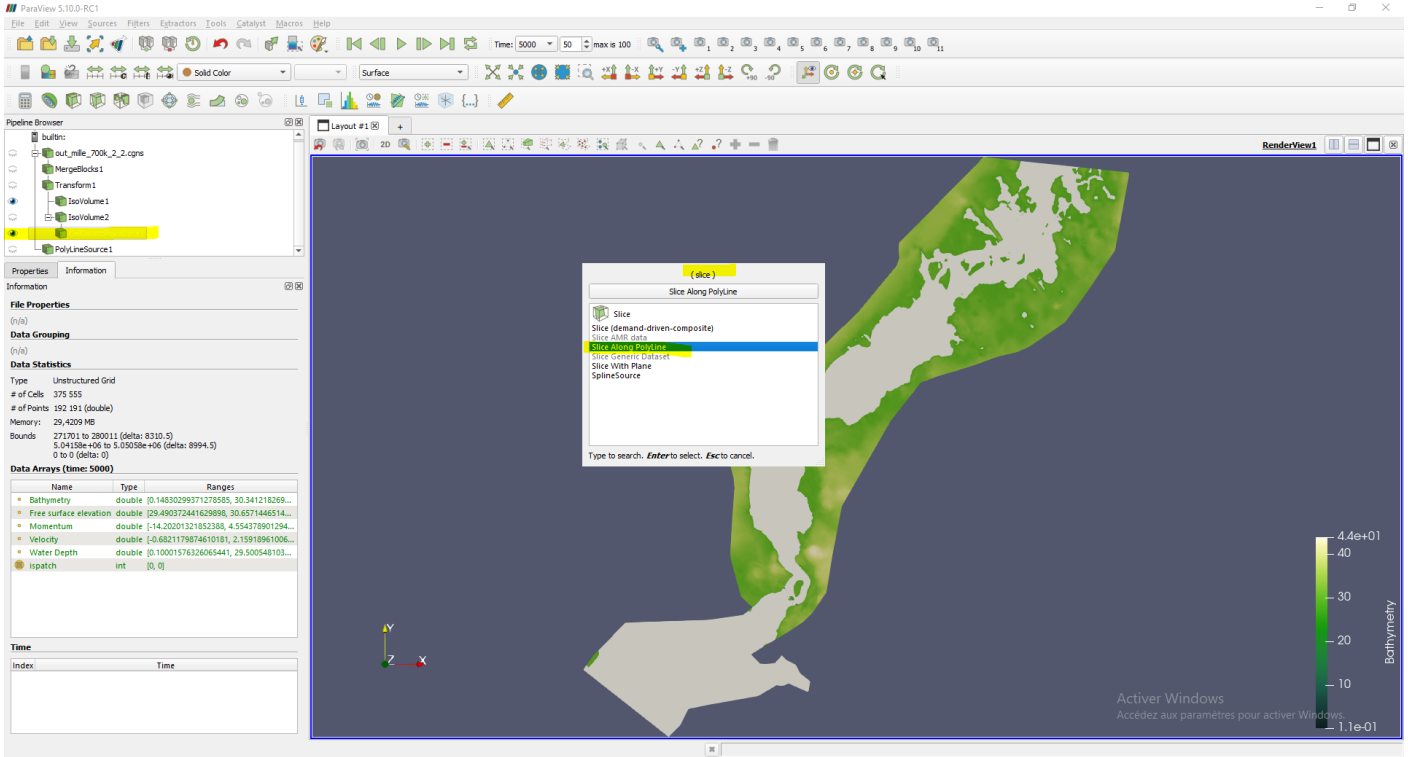


Figure 70: Selection of the *Slice along polyline* tool to project the solution onto the beveled line.

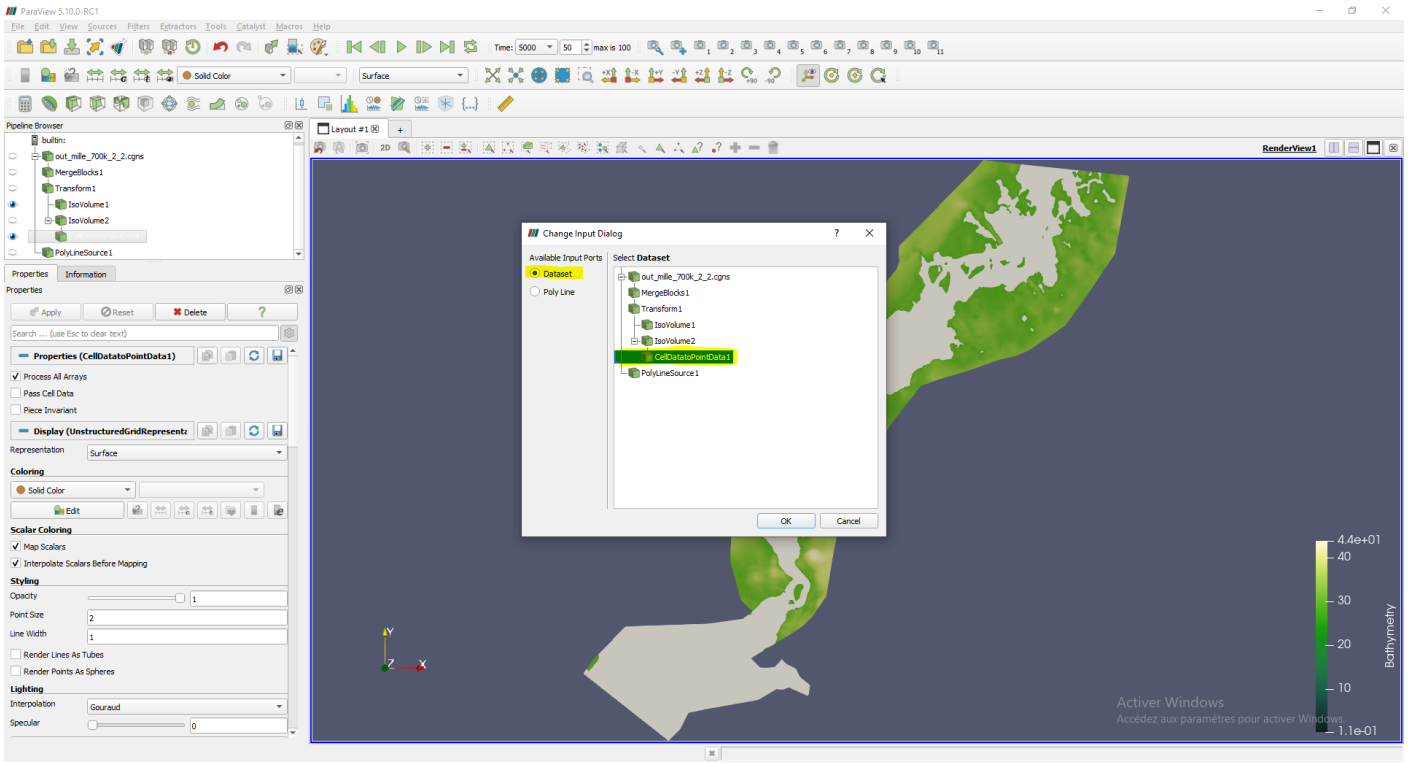


Figure 71: Check that the selected *Dataset* is indeed *CellDataToPointData1*.

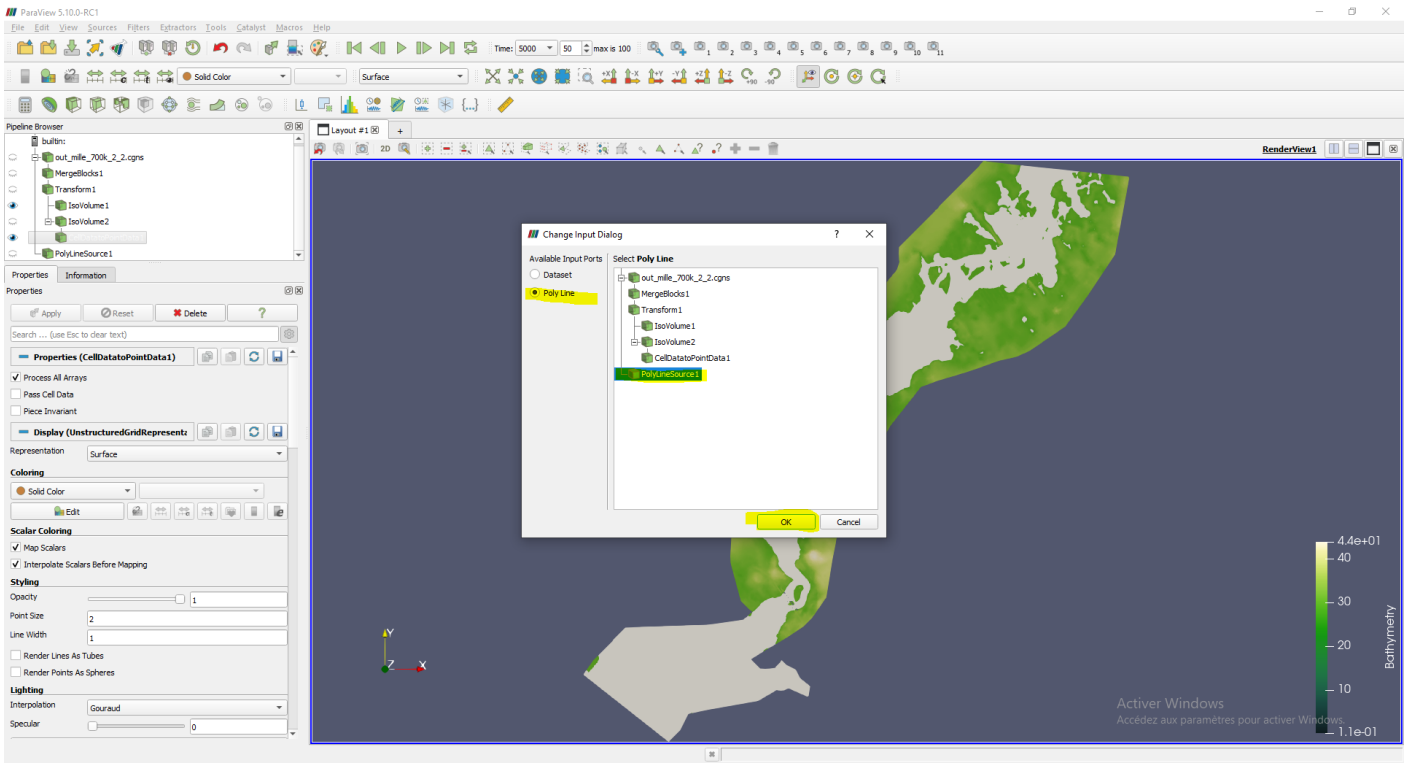


Figure 72: Check that the selected *Polyline* is indeed *PolylineSource1*. Then select *ok* then *Apply* to apply the changes.

If you want to use this line again to, for example, compare two calculation methods, you can click on the save icon located in the "properties" category above the coordinates of the points. So, when we want to use this line again, the *Polyline* tool will have saved the coordinates and will use them as default values.

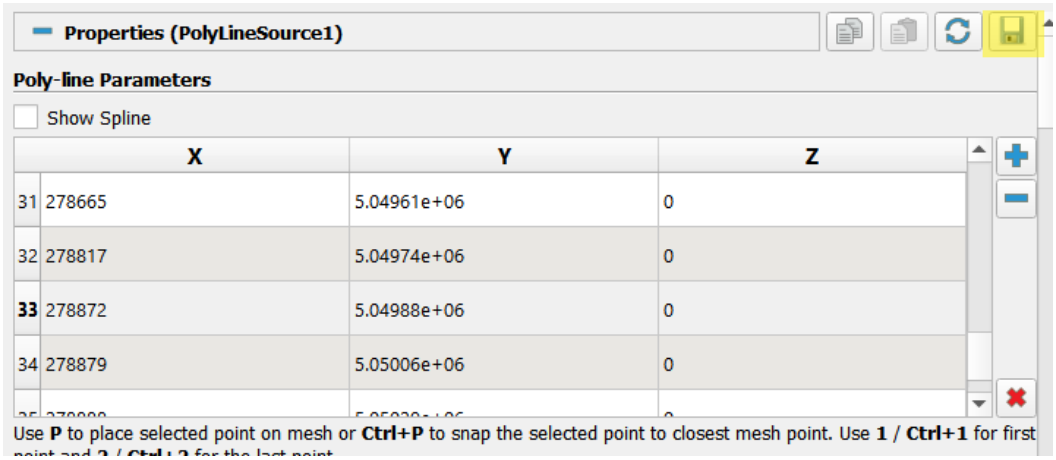


Figure 73: Saving properties of *Polyline*

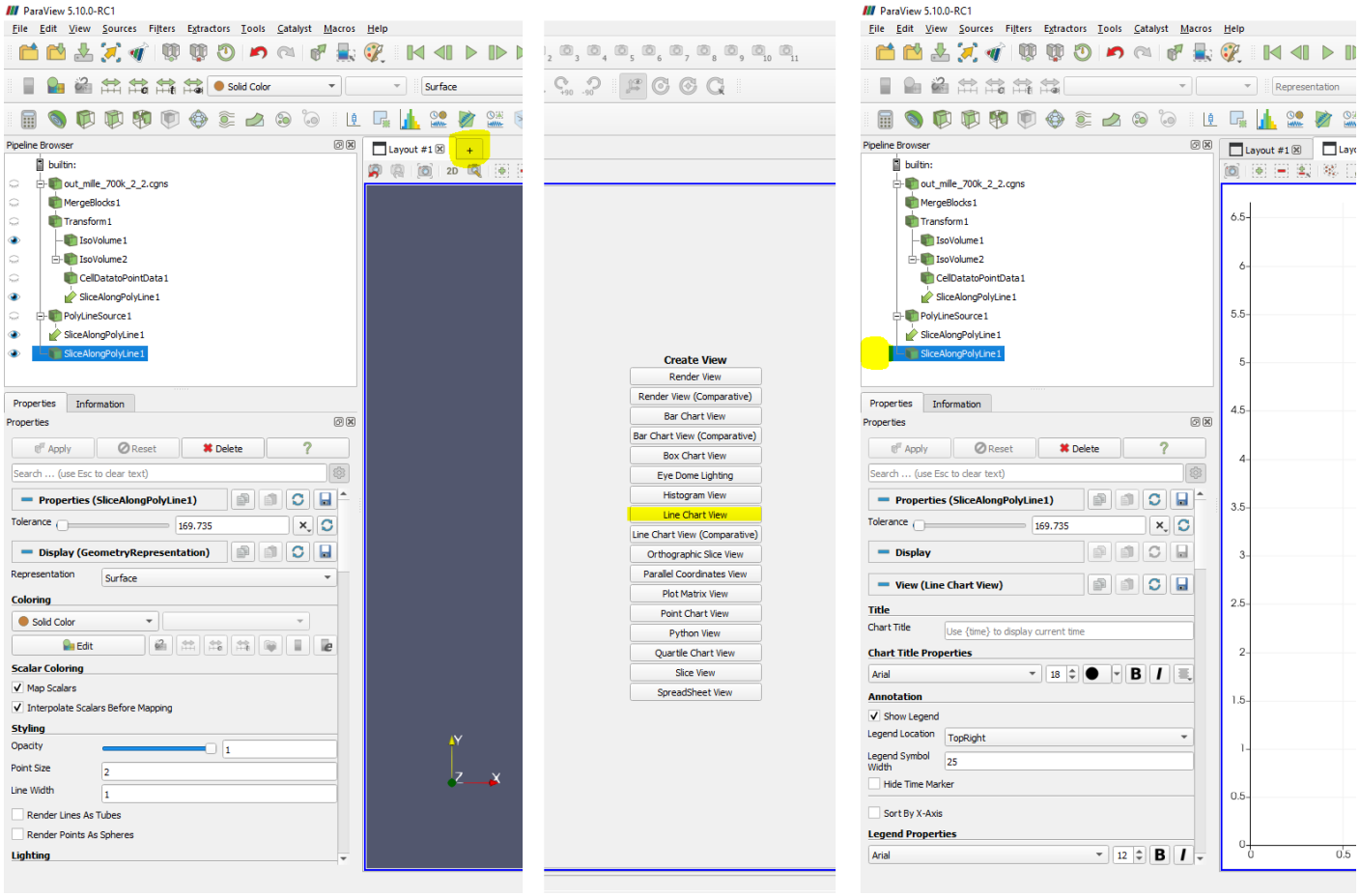


Figure 74: Opening a new viewing window, selecting *Line Chart View* mode and selecting the object to view (click on the blue eye located to the left of the desired object).

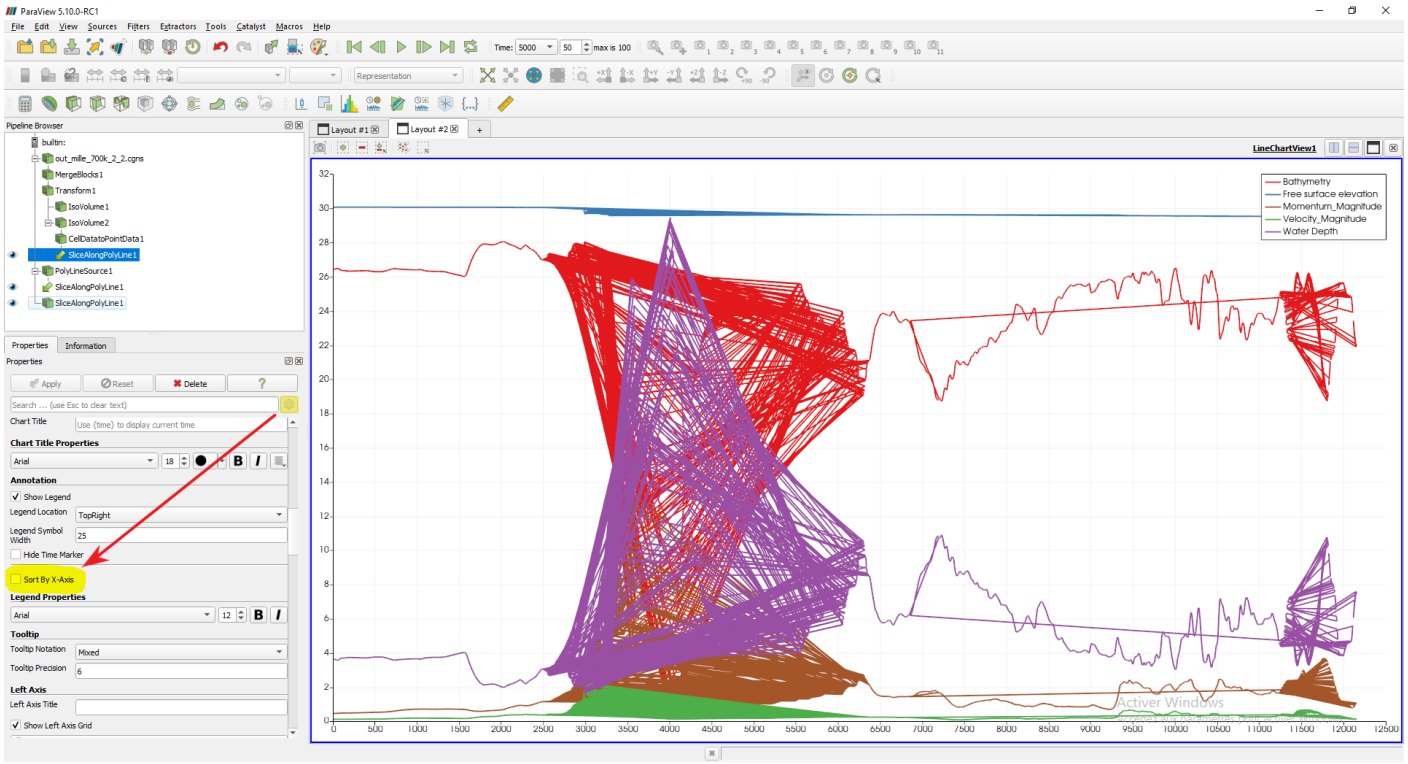


Figure 75: Selection of advanced parameter mode then selection of the *Sort by X-Axis* function.

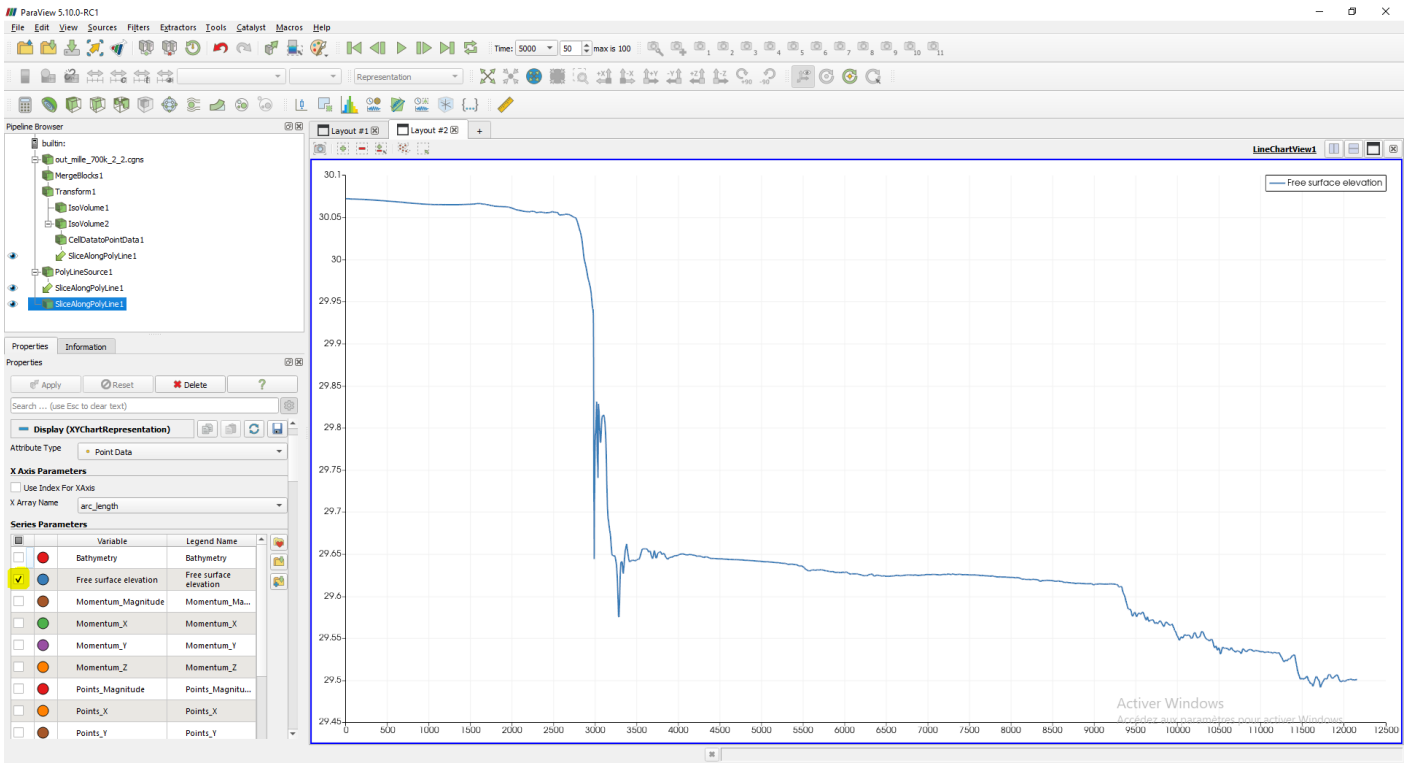


Figure 76: Selection of the attributes of the solution to visualize.

Now we will see how to display flood lines. The approach is similar to the previous one, but a few points differ. Here is the procedure to follow, the details of the tools *Transform* and *IsoVolume* as well as the final tree structure are detailed in figure 77.

- opening the file $out\_mille\_700k\_2\_2.cgns$
- use of *MergeBlocks* to merge subdomains
- use of *Transform* to flatten the domain, and select *Bathymetry* in *Coloring* to colour the ground.
- use of *CellDatatoPointData*

- use of *IsoVolume* to colour the water.
- Use Contour on the CellDatatoPointData to display the interface between the water and the shore.

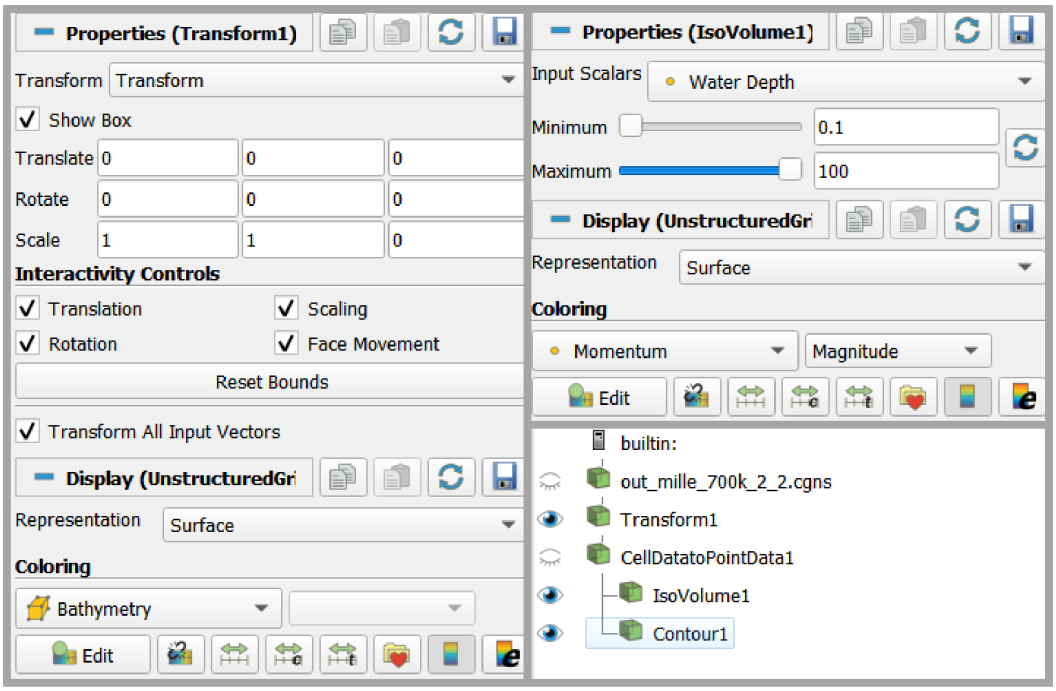


Figure 77: Detail of *Transform*, *Isovolume* and the tree structure.

If we repeat the operation with a different order (or a different flow rate), we can display the two flood lines on the same domain. We then obtain a result similar to this one (the red box is a zoom on an area representative of the difference in water height):

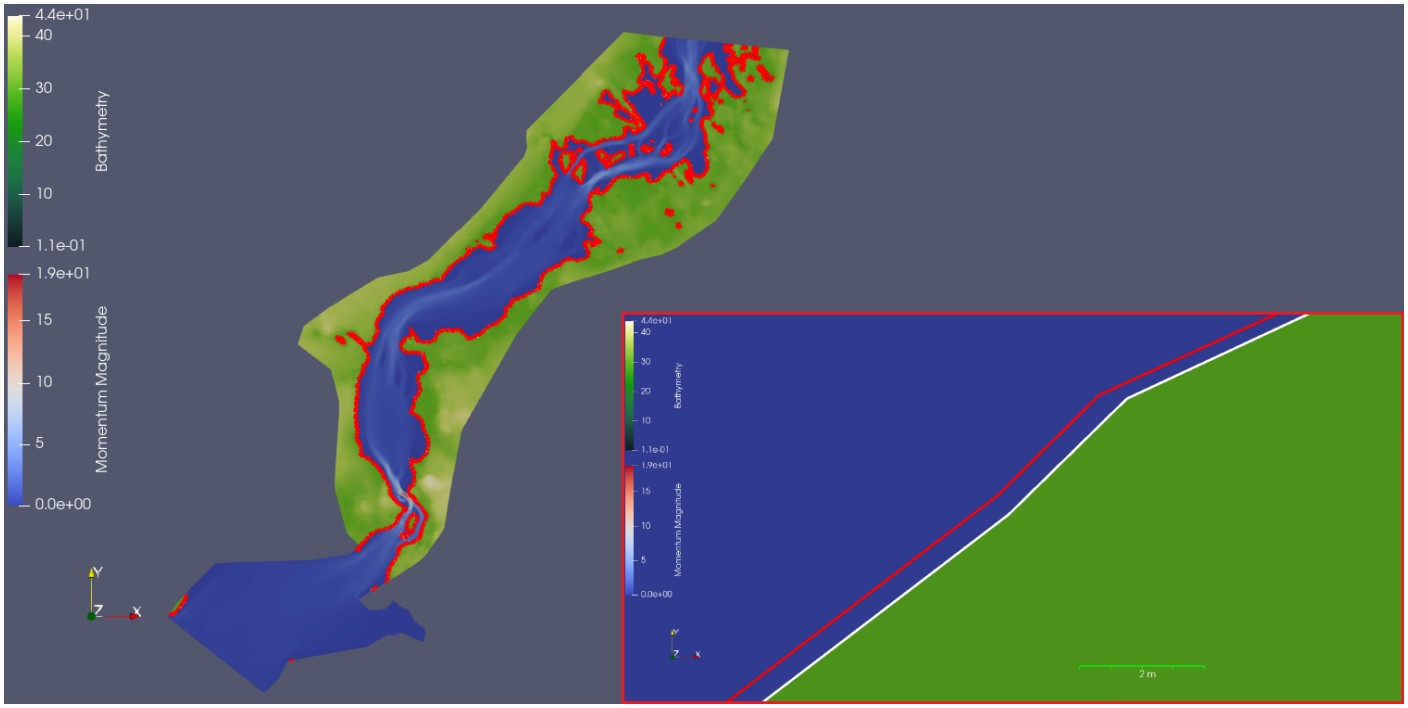


Figure 78: Detail of *Isovolume*, *Transform* and the tree structure.

5 Test Cases

In this section we present several test cases which are used to evaluate the capabilities of *CuteFlow*. Each subsection will aim to compare solutions of different orders with different initial conditions. The files that will be used in this section are located in the folder CuteFlow/docs/.

5.1 Fictional dam breakage cases

First, we will use the case of a fictitious one-dimensional dam break. The initial conditions that we will use for the different test cases are presented in the table 5.1. The subscripts "L" and "R" refer to the left and right part of the domain, respectively. These initial conditions are based on the article [8].

	$h_L[m]$	$u_L[m.s^{-1}]$	$b_L[m]$	$h_R[m]$	$u_R[m.s^{-1}]$	$b_R[m]$
Test 1	1	0	0	0	0	0
Test 2	0.51	2.5	0	0.48	-5.8	0
Test 3	1	-3	0	1	3	0

Figure 79: Initial conditions for the case of a one-dimensional dam break

5.1.1 Exact and reference solutions

The Saint-Venant equations and their numerical resolution using a time-explicit finite volume method are presented below. The notations used are presented in figure 80. This section is based on section 6 of the article [4].

The equations can be written as follows:

$$U_t + G(U)_x + H(U)_y = S(U),$$

with

$$U = \begin{bmatrix} h \\ h\bar{u} \\ h\bar{v} \end{bmatrix}, \quad G(U) = \begin{bmatrix} h\bar{u} \\ h\bar{u}^2 + \frac{1}{2}gh^2 \\ h\bar{u}\bar{v} \end{bmatrix},$$

$$H(U) = \begin{bmatrix} h\bar{v} \\ h\bar{u}\bar{v} \\ h\bar{v}^2 + \frac{1}{2}gh^2 \end{bmatrix}, \quad S(U) = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}.$$

where $\bar{u}$ and $\bar{v}$ are depth-averaged velocities in the x and y directions, h is the height of the water column as defined in figure 80, and h is the gravitational acceleration.

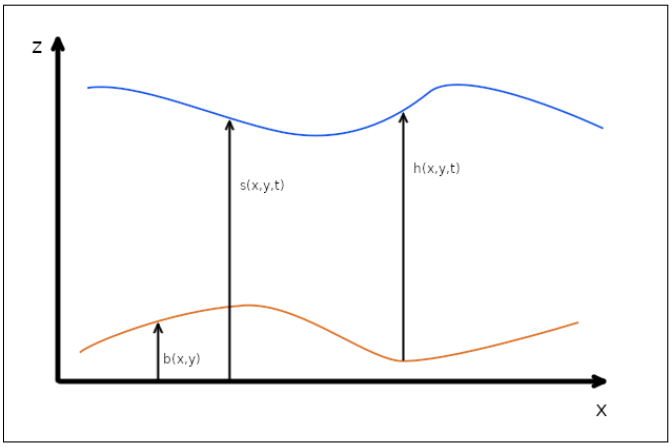


Figure 80: Illustration of ratings

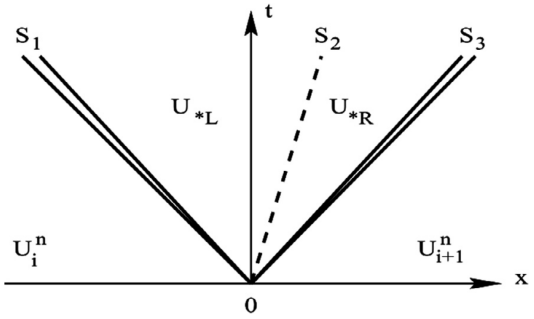


Figure 81: x–t diagram for the solution of the Riemann problem defined by the left (Ui) and right (Uj) states ([2])

When possible, the exact solution is calculated via an annex Python code, otherwise, a reference solution is generated using a mesh file of 37 million elements.

5.1.2 Simulation on mesh_6138 - Test 1

We simulate a dam break on a rectangular area 10 meters long with a water height of 1m on the left and a dry bed on the right.

$$\begin{cases} h = \begin{cases} 1 \; si \; x < 5m \\ 0 \; si \; x > 5m \end{cases} [m], \\ \bar{u} = 0 \; [m/s]. \end{cases}$$

Here we use a mesh containing 6138 triangular cells presented in Figure 82. Figure 5.1.2 presents the evolution of the water height h and Figure 5.1.2 the evolution of the speed u. We take advantage of the fact that we have the exact solution for this test to compare it to the other solutions (order 1 and 2 on a mesh with 6138 elements) and the reference solution (order 1 on a mesh with 37M elements).

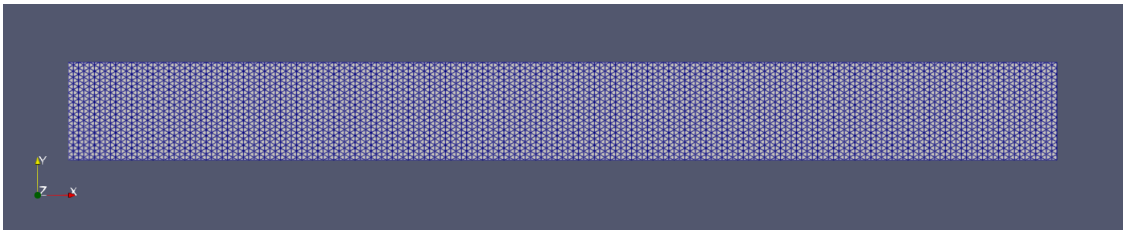
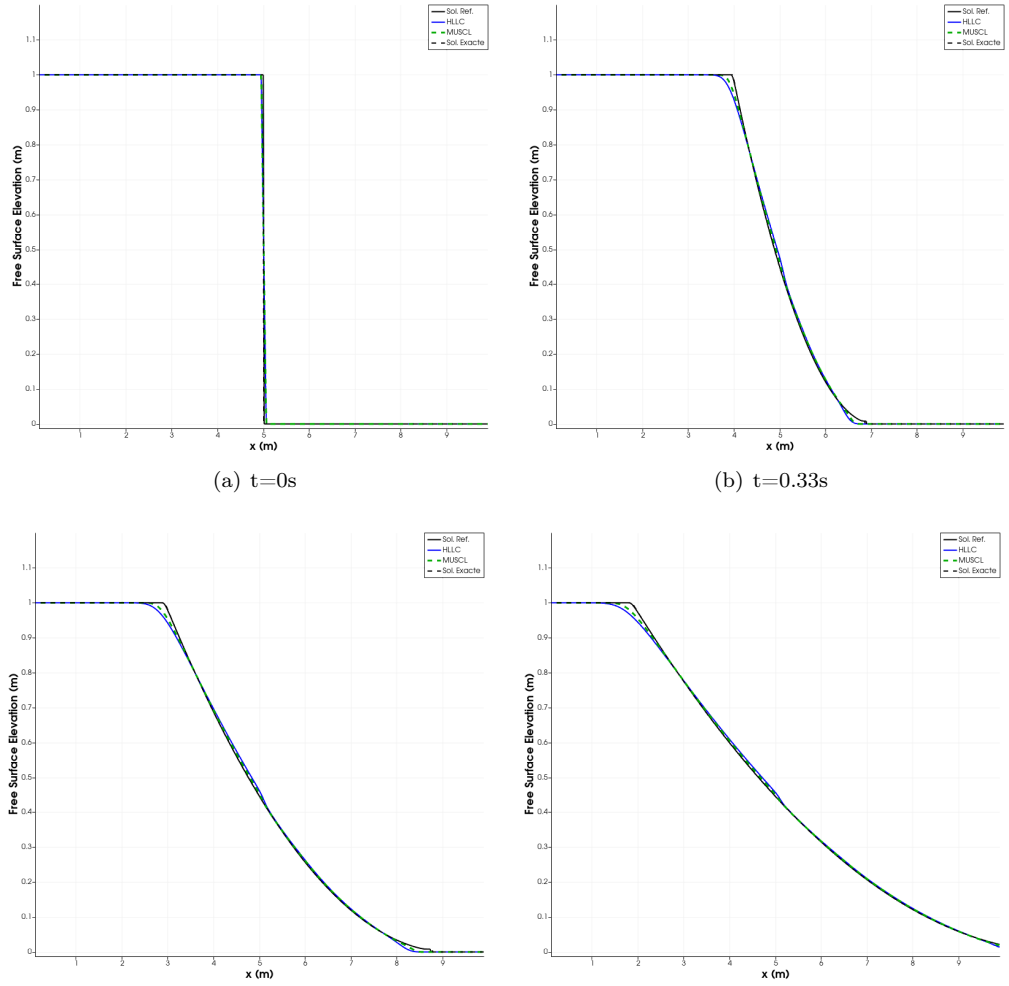


Figure 82: Mesh of 6138 elements



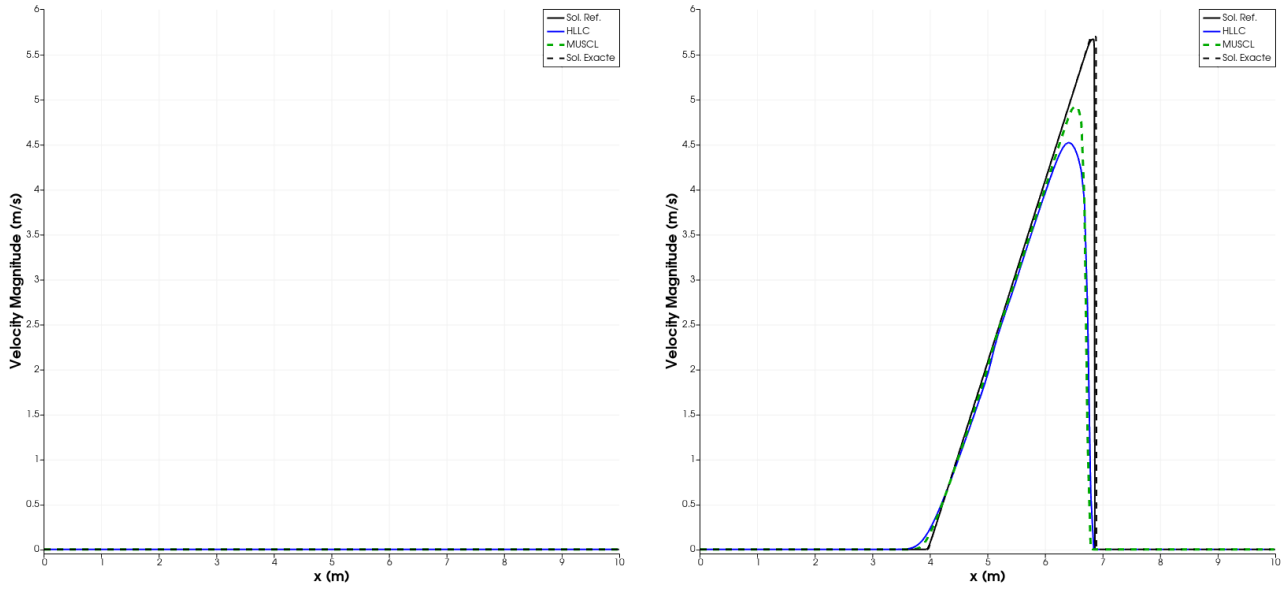
(a) t=0s

(b) t=0.33s

(c) t=0.66s

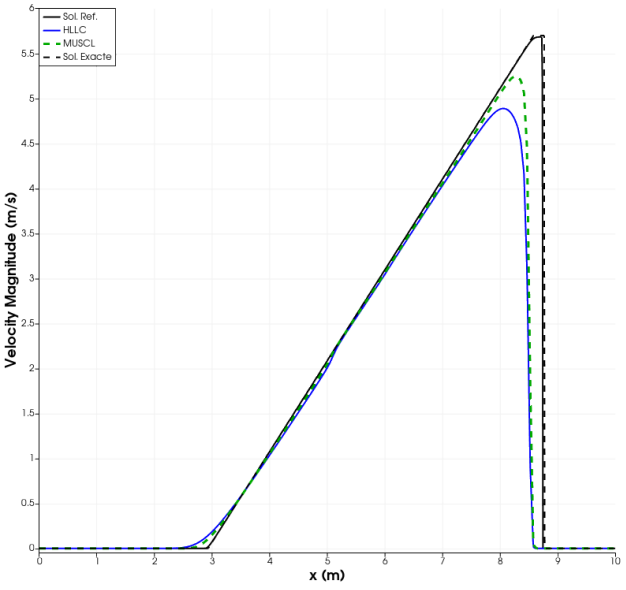
(d) t=1s

Figure 83: Solutions projected on the y axis for test 1, one-dimensional dam break on a mesh of 6138 cells, reference solution on a mesh of 37 million cells at t = 0, 0.33, 0.66 and 1 seconds (Free Surface Elevation) and exact solution.

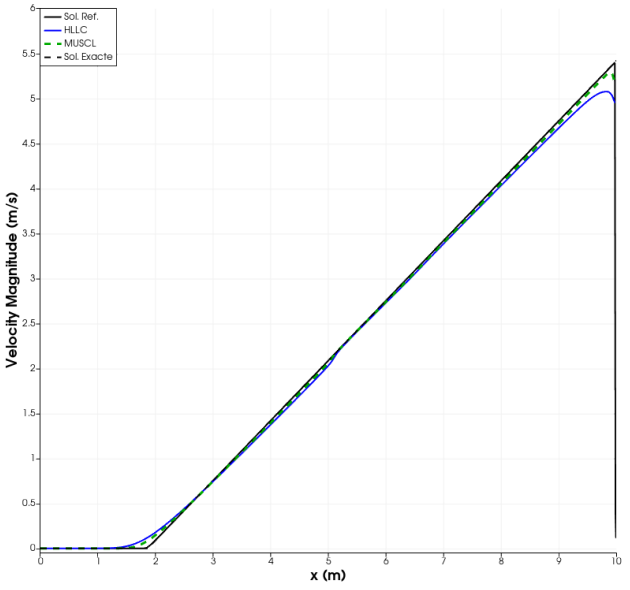


(a) t=0s

(b) t=0.33s



(c) t=0.66s



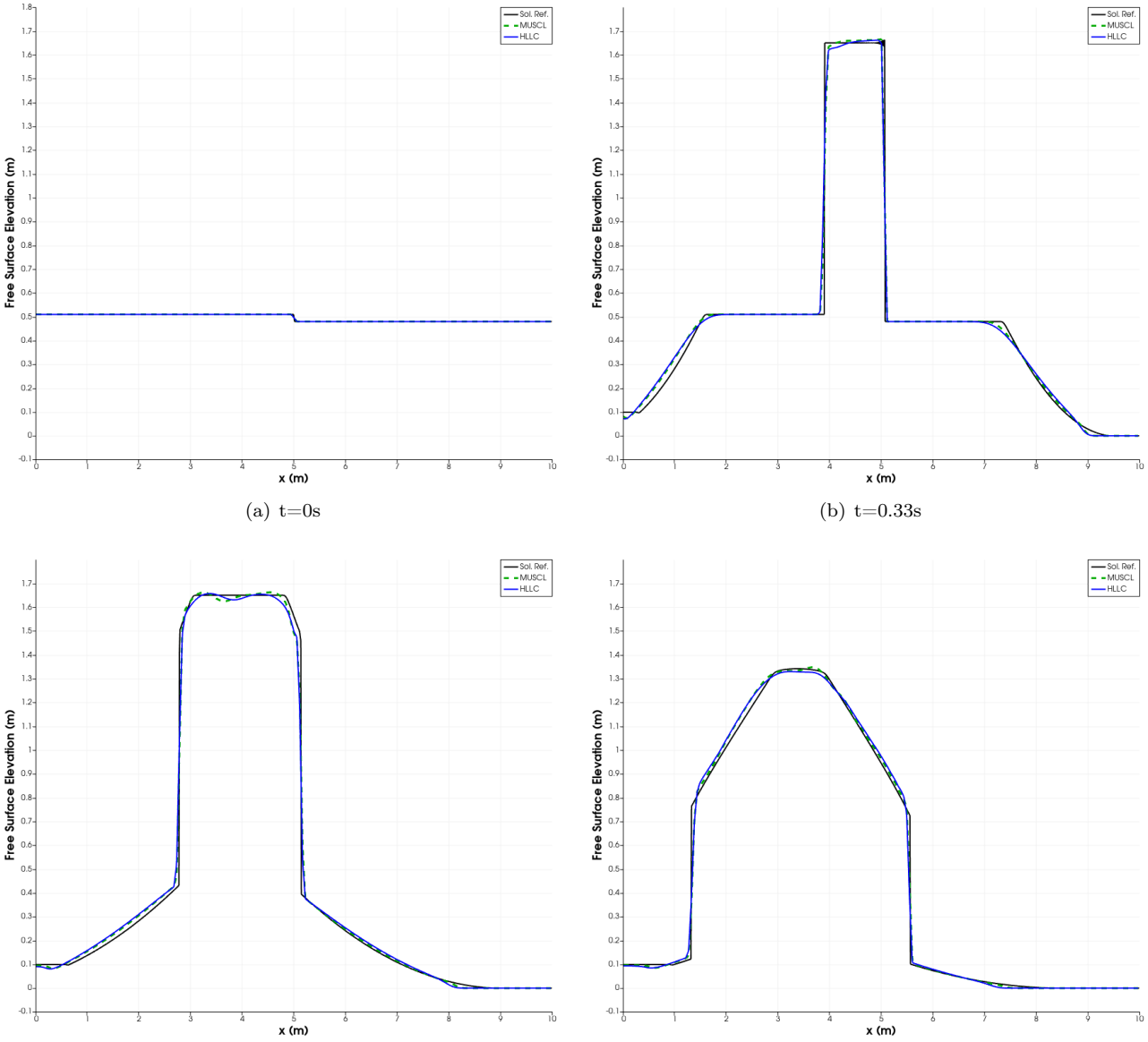
(d) t=1s

Figure 84: Solutions projected on the y axis for test 1, one-dimensional dam break on a mesh with 6138 cells, reference solution on a mesh with 37 million cells at t = 0, 0.33, 0.66 and 1 seconds (Velocity magnitude) and exact solution.

5.1.3 Simulation on mesh_6138 - Test 2

As a reminder, the initial conditions of this simulation are as follows:

$$\begin{cases} h = \begin{cases} 0.51 \; si \; x < 0.5m \\ 0.48 \; si \; x > 0.5m \end{cases} [m], \\ \bar{u} = \begin{cases} 2.5 \; si \; x < 0.5m \\ -5.8 \; si \; x > 0.5m \end{cases} [m/s]. \end{cases} \tag{1}$$



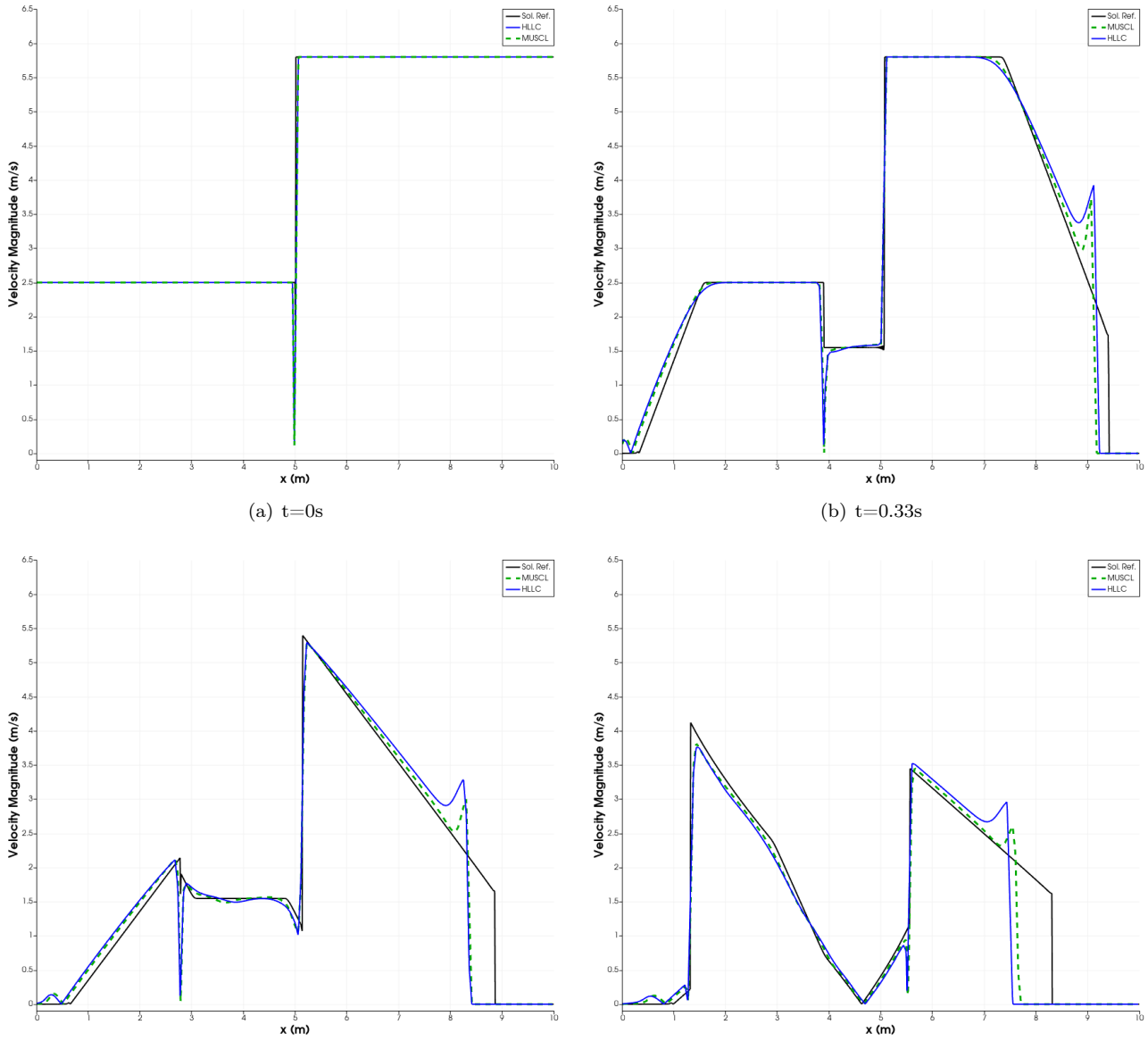
(a) t=0s

(b) t=0.33s

(c) t=0.66s

(d) t=1s

Figure 85: Solutions projected on the y axis for test 3, one-dimensional dam break on a mesh with 6138 cells and reference solution on a mesh with 37 million cells at t = 0, 0.33, 0.66 and 1 seconds (Free Surface Elevation)



(a) t=0s

(b) t=0.33s

(c) t=0.66s

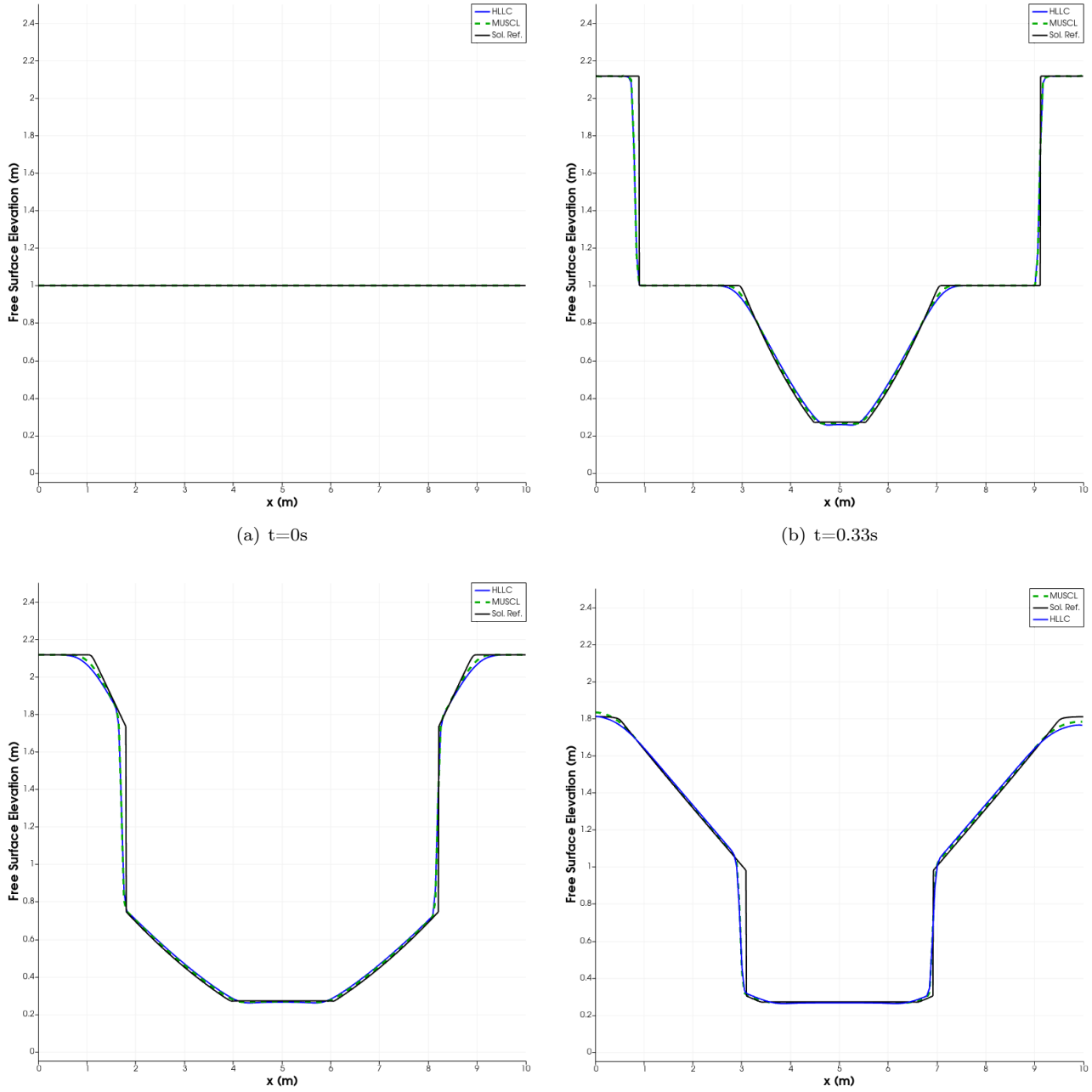
(d) t=1s

Figure 86: Solutions projected on the y axis for test 2, one-dimensional dam break on a mesh with 6138 cells and reference solution on a mesh with 37 million cells at t = 0, 0.33, 0.66 and 1 seconds (Speed according to X)

5.1.4 Simulation on mesh_6138 - Test 3

As a reminder, the initial conditions of this simulation are as follows:

$$\begin{cases} h = \begin{cases} 1 \; si \; x < 0.5m \\ 1 \; si \; x > 0.5m \end{cases} [m], \\ \bar{u} = \begin{cases} -3 \; si \; x < 0.5m \\ 3 \; si \; x > 0.5m \end{cases} [m/s]. \end{cases} \tag{2}$$



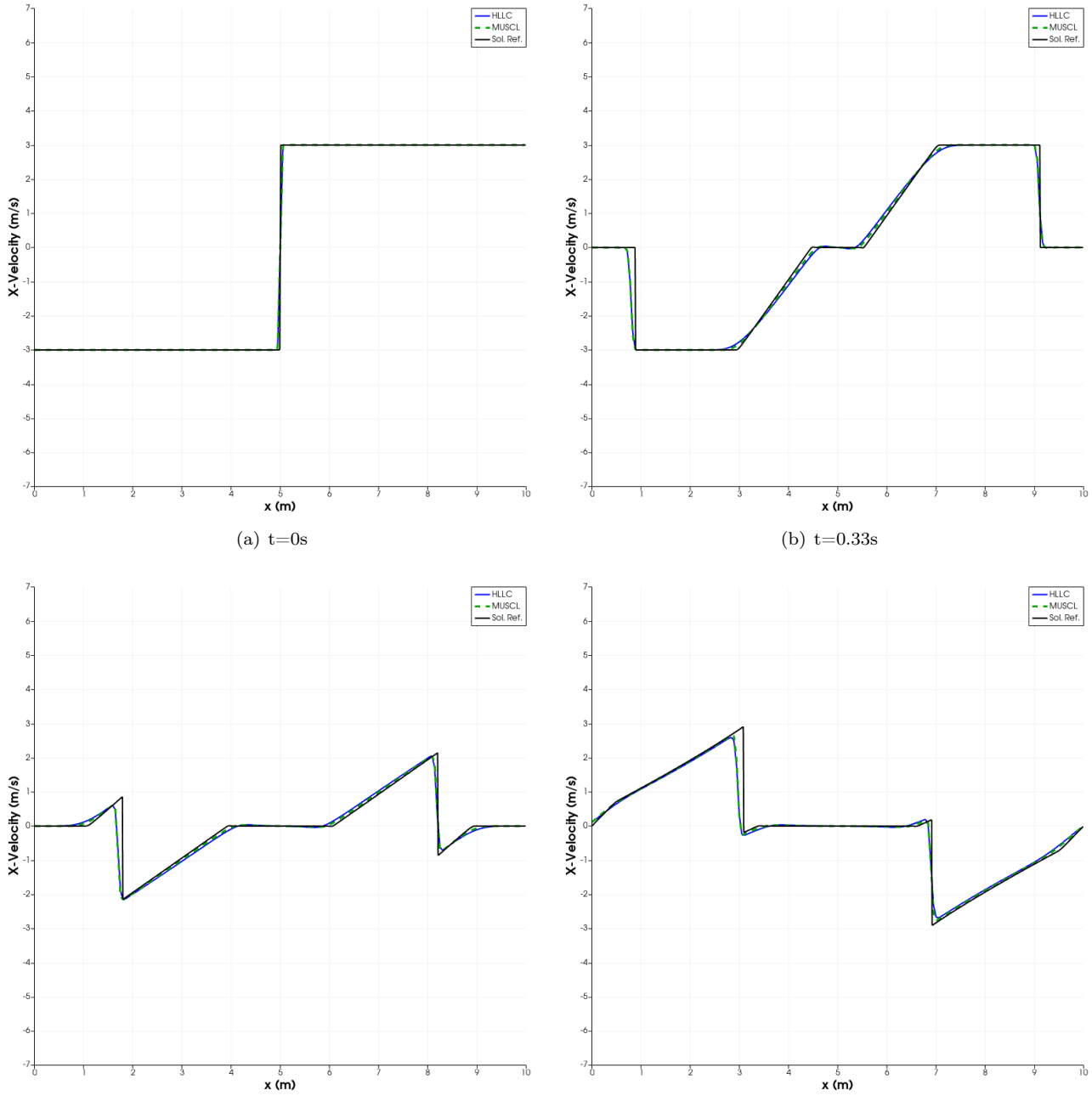
(a) t=0s

(b) t=0.33s

(c) t=0.66s

(d) t=1s

Figure 87: Solutions projected on the y axis for test 3, one-dimensional dam break on a mesh with 6138 cells and reference solution on a mesh with 37 million cells at t = 0, 0.33, 0.66 and 1 seconds (Free Surface Elevation)



(a) t=0s

(b) t=0.33s

(c) t=0.66s

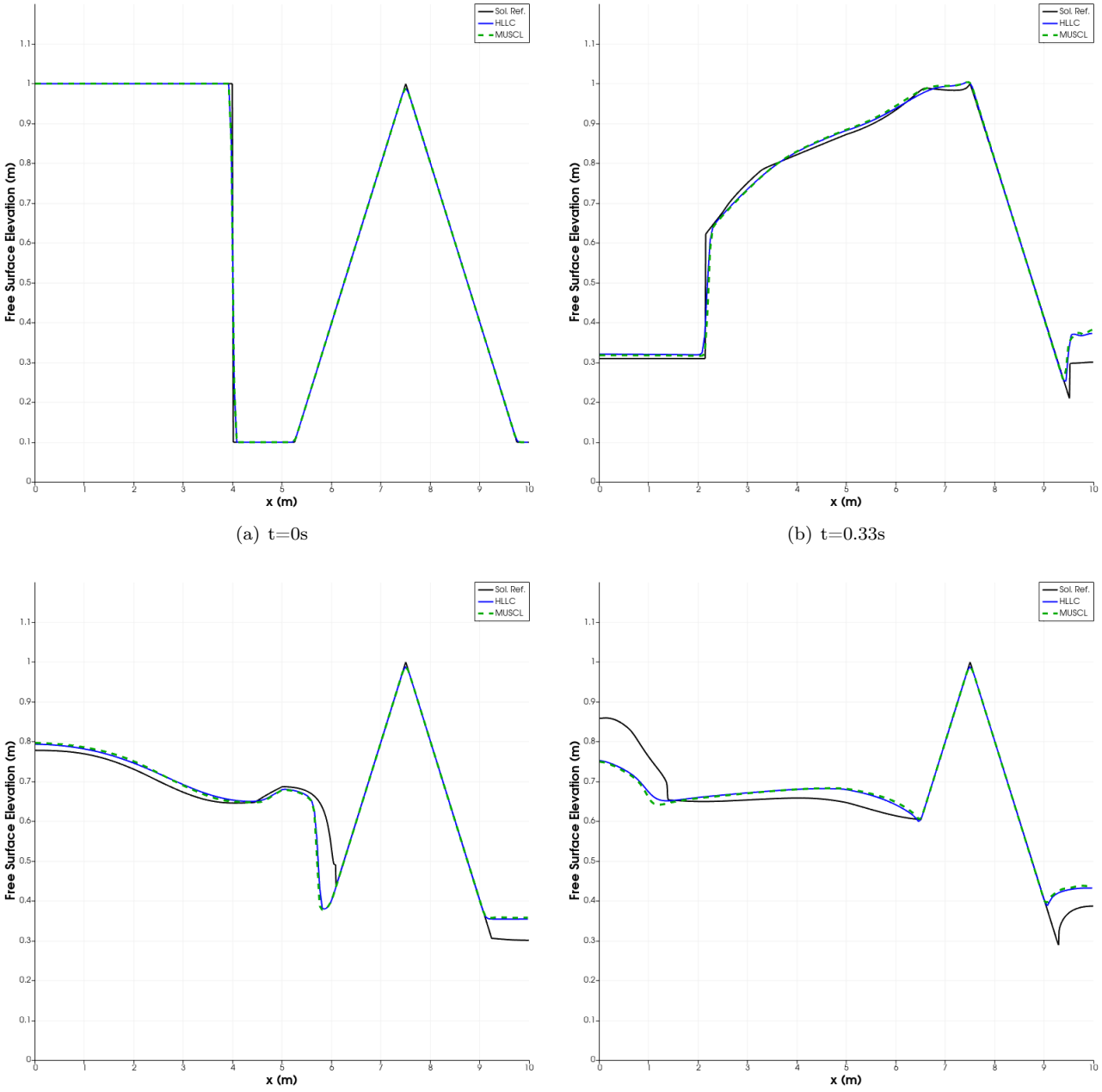
(d) t=1s

Figure 88: Solutions projected on the y axis for test 3, one-dimensional dam break on a mesh with 6138 cells and reference solution on a mesh with 37 million cells at t = 0, 0.33, 0.66 and 1 seconds (Speed according to X)

5.2 Case of fictitious dam breakage with bump

We simulate a dam break on a rectangular area 10 meters long with a water height of 1m on the left and 0.1m on the right. This time the area is not flat but has a "bump" on the right. The reference solution is calculated on a mesh of 37 million cells.

$$\begin{cases} h = \begin{cases} 1 \; si \; x < 4m \\ 0.1 \; si \; x > 4m \end{cases} [m], \\ \bar{u} = 0 \; [m/s]. \end{cases} \tag{3}$$



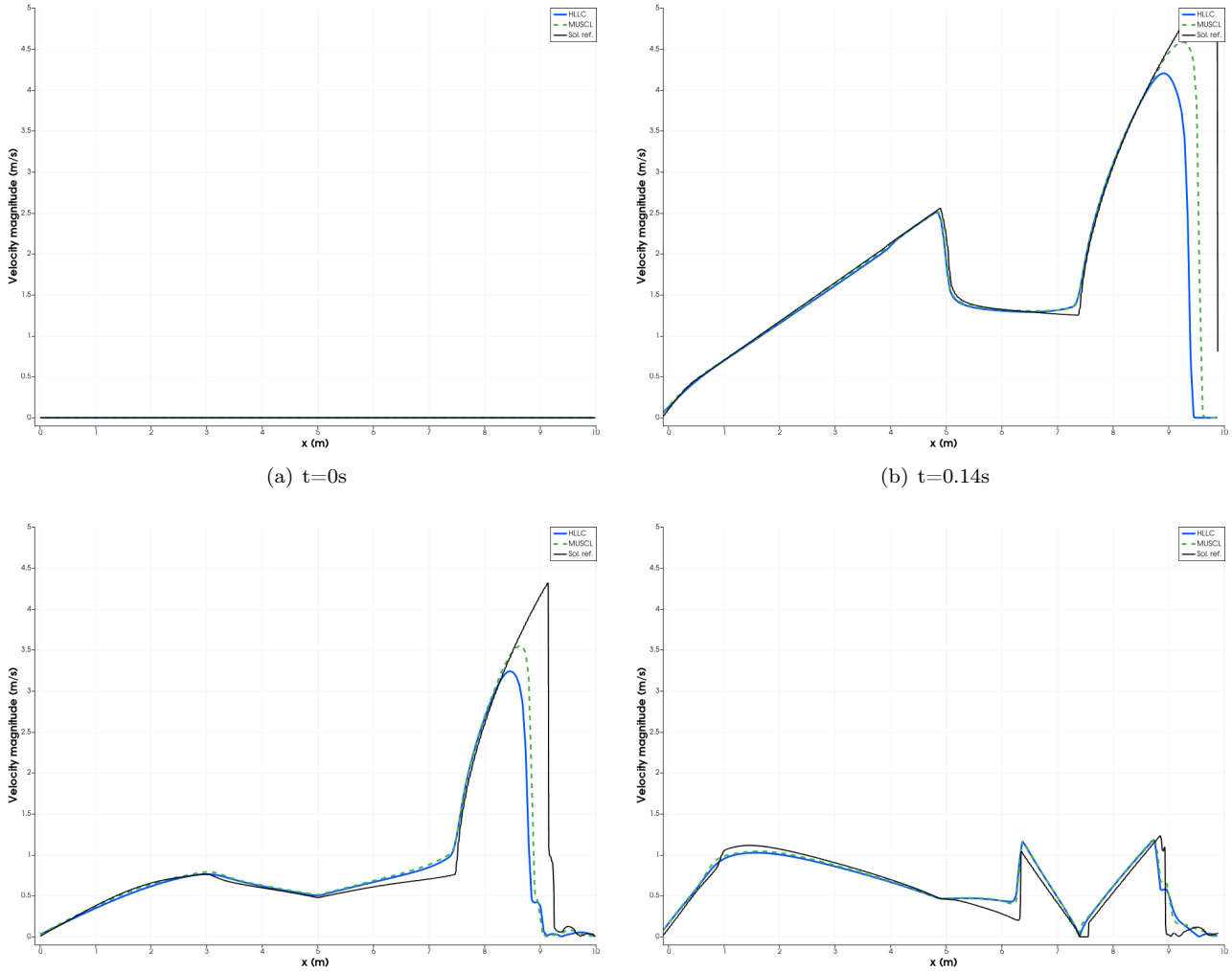
(a) t=0s

(b) t=0.33s

(c) t=0.66s

(d) t=1s

Figure 89: Solutions projected on the y axis for one-dimensional dam breaking on a mesh with 6138 cells and reference solution on a mesh with 37M cells at t = 0, 0.33, 0.66 and 1 seconds (FSE)



(a) t=0s

(b) t=0.14s

(c) t=0.77s

(d) t=1s

Figure 90: Solutions projected on the y axis for one-dimensional dam breaking on a 6138 cell mesh and reference solution on a 95459 cell mesh at t = 0, 0.14, 0.77 and 1 seconds (Velocity Magnitude)

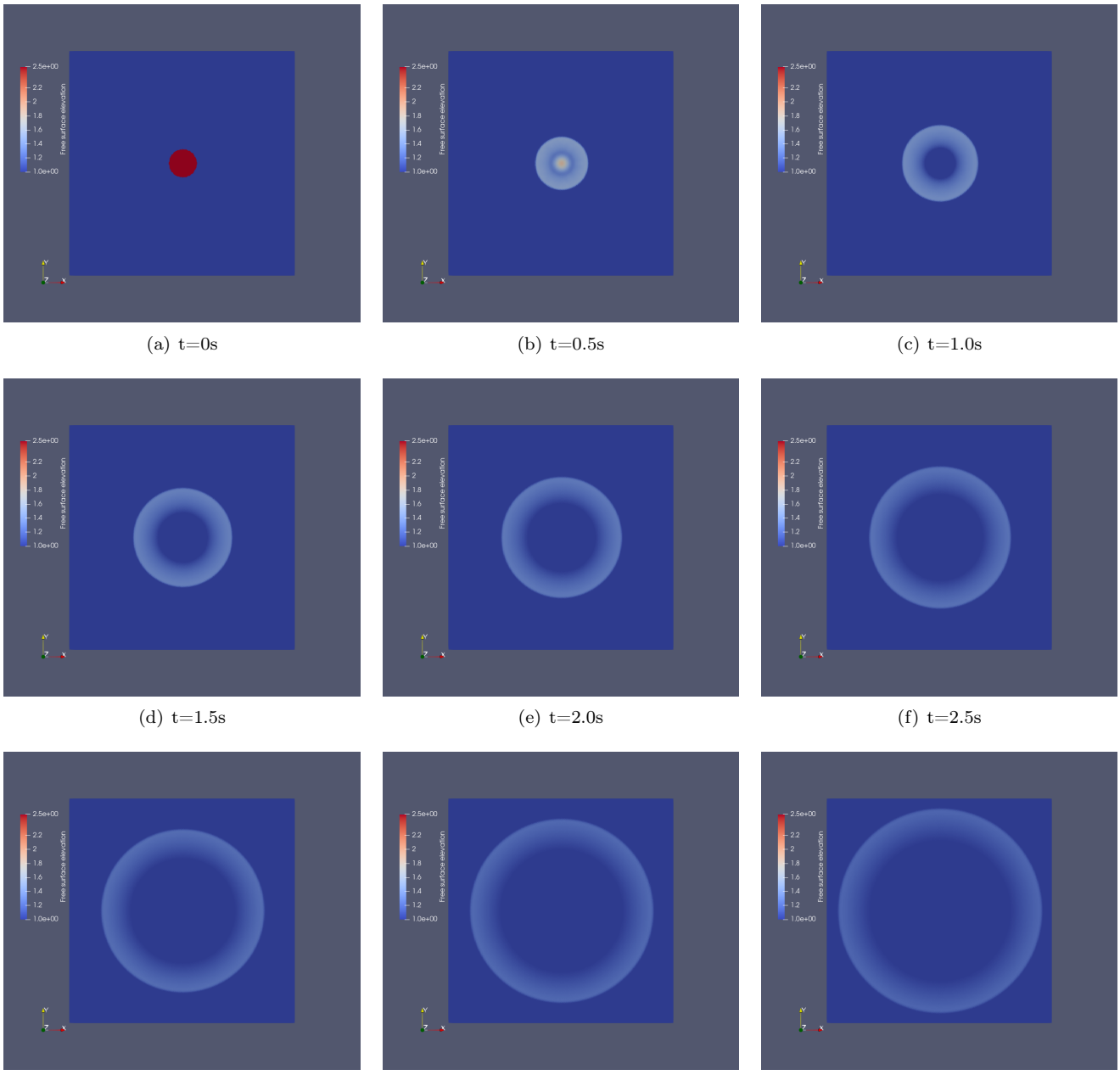
5.3 Case of circular dam breakage

In this section, we present the results for a circular dam break. The area is defined as a square area of 40 meters on each side. The bottom is flat and the interior part of the water circle is of a higher height than the exterior part at the initial time t=0 such that:

$$\begin{cases} h = \begin{cases} h_{in} = 2.5 \\ h_{out} = 1 \end{cases} [m], \\ u = 0, \\ v = 0. \end{cases} \tag{4}$$

The domains "in" and "out" are separated by the circle of equation $r(x, y) = \sqrt{(x-20)^2 + (y-20)^2} = 2.5$, the x and y coordinates corresponding to the sides of the square domain with 0<x<40 and 0<y<40.

The following figure shows the evolution of the water height on an order 2 diagram with a mesh of 90,000 elements.



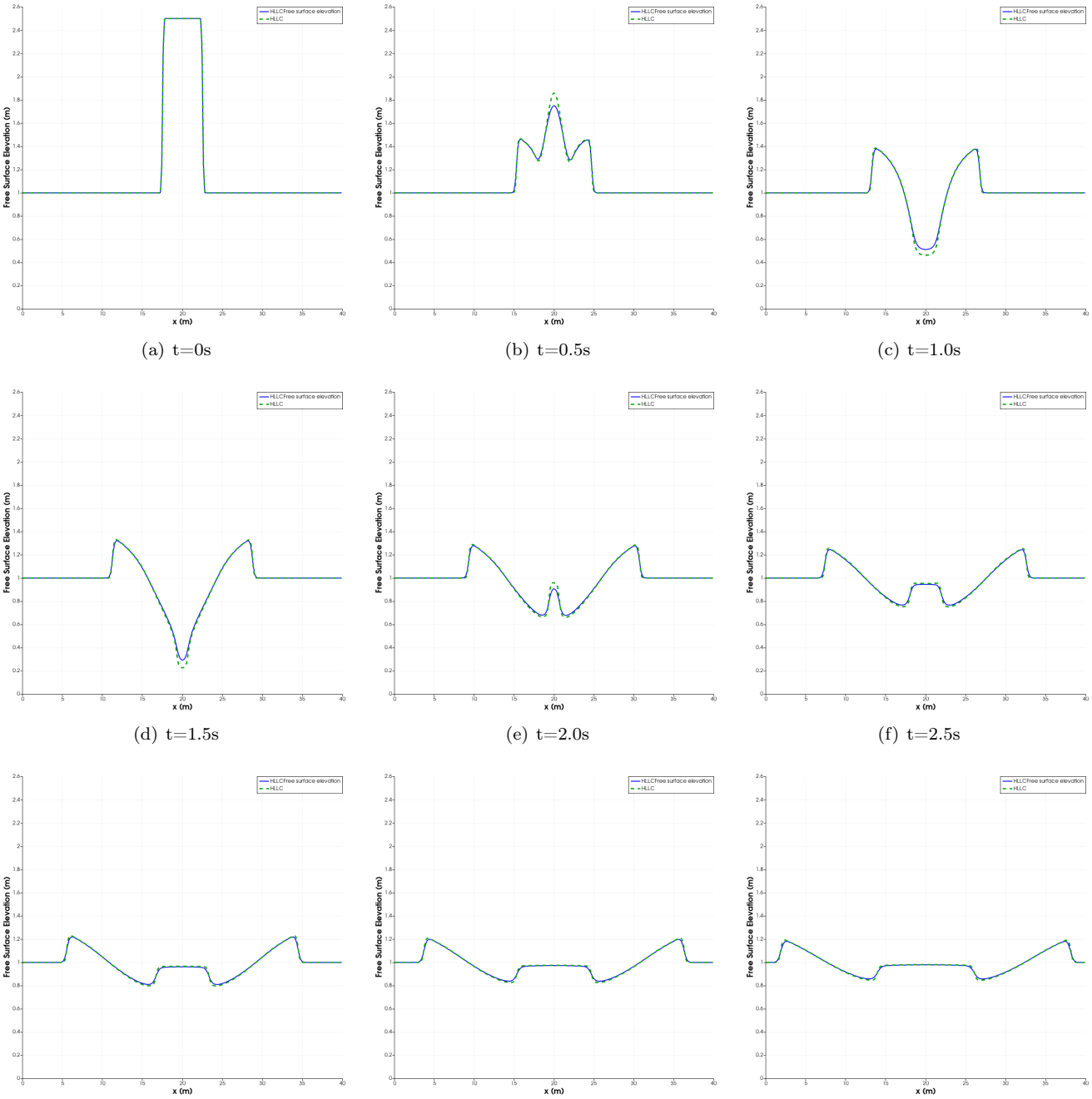
(a) t=0s (b) t=0.5s (c) t=1.0s

(d) t=1.5s (e) t=2.0s (f) t=2.5s

(g) t=3.0s (h) t=3.5s (i) t=4.0s

Figure 91: Évolution de la hauteur d'eau (FSE)

We can then compare the solutions between different orders, here on a mesh of 90,000 elements, the solution is drawn on a line parallel to the sides crossing the domain from one edge to the other passing through its center.

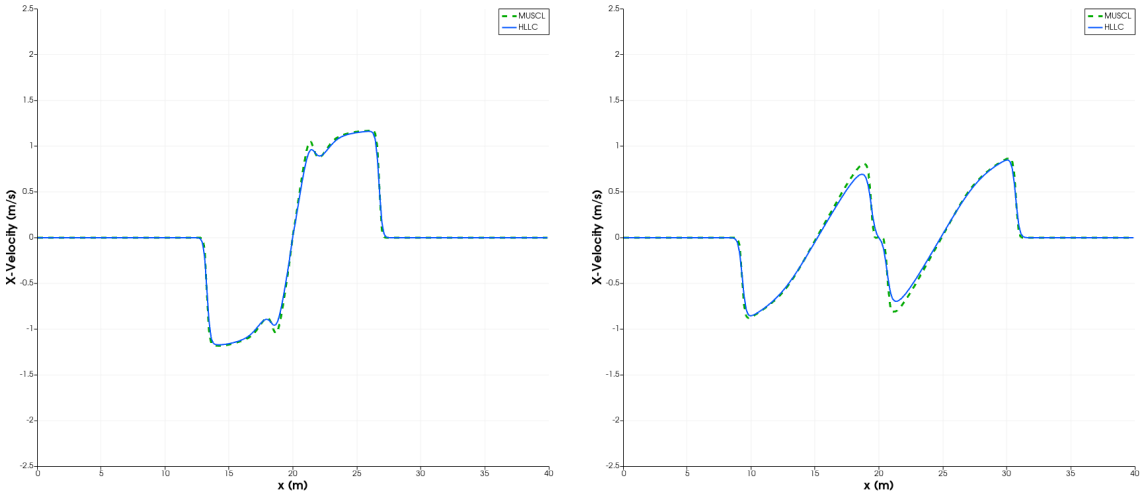


(a) t=0s (b) t=0.5s (c) t=1.0s

(d) t=1.5s (e) t=2.0s (f) t=2.5s

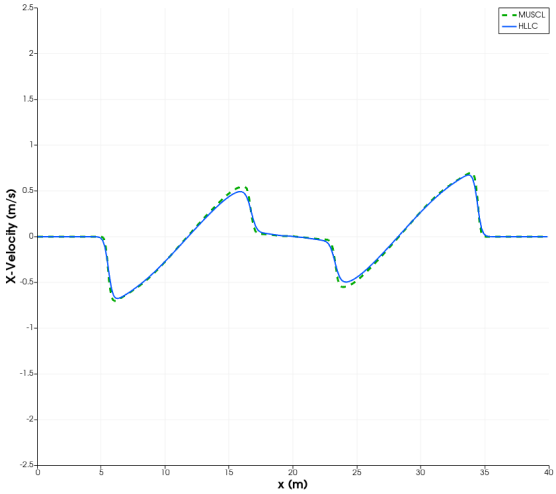
(g) t=3.0s (h) t=3.5s (i) t=4.0s

Figure 92: Evolution of water height h

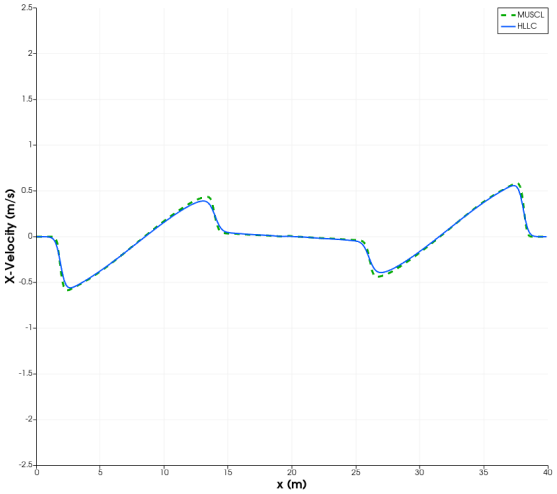


(a) t=1s

(b) t=2s



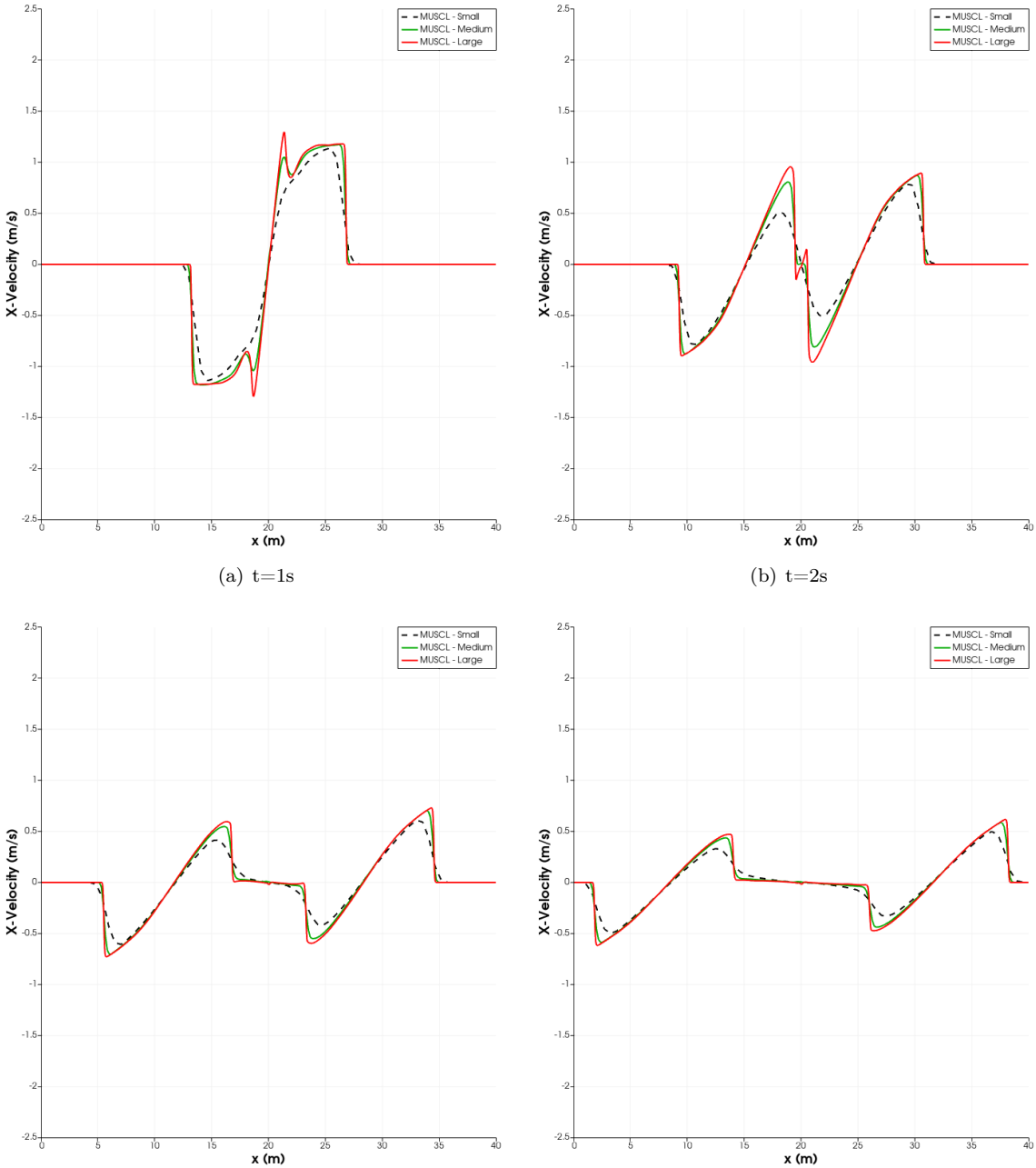
(c) t=3s



(d) t=4s

Figure 93: Comparison between orders 1 and 2 at t = 1, 2, 3 and 4 seconds on a mesh with 90k elements (Speed according to X)

Finally we compare the results obtained with different meshes which are respectively of 9546, 93031 and 947939 elements.



(a) t=1s

(b) t=2s

(c) t=3s

(d) t=4s

Figure 94: Comparison between the different meshes at t = 0, 1, 2 and 4 seconds (Speed according to X)

5.4 Case of the Milles Îles River

In this section, we compare the solutions of order 1 and 2 in the case of the Milles Îles River. We use a mesh modelling the section of the river with 700,000 elements.

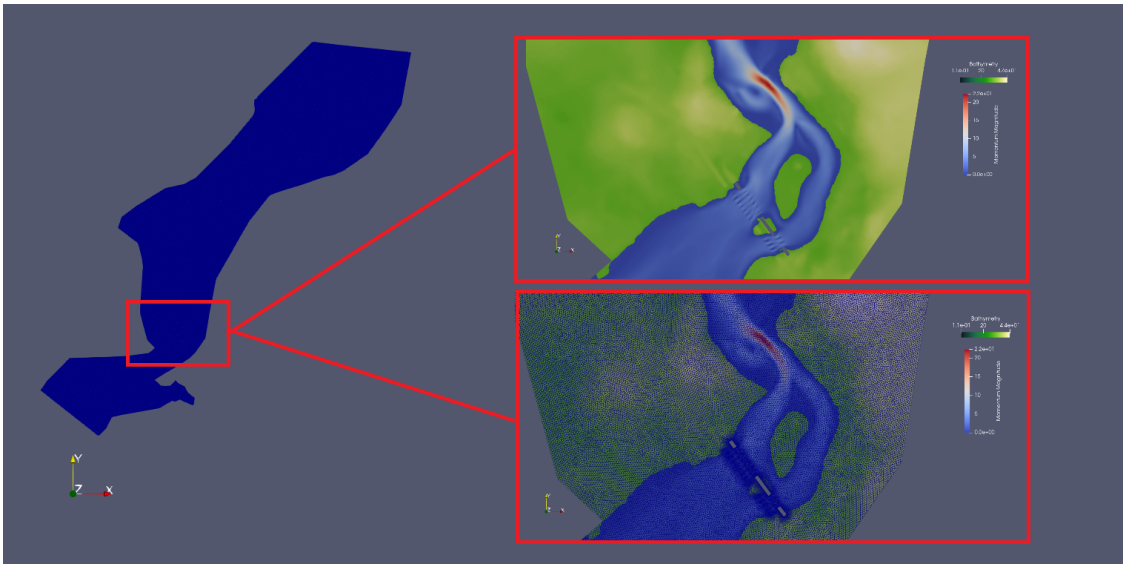
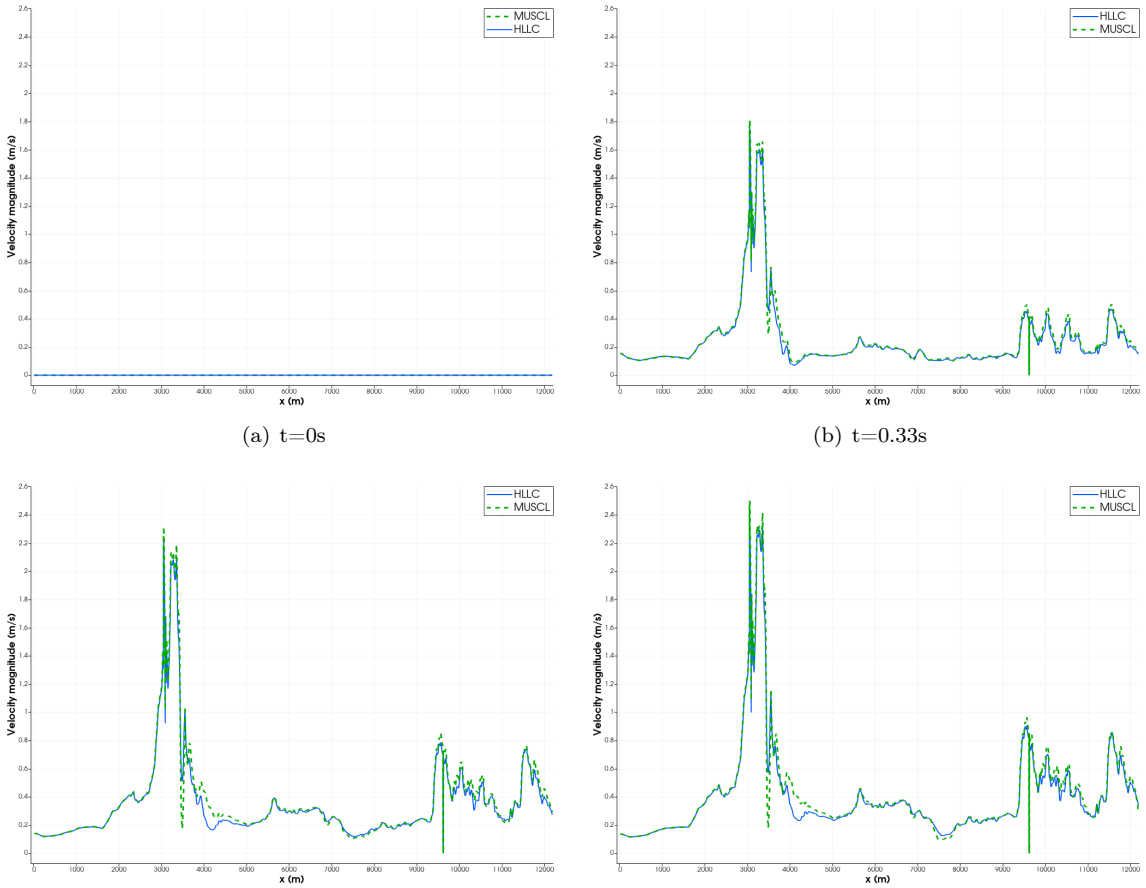


Figure 95: Mesh of 700k elements of the Milles Îles

The solution presented in Figure 96 is projected along a line following the course of the river. The flood lines calculated by the two methods are compared in Figure 97.



(a) t=0s

(b) t=0.33s

(c) t=0.66s

(d) t=1s

Figure 96: Solutions projected along a line for the Milles Îles River on a 700k cell mesh at t = 0, 3300, 6600 and 10000 seconds

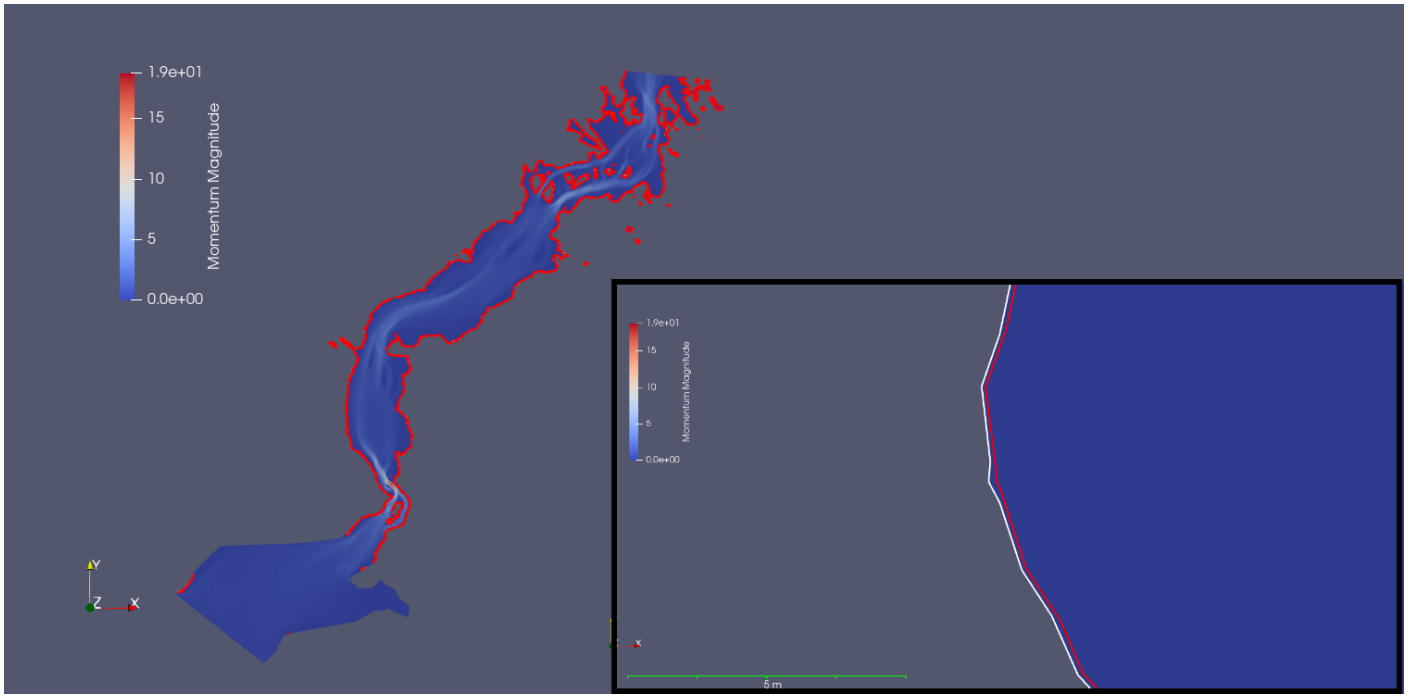


Figure 97: Difference in water height between the HLLC (in white) and MUSCL (in red) methods at 10,000 seconds on a representative wet/dry interface downstream of the Milles Îles River

We now wish to make a comparison between the solutions for a change in the input flow (without changing the output level). Figure 99 presents the flood line differences for three flow rates in the Milles Îles River: 600 $m^3.s^{-1}$, 1000 $m^3.s^{-1}$ and 1400 $m^3.s^{-1}$. The flood lines due to these flows are respectively coloured red, white and blue. The solutions are calculated with the MUSCL method and the study area is presented in Figure 98.

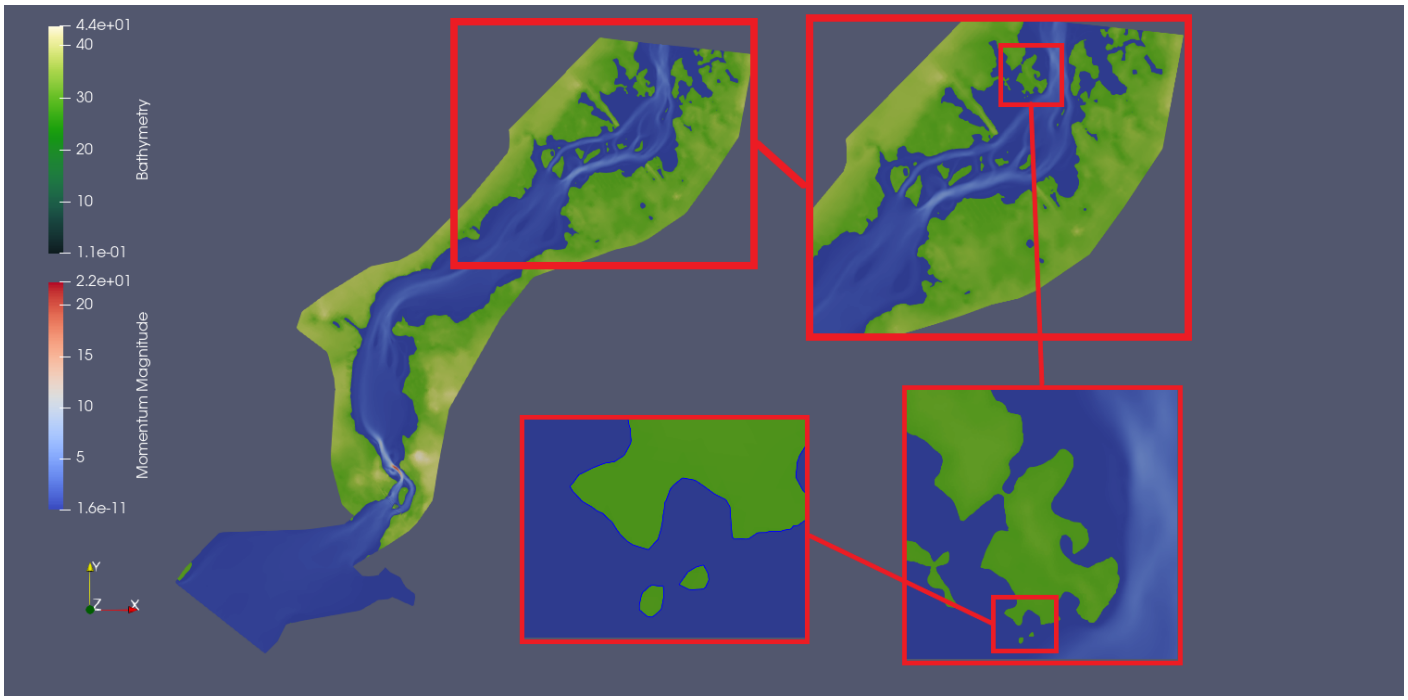
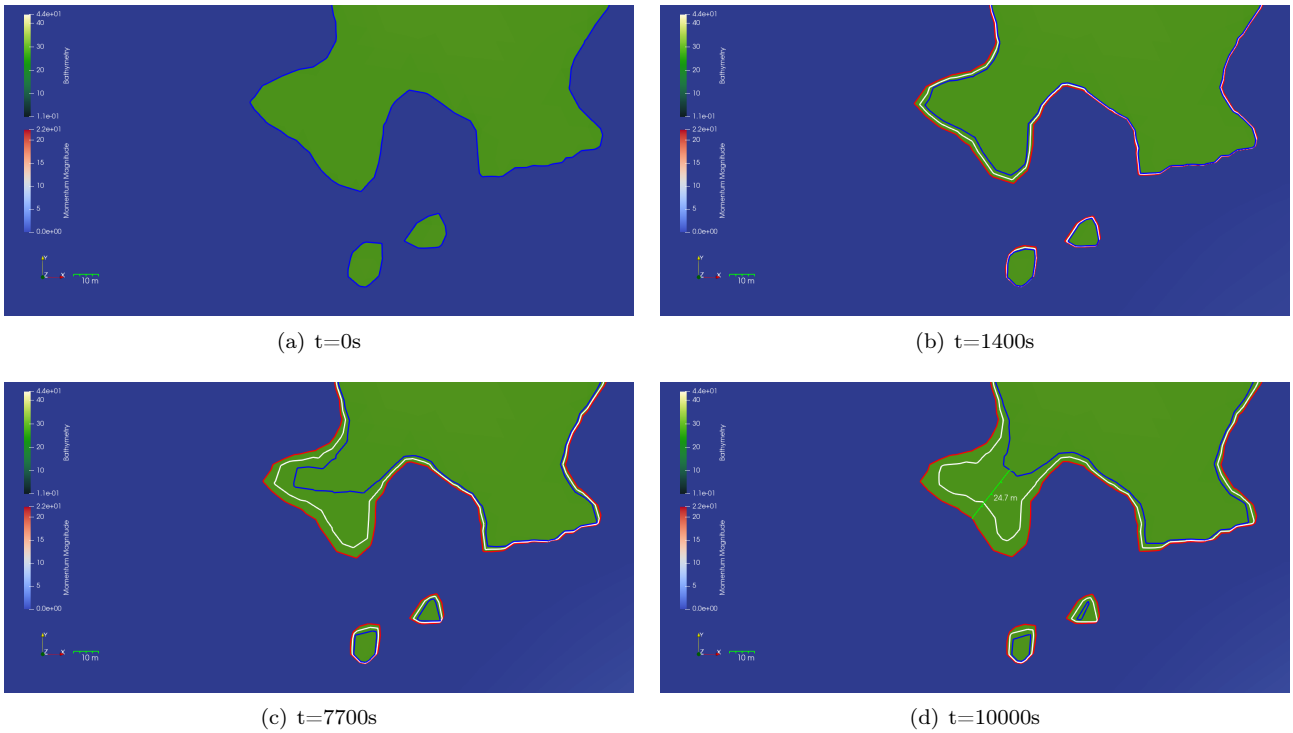


Figure 98: Localisation de la zone pour l'analyse



(a) t=0s

(b) t=1400s

(c) t=7700s

(d) t=10000s

Figure 99: Difference in water height between the three different flow rates at t = 0, 3300, 6600 and 10000 seconds in an area downstream of the Milles Îles River

5.5 Case of the Montreal archipelago

In this section, we study the area of the Montreal archipelago. The data used in this section can be found in the folder CuteFlow/docs/Archipel_de_Montreal. This domain has 7 inlets and the initial conditions consist of giving an initial flow rate to each of these inlets as well as an outlet water height.

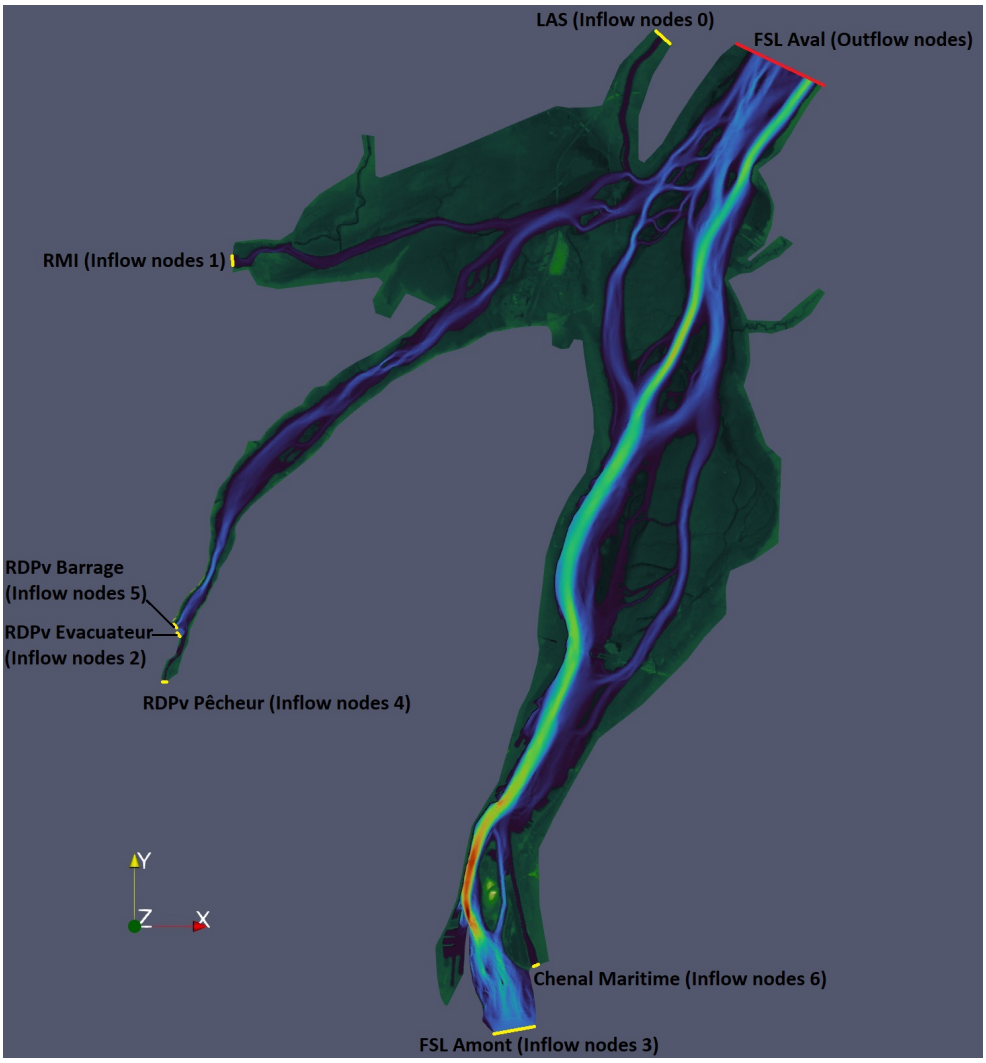


Figure 100: Naming of the entrances to the domain of the Montreal archipelago

Firstly, we will compare the results obtained on a mesh with a constant Manning number to real measurements carried out in the river (see section 5.5.1). Then, the results of these comparisons will be used to generate a mesh with variable Manning numbers in order to generate a solution that best approximates reality (see section 5.5.2).

5.5.1 Manning constant

5.5.1.1 Simulation archi_0160

The first case for which we have real measurements, archi_0160, uses an outlet water height of 5.93 m with the inlet flow rates from Table 101.

	Inflow 0	Inflow 1	Inflow 2	Inflow 3	Inflow 4	Inflow 5	Inflow 6
Flow rate $[m^3/s]$	10	142	858	10300	16.5	96	10

Figure 101: Input flow rates for the archi_0160 case

The solutions of order 1 and 2 projected onto a water line in Rivière-Des-Prairies (see Figure 102) are presented in Figure 103.

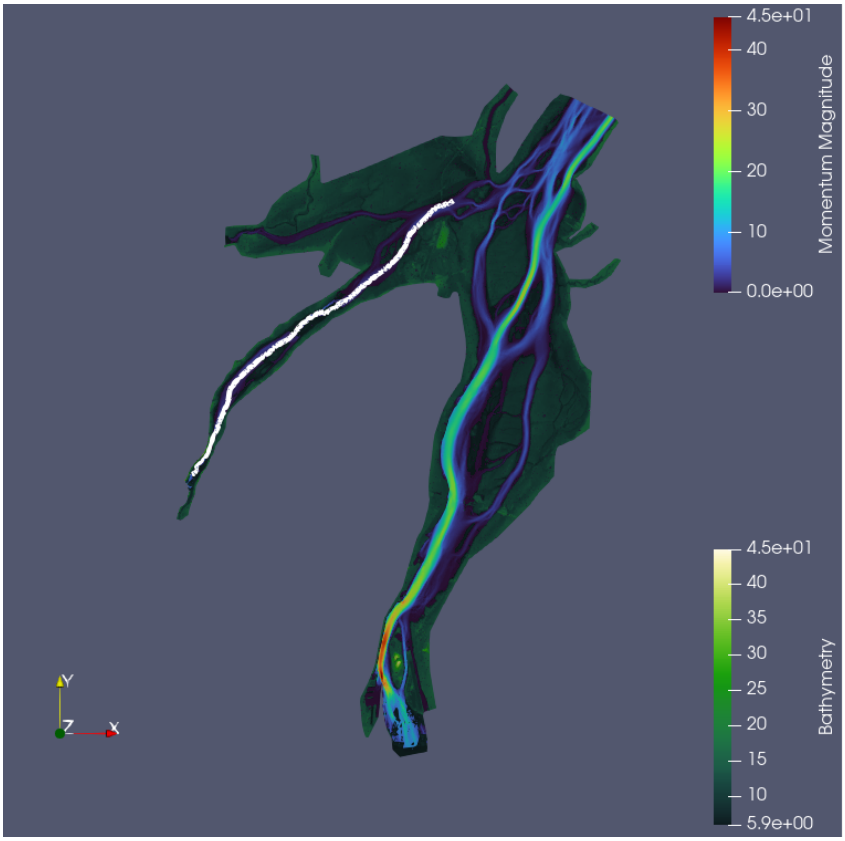


Figure 102: archi_0160 - Line for displaying the solution - Rivière-Des-Prairies

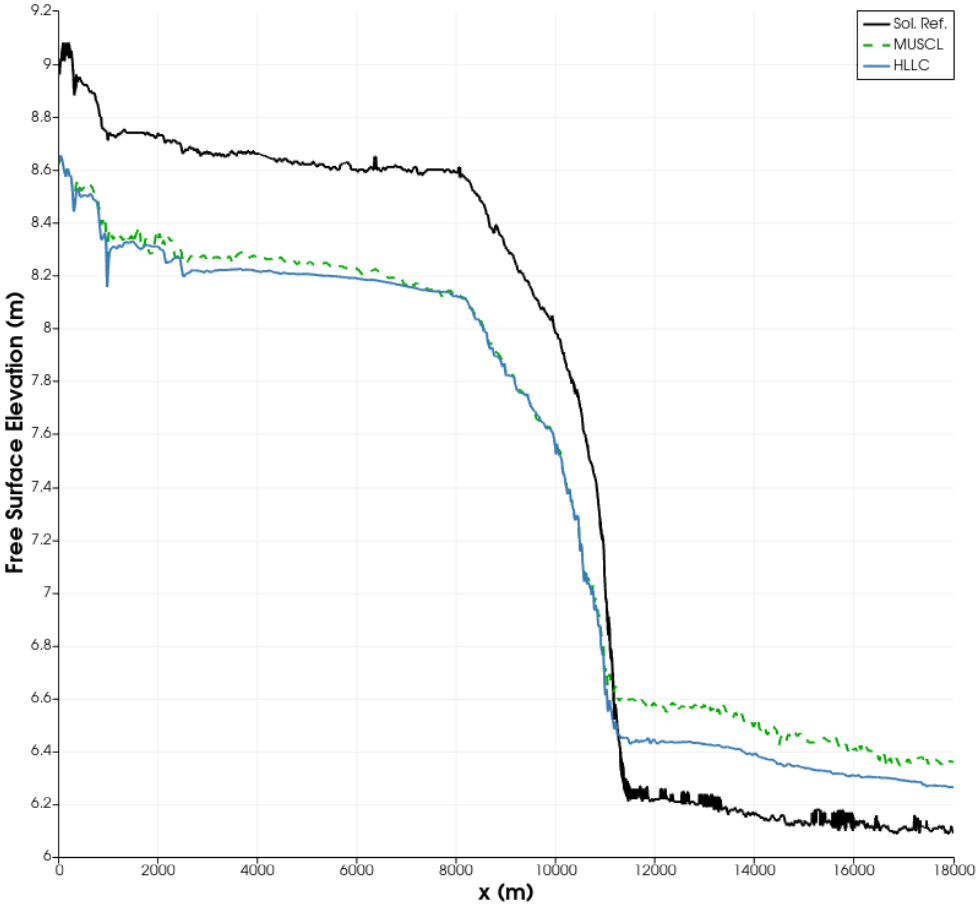


Figure 103: archi_0160 - Difference between orders 1 and 2 on the constant Manning mesh and the readings at t = 80,000 seconds on a line in Rivière-Des-Prairies (102) (Free Surface Elevation)

5.5.1.2 Archival simulation_0164

For the archi_0164 case, an outlet water height of 7.05 m is used with the inlet flow rates presented in the Table104.

	Inflow 0	Inflow 1	Inflow 2	Inflow 3	Inflow 4	Inflow 5	Inflow 6
Flow rate $[m^3/s]$	200	760	2293	10300	16	1	10

Figure 104: Input flow rates for the archi_0164 case

The solutions of order 1 and 2 projected onto a water line in Rivière-Des-Prairies (see Figure 102) are presented in Figure 105.

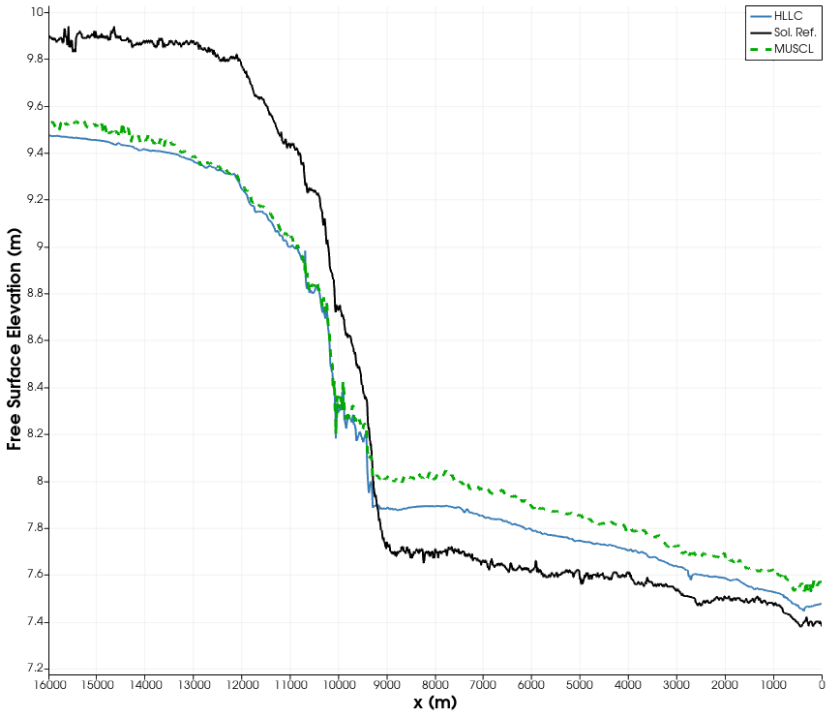
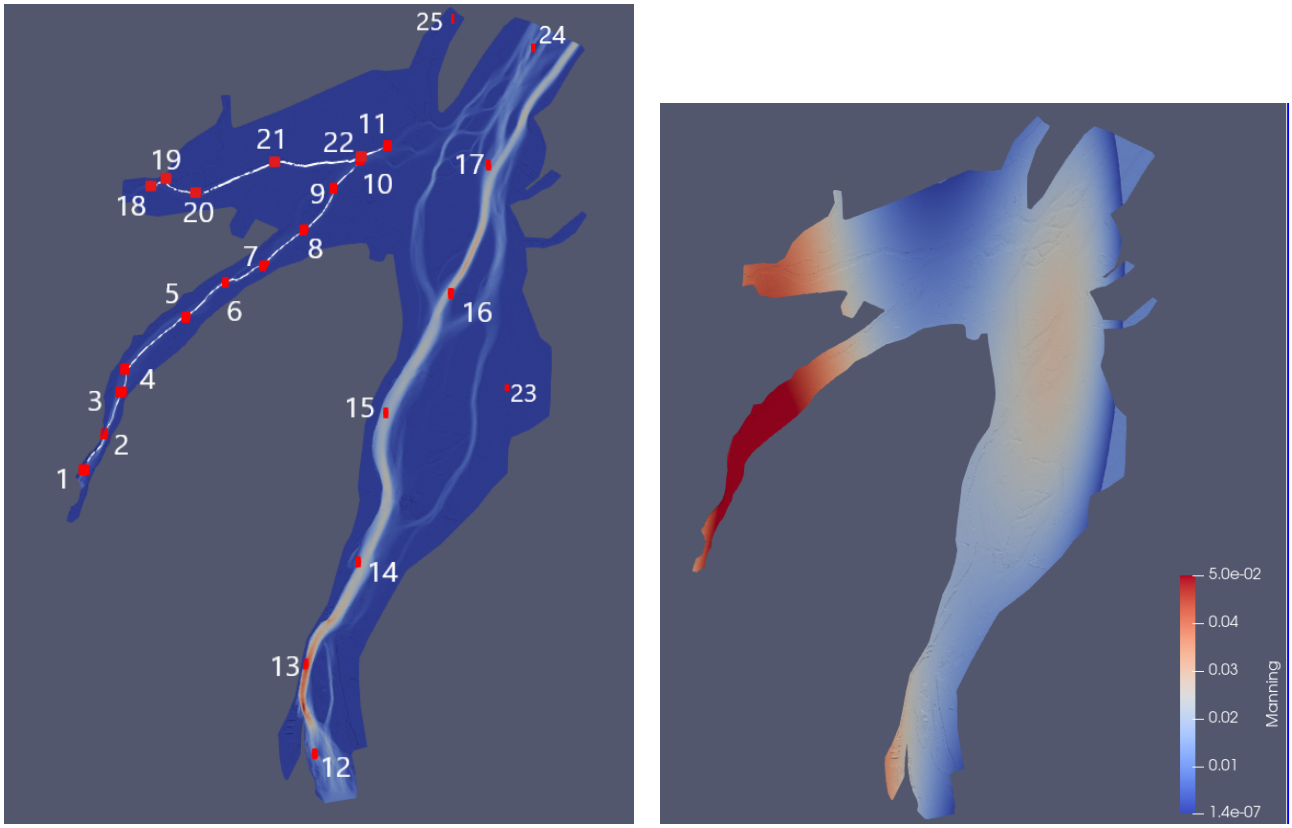


Figure 105: archi_0164 - Difference between orders 1 and 2 on the constant Manning mesh and the readings at t = 40,000 seconds on a line crossing Rivière-Des-Prairies (102) (Free Surface Elevation)

5.5.2 Manning variable

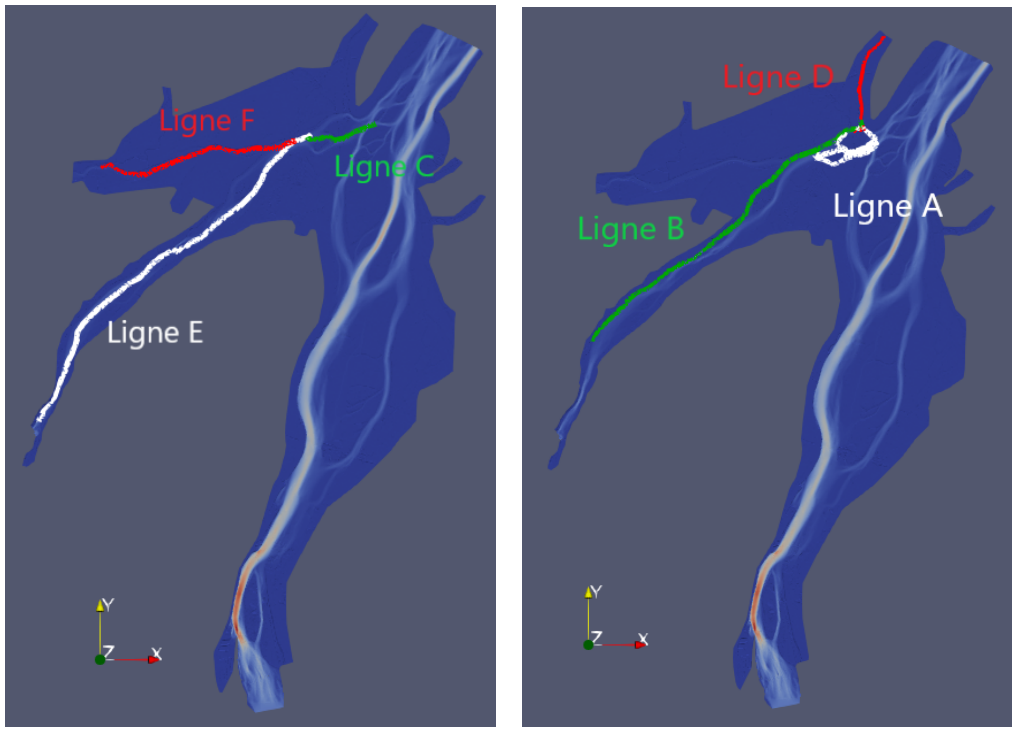
To try to improve the reliability of the results, we wish to modify the Manning number representing the friction at the bottom of the river. In the previous section, both simulations used a constant Manning number throughout the domain. We will now use a different Manning number for each mesh of the domain. In this way, we will be able to improve the results by increasing the Manning number in areas where the water level is too low and by decreasing it in areas where it is too high.

The section 2.4 presents the procedure to follow to change the Manning number of the mesh. Here we use 25 points in the mesh (see Figure 106(a)) to which we will assign a very particular Manning number. After several tests, we arrive at a map of Manning numbers presented in Figure 106(b).



(a) (b)

Figure 106: Points used in the script (a), Manning numbers on the mesh (b)

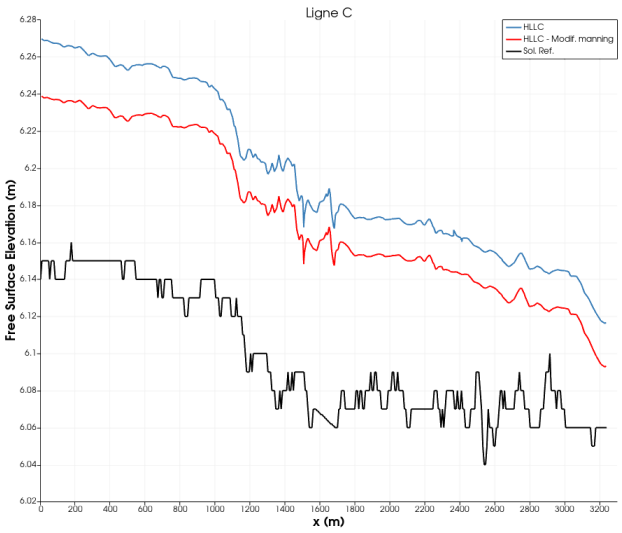


(a) (b)

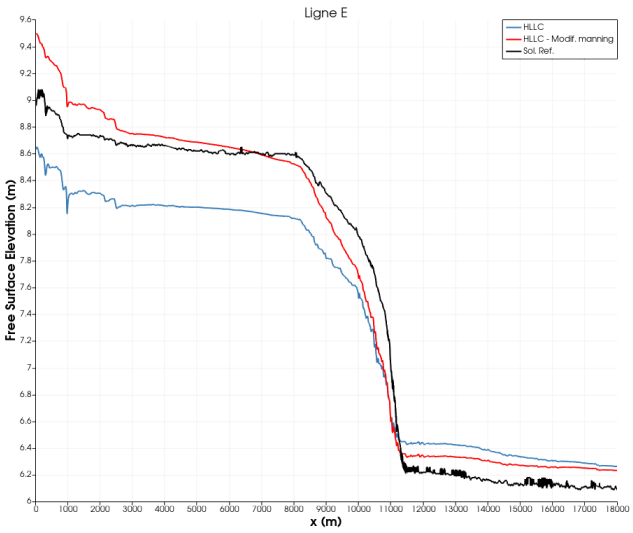
Figure 107: Lines for calculating results

5.5.2.1 Archival case_0160

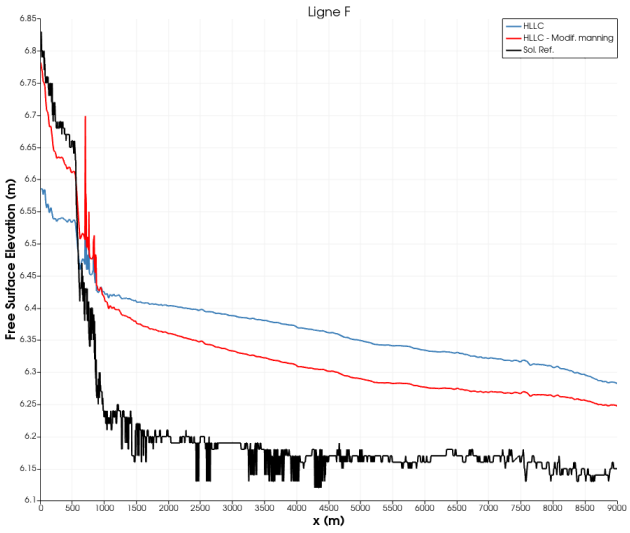
The calculations are carried out on 3 lines in the river presented in Figure 5.5.2. The results obtained are presented in Figure 108.



(a) Ligne C



(b) Ligne E

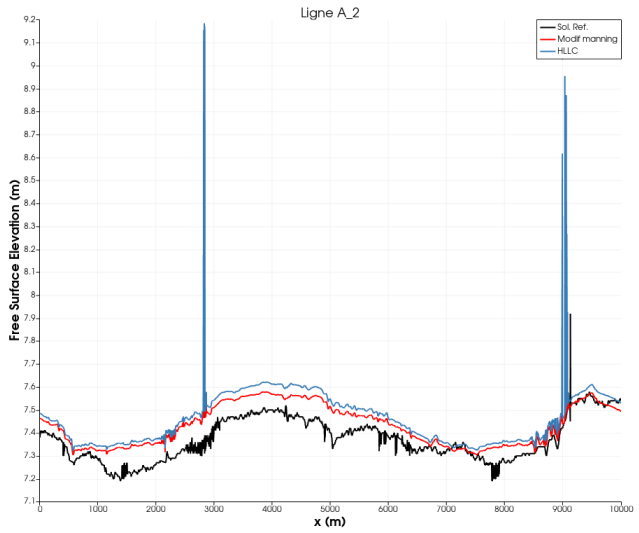


(c) Ligne F

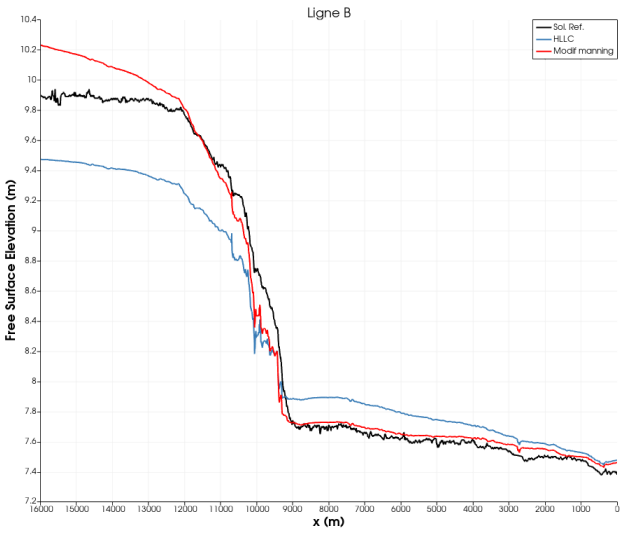
Figure 108: archi_0160 - Difference between the results of the HLLC method (figure 103), the results calculated on the mesh with the modified Manning numbers and the readings at t = 40,000 seconds on 3 lines crossing the archipelago of Montreal (Free Surface Elevation)

5.5.2.2 Archival case_0164

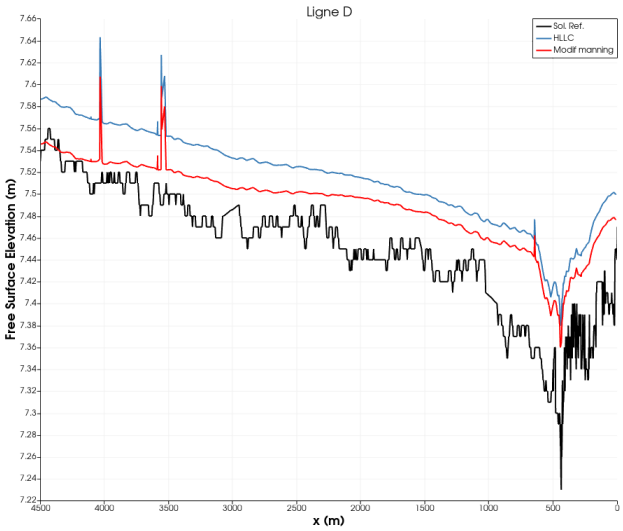
The calculations are carried out on 3 lines in the river presented in Figure 5.5.2. The results obtained are presented in Figure 109.



(a) Ligne A



(b) Ligne B



(c) Ligne D

Figure 109: archi_0164 - Difference between the results of the HLLC method (figure 103), the results calculated on the mesh with the modified Manning numbers and the readings at t = 40,000 seconds on 3 lines crossing the archipelago of Montreal (Free Surface Elevation)

These simulations allow us to see improvements on the lines studied. Meshes can be further refined for results with increased accuracy. The interest in carrying out this work is to generate a mesh whose properties come as close as possible to those of the real river.

6 Calibrating the Manning number with CuteFlow

In free surface flows, the roughness of the flow bed exerts a significant influence on the behavior of the water, varying from one point to another in the flow domain and depending on several parameters. For natural flow studies, the choice of appropriate roughness coefficients is of crucial importance to guarantee the reliability of the results.

In this part, we present an efficient methodology for the identification of the Manning coefficient, based on field data and a sample of numerical results. This method takes place in two stages and combines numerical simulations (performed with CuteFlow) with non-deterministic methods assisted by a substitution model to determine the optimal values of the Manning coefficients.

The approach adopted consists of generating a reference solution for a flow in a domain divided into a number $\boldsymbol{m}$ of

subdomains, each with Manning roughness coefficients $\boldsymbol{n}$ assigned. Subsequently, the calibration process starts from an arbitrary initial solution, generated by the reduced ensemble model, from an arbitrary distribution of Manning parameters. The Manning parameters used to perform the numerical simulations are obtained by sampling using the Sobol algorithm. This calibration process can be illustrated by the figure below:

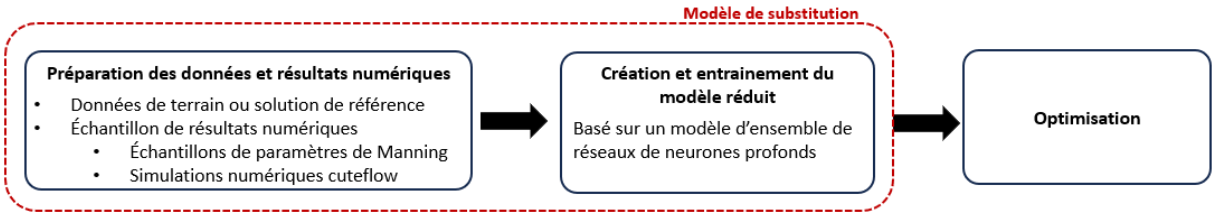


Figure 110: Flowchart of the Manning coefficient calibration process *(based on a substitution model)*

6.1 Data preparation and numerical results

This step essentially consists of generating a reference solution (or field measurements) as well as a sample of numerical results. The reference solution will serve as a basis for calculating the error function in the optimization process while the numerical results will be extracted to train the neural networks.

Here we use the mesh_6072.cgns file as the application domain. We assume that the mesh has already been created and provided and that the reference solution has already been calculated. In our example, the reference solution was calculated with the new mesh generated in section 2.4.1. In order to be more efficient in the following, that is to say, the preparation of the sample of numerical results, the calculations in CuteFlow will be launched simultaneously. We present below the steps to follow to achieve this.

6.1.1 Mesh zoning and Manning number assignment

6.1.1.1 Extraction of cell/mesh identifiers

The first step is to create the Domain Zoning Excel file containing the names of the zones and the cell numbers belonging to them (see section 2.4.1).

Once this step is completed, we can import the Excel file into our workspace on the servers. We will carry out this example in the CuteFlow/Calibration_manning/example2 folder.

```
# From the Cuteflow folder
cd Calibration_manning

# Create file example2
mkdir example2
cd example2

# Download the zoning Excel file from your local computer to the current folder on the server

# From the example2 folder, copy the Zoning python code
cp ../code/Multi_Zonage.py .
```

6.1.1.2 Sampling and zoning

The second step consists of designing an experimental plan for the different values of the Manning coefficients which will be used for the different simulation cases. To do this, there are a multitude of methods or algorithms for generating a list of samples (m-tuples) depending on the number of dimensions or variables to be tested. In our case, we use the Sobol algorithm for sampling Manning values. The procedure for using this code is detailed in the accompanying file **Readme.txt**.

```
# Create the base_files folder in the example2 folder
cd example2
mkdir base_files

# Copy the mesh file to the base_files folder
cp ../../../meshes/rect_mesh_convergence/mesh_6072.cgns .
```

```
# Copy the Cuteflow executable
cp ../../../bin/cuteflow .

# Copy sample code
cp ../../code/Sobol_sequence_Sampling/Sobol_seq.py .
```

At this stage, the result of the ***tree*** command launched in the Calibration_manning/example2 folder should be consistent with the figure below:

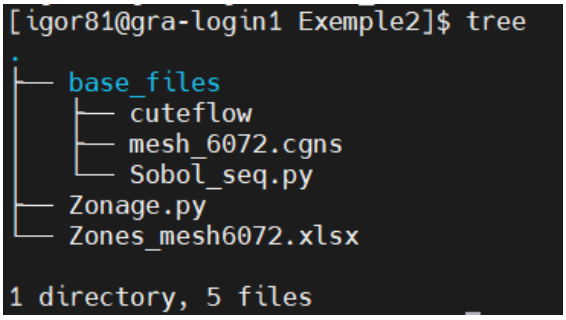


Figure 111: Folder tree example2

We create/activate the virtual environment in which we will launch the Python code (see section 2.4). Once the environment is activated, we use the code like this:

```
# From folder base_files
python3 Sobol_seq.py
```

The Manning.txt file contains a set of **N (m-tuples)** Manning values that will be assigned to each of the zones depending on the case. In the Sobol_seq.py code, we specify the number N **(N=2**m_samples)** of samples, the coefficient of variation **cv**, the corresponding score **z** at the confidence level associated with **cv**, and the central values (reference values) for each dimension. The search intervals for the Manning coefficients are calculated in the code, taking these reference values as target solutions, and considering a uniform distribution with a coefficient of variation **cv**.

In the following example,

```
space = calculRange(0.15, 1.96, [0.022, 0.024, 0.025, 0.017])

# Configuring QMC Sampling: Input Parameters
n_variables = len(space[0])
m_samples = 4 # The number of samples is n_variables = 2**m_samples.
```

we defined **N=16** samples, **cv = 5%**, **z=1.96** and 4 dimensions for the 4 defined zones, the first 3 of which represent the 03 zones selected and imported into the Zones_mesh6072.cgns zoning file.

Once the Manning.txt file is created, we execute the zoning code as follows:

```
cd ..

# Copy the Manning.txt file to the current folder
cp base_files/Manning.txt .

# Creation of folders and Excel files containing cell identifiers and respective Manning values
python3 Multi_Zoning.py Zones_mesh6072.xlsx
```

The Multi_Zonage.py code reads the Manning values file as input and creates for each m-tuple (each line of the Manning.txt file) a Case_*[1- 16] folder. In each of these folders, it generates a new zoning file containing, in addition, the Manning values for each cell.

You can check the correct structure of the Calibration_manning/example2 folder by running the tree command in this folder. The output should look like the figure below:

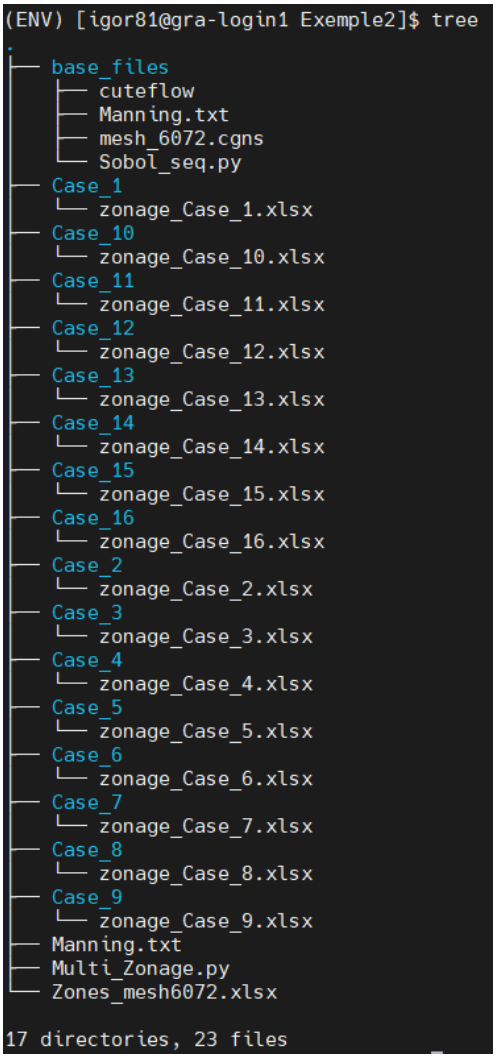


Figure 112: Output of the command ***tree*** after execution of the Multi_Zonage.py script

6.1.2 Adding Manning numbers to mesh files (mesh.cgns)

6.1.2.1 Creating preliminary input files

In order to add a Manning number to each of the meshes in the mesh file, we need the input file containing the cell IDs and the corresponding Manning numbers for each of them. To create these files, we use the script ***Multi_gen_entree_gen_Manni*** which will generate a file "Entree_gen_manning_k.txt", k={1, 2,. . . ,16} for each of the 16 cases considered.

From the **Calibration_manning/example2** folder, we make sure we are in the virtual environment then we execute the following commands:

```
#Copy the Multi_gen_entree_gen_Manning.py script into the working directory
cp ../code/Multi_gen_entree_gen_manning.py .

# Execute code
python3 Multi_gen_entree_gen_manning.py
```

Once this command is executed, a simple check with the tree or ls command confirms that each of the 16 folders contains a text file "Entree_gen_manning_k", k={1,2,. . . ,16}

6.1.2.2 Launching the Multi.sh script

The Multi.sh script reads two files as input and creates in each simulation folder a data file (**data.f**) and a python code for assigning manning values to the entire mesh (textbfgen_manning.py).

In the first file (**Manning.txt**) passed as an argument, it reads for m-tuple (corresponding to a simulation case), the last value of the Manning coefficient and stores it in variable average_manning. It corresponds to the Manning coefficient of the area not selected at the zoning stage. In the second file (**initialization.dat**) passed as an argument, the Multi.sh script reads the flow rate and water height values (upstream and downstream) and then generates the data file in each folder with these parameters.

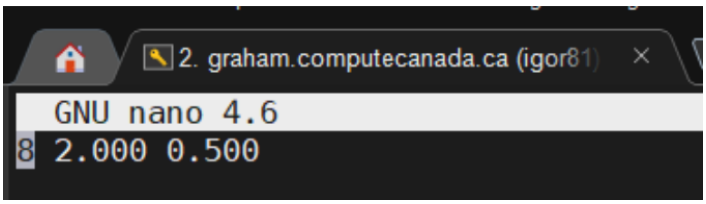


Figure 113: Contents of the initialization.dat file

In the "initialisation.dat" file, the first value corresponds to the flow rate, the second to **h** upstream and the last to **h** downstream.
The Multi.sh script also calls two other bash scripts (gen_donnees.sh and gen_manning.sh) to generate the desired output files. It is important to copy them into the current directory, like this:

```
# From the Calibration_manning/example2 folder
# Copy the gen_donnees.sh and gen_manning.sh files
cp ../scripts/gen_donnees.sh gen_manning.sh .

# Copy script Multi.sh
cp ../scripts/Multi.sh .
```

Once this step has been completed, you can launch the script with the command:

```
# Launching the Multi.sh script
./Multi.sh Manning.txt initialisation.dat
```

At this stage, the contents of each folder should be similar to the figure below:

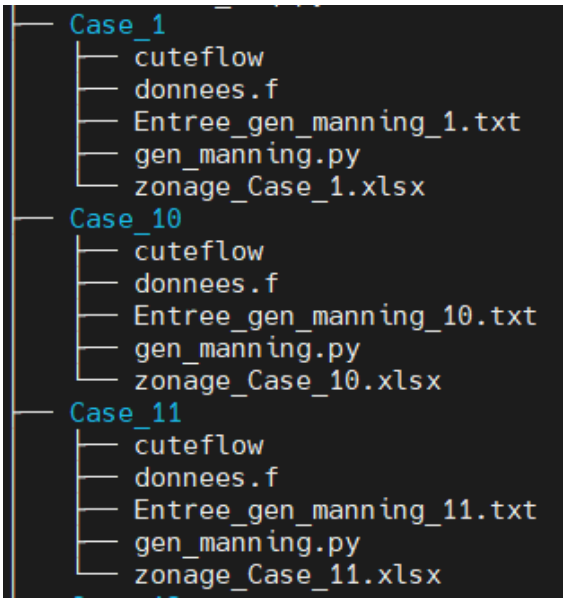


Figure 114: Contents of each folder after running the Multi.sh script

6.1.2.3 Creation of the mesh files manning_mesh*.cgns and manning_mesh.vtk

The Multi_manning.sh script allows you to move to each folder and launch the gen_manning.py code in order to assign Manning values to the mesh cells.

This script launches as follows:

```
# Copy the Multi_manning.sh script into the current folder
cp ../scripts/Multi_manning.sh .

# Make sure you are in the virtual environment and launch the script
./Multi_manning.sh fichier_maillage.cgns
```

Before launching the Multi_manning.sh script, you must ensure that the variables are well defined (name of the mesh file, number of cases) as below:

```
#!/usr/bin/bash

for i in {1..16}
do
   cd Case_$i
   cp ../base_files/mesh_6072.cgns .
   python3 gen_manning.py mesh_6072.cgns

   mv manning_mesh_6072.cgns manning_mesh_6072_Case_$i.cgns

   rm mesh_6072.cgns
   cd ..
done
let i=$i+1
```

Figure 115: Structure of the Multi_manning.sh script (number of cases = 16)

6.1.3 Starting calculations: job array

All of the previous steps were used to create the folder structure needed to run all 16 simulations simultaneously. You must now launch the calculations on the Compute Canada cluster. To do this, we also use a job array as in section 3.4.4. The file we will use for this step is the **calib_array.sh** file contained in the Calibration_manning/scripts folder.

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --gres=gpu:p100:2
#SBATCH --mem=4000M
#SBATCH --time=0-00:10
#SBATCH --account=def-soulaima
#SBATCH --array=1-16

cd Case_${SLURM_ARRAY_TASK_ID}
mpirun --mca pml ob1 -n 1 sh -c './cuteflow > outfile.$OMPI_COMM_WORLD_RANK'
```

Figure 116: calib_array.sh file (number of simulations =16)

To submit the job to the scheduler:

```
# Copy the script to the current folder
cp ../scripts/calib_array.sh .

# Launch job_array
sbatch calib_array.sh
```

The 10 simulations will be launched on the calculation cluster as can be seen in the figure below:

```
(ENV) [igor81@gra-login3 Exemple2]$ sq
             JOBID     USER             ACCOUNT           NAME  ST  TIME_LEFT NODES CPUS TRES_PER_N MIN_MEM NODELIST (REASON)
       16615321_1   igor81     def-soulaima_gpu calib_array.sh   R       9:57     2    2 gres:gpu:p   4000M gra[978-979] (None)
       16615321_2   igor81     def-soulaima_gpu calib_array.sh   R       9:57     2    2 gres:gpu:p   4000M gra[980-981] (None)
  16615321_[3-16]   igor81     def-soulaima_gpu calib_array.sh  PD      10:00     2    2 gres:gpu:p   4000M  (Priority)
```

Figure 117: Launch of job_array

6.2 Post-processing solution files *out_ *.cgns* and data extraction

At this stage, the user can either do remote viewing with the Paraview software installed on the computing clusters (see section 4.1) or download the files locally and access them from the software installed on his computer.

6.2.1 Viewing the solution files generated by CuteFlow

The process of viewing solution files generated by CuteFlow is very simple. It boils down to the following main steps (see section 4.2):
Opening the solution file **out_*.cgns** > choosing the data reader > applying the MergeBblocks tool *(if necessary)* > selecting and applying the Cell Data to point Data tool to avoid aliasing *(optional)*.

6.2.2 Data extraction

Once the solution file is opened in Paraview, it is possible to extract and export the temporal solutions on the cells of interest of the mesh. In this part we present a simple example of extracting solutions from the out_manning_mesh_6072_Case_5.cgn file. The solutions are extracted along a horizontal line dividing the domain in two.

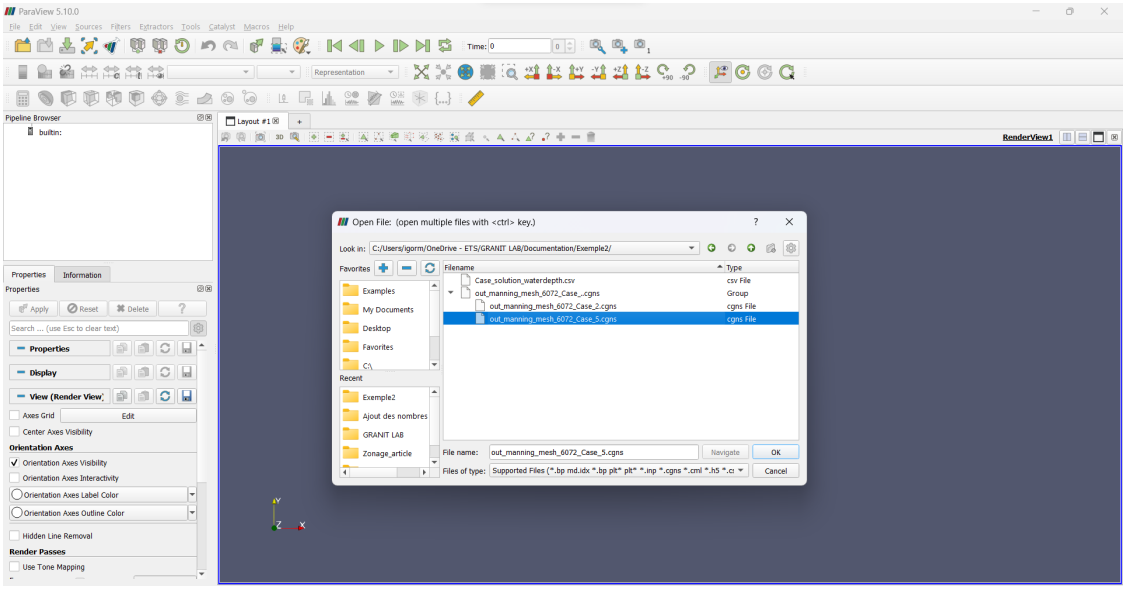


Figure 118: Opening the out_manning_mesh_6072_case_5.cgns file and choosing the reader

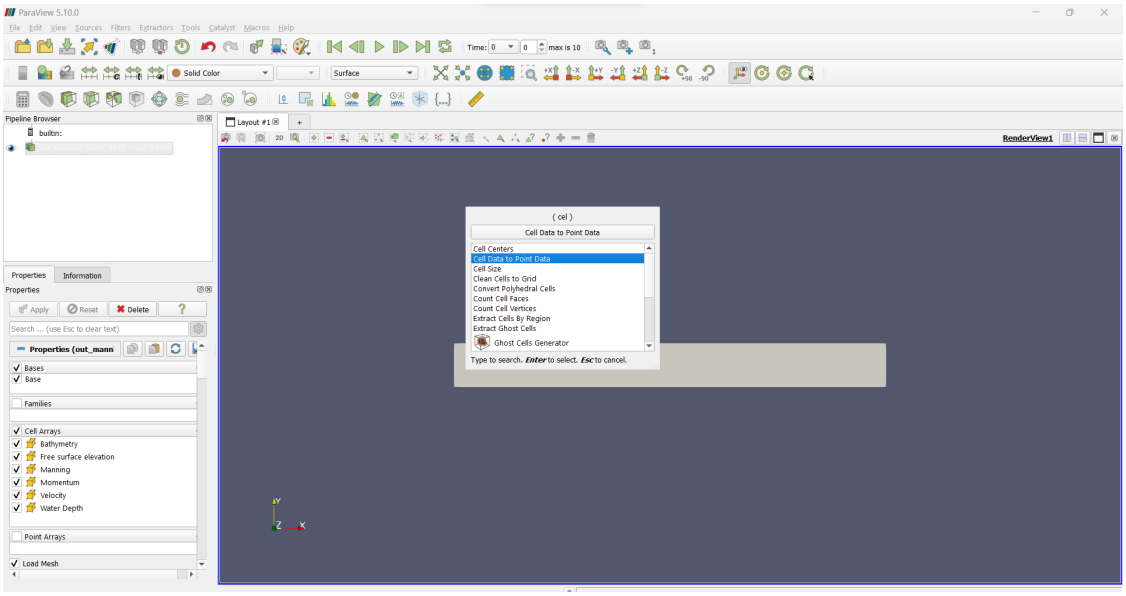


Figure 119: Selecting the tool *Cell Data to Point Data*

The ***Waterdepth_case5*** file will be created in the specified directory and can be opened with a text editor such as Notepad++ or a spreadsheet program such as Excel, for later use.

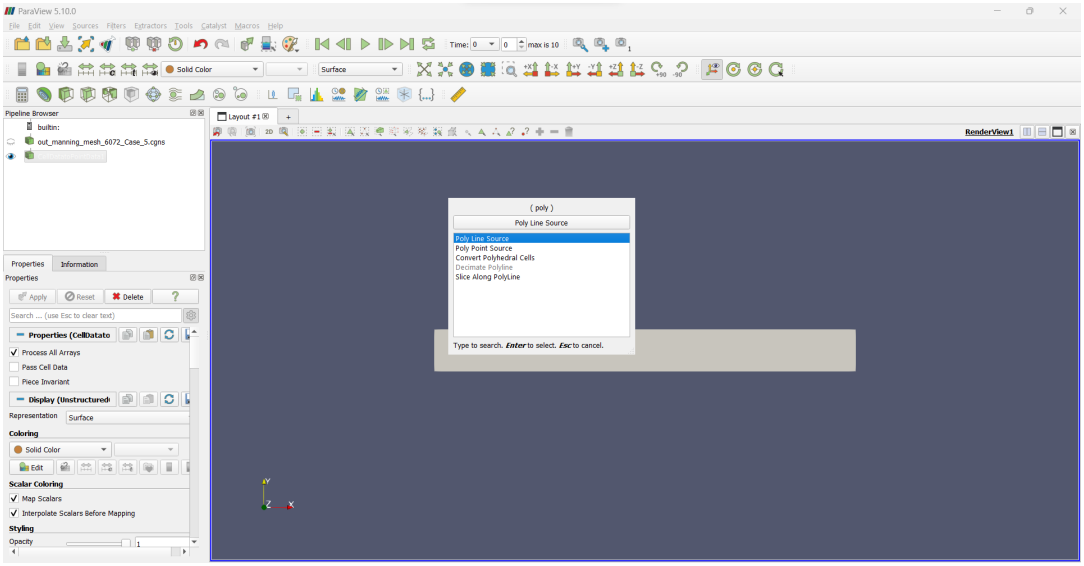


Figure 120: Selecting the *Poly Line Source* tool to draw the line for viewing the results.

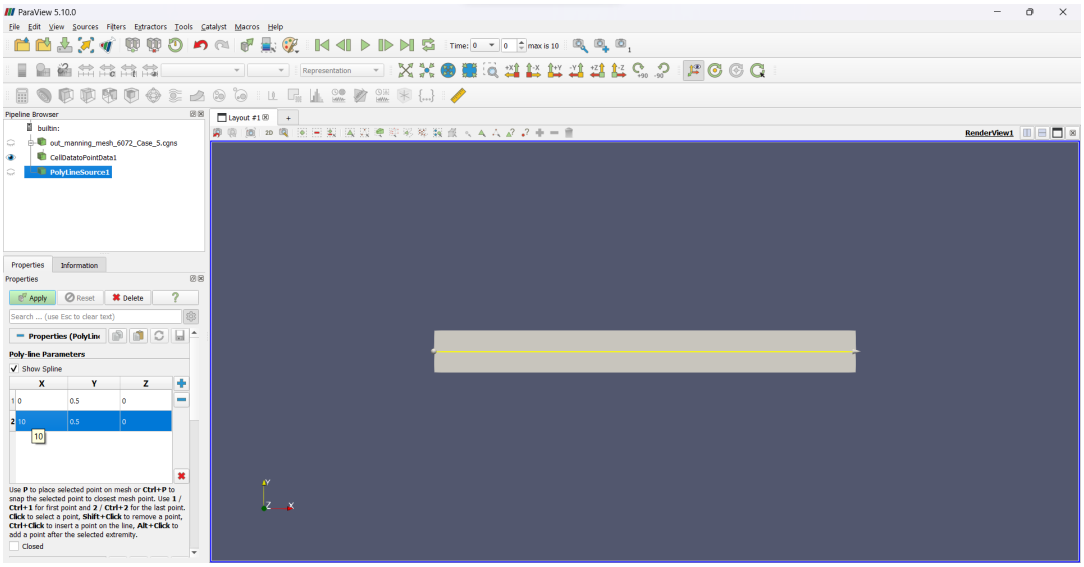


Figure 121: Choice of start and finish coordinates of the line, then *Apply*

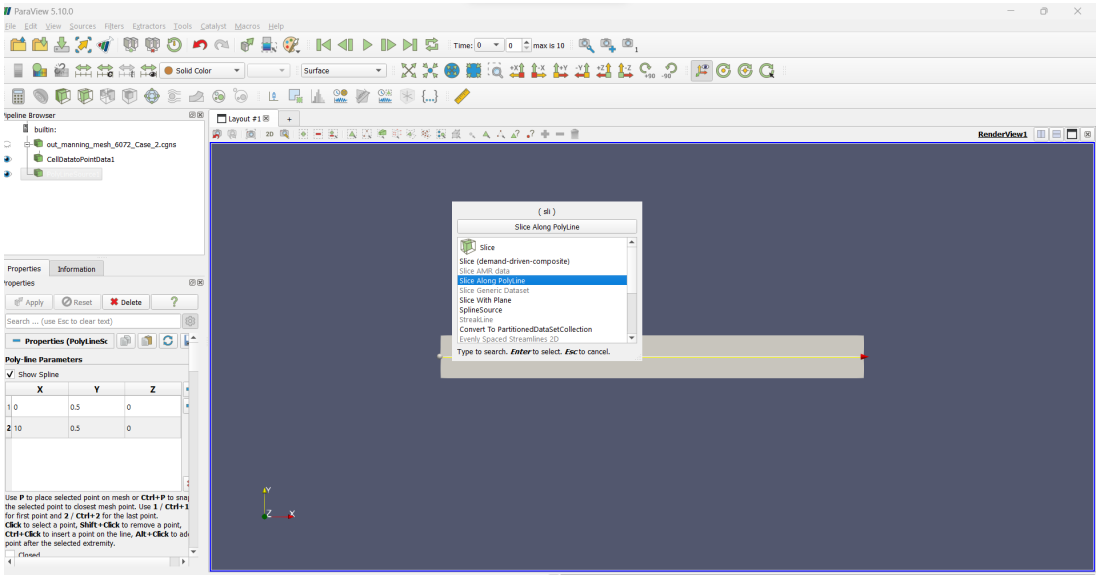
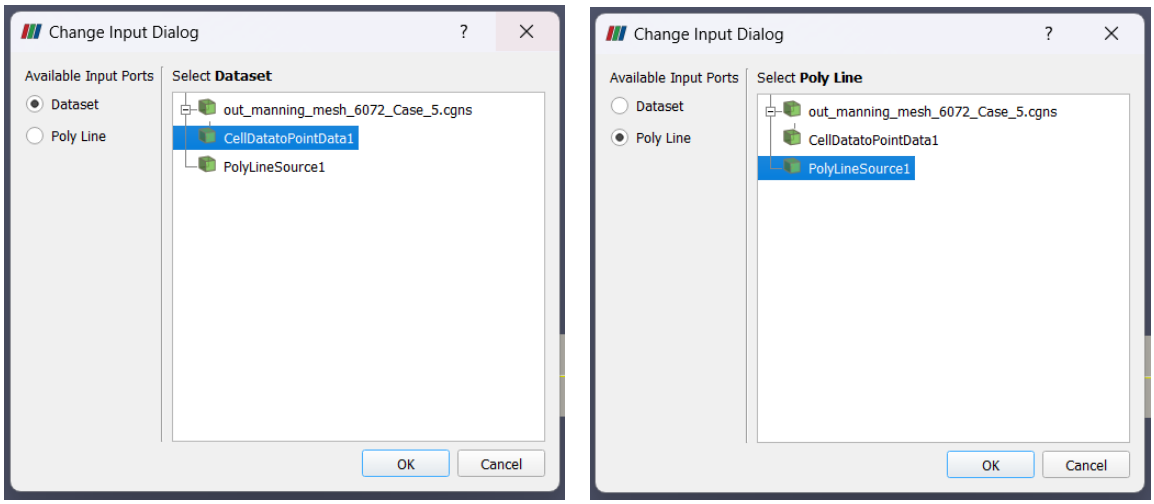


Figure 122: Selection of the *Slice along polyline* tool to project the solution onto the line.



(a) Check that *Dataset* is indeed *CellDataToPointData1*. (b) Check that *Poly Line* is indeed *PolyLineSource1*.

Figure 123: Checking the projection parameters of the solution on the drawn line

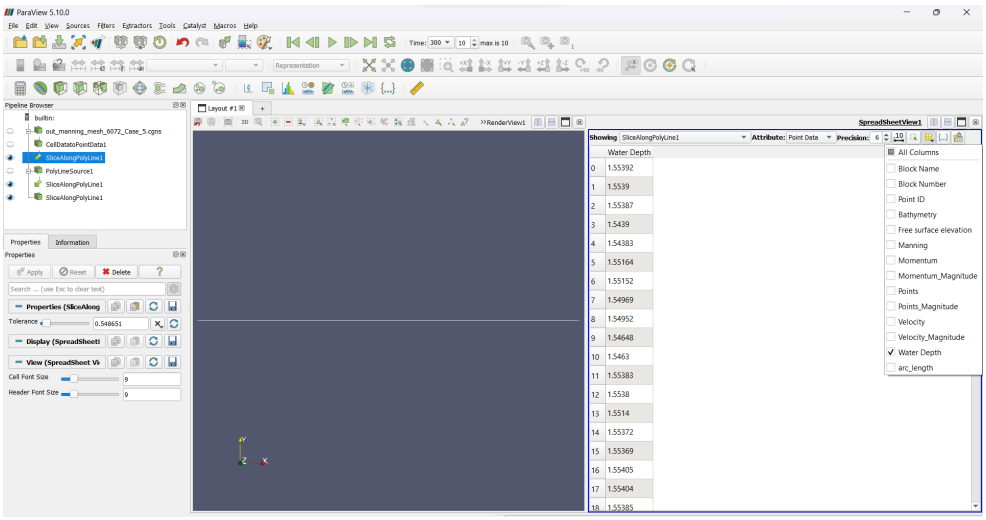


Figure 124: Selection of attributes to export

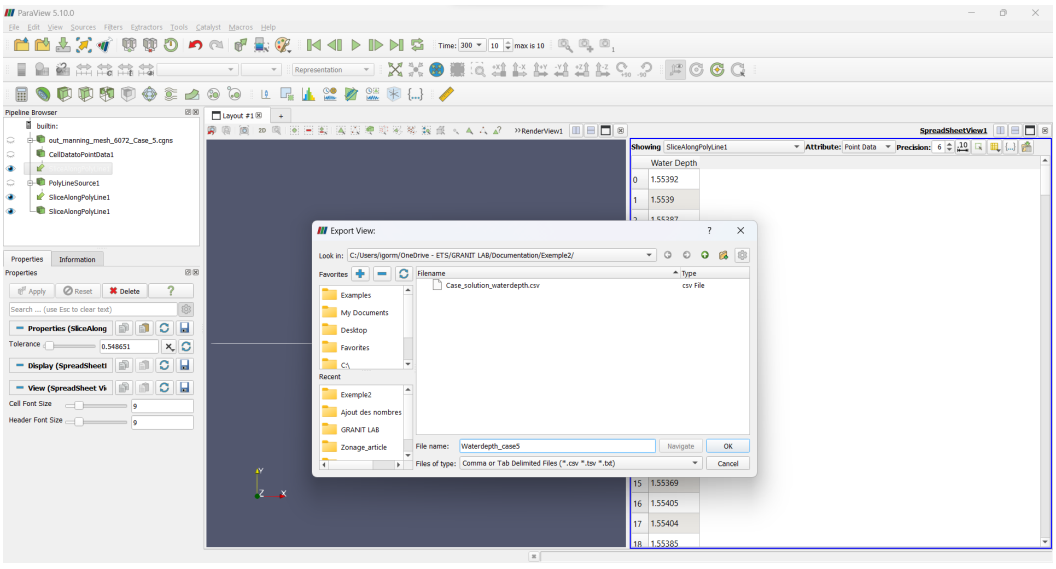


Figure 125: Saving solutions in the file *Waterdepth_ case5*

Repeat the same steps for the other 15 solution files and collect the results in a single file as below (***Outputs_ 223.xlsx*** in our case for example). This file will subsequently be used to train the neural networks constituting the overall model.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	**Case1**	**Case2**	**Case3**	**Case4**	**Case5**	**Case6**	**Case7**	**Case8**	**Case9**	**Case10**	**Case11**	**Case12**	**Case13**	**Case14**	**Case15**	**Case16**
2	1.54398	1.55408	1.55289	1.55431	1.55392	...	...	...	...	...	...	...	...	...	...	...
3	1.54407	1.5541	1.55259	1.55431	1.5539	...	...	...	...	...	...	...	...	...	...	...
4	1.54935	1.55425	1.55259	1.55431	1.55387	...	...	...	...	...	...	...	...	...	...	...
5	1.54613	1.55425	1.55282	1.55431	1.5439	...	...	...	...	...	...	...	...	...	...	...
6	1.54547	1.55413	1.55275	1.55426	1.54383	...	...	...	...	...	...	...	...	...	...	...
7	1.54531	1.55414	1.55275	1.55426	1.55164	...	...	...	...	...	...	...	...	...	...	...
8	1.54516	1.55431	1.55251	1.55426	1.55152	...	...	...	...	...	...	...	...	...	...	...
9	1.55334	1.55431	1.55243	1.5543	1.54969	...	...	...	...	...	...	...	...	...	...	...
10	1.55334	1.55431	1.55394	1.5543	1.54952	...	...	...	...	...	...	...	...	...	...	...
11	1.55061	1.55431	1.55412	1.54437	1.54648	...	...	...	...	...	...	...	...	...	...	...
12	1.55061	1.55431	1.55411	1.54426	1.5463	...	...	...	...	...	...	...	...	...	...	...
13	1.55301	1.55426	1.55392	1.54501	1.55383	...	...	...	...	...	...	...	...	...	...	...
14	1.55295	1.55426	1.5539	1.54487	1.5538	...	...	...	...	...	...	...	...	...	...	...
15	1.55289	1.55426	1.55387	1.54866	1.5514	...	...	...	...	...	...	...	...	...	...	...
16	1.55259	1.5543	1.5439	1.54866	1.55372	...	...	...	...	...	...	...	...	...	...	...
17	1.55259	1.5543	1.54383	1.55032	1.55369	...	...	...	...	...	...	...	...	...	...	...
18	1.55282	1.54437	1.55164	1.55016	1.55405	...	...	...	...	...	...	...	...	...	...	...
19	1.55275	1.54426	1.55431	1.54579	1.55404	...	...	...	...	...	...	...	...	...	...	...
20	1.55275	1.54501	1.55431	1.54562	1.55385	...	...	...	...	...	...	...	...	...	...	...
21	1.55251	1.54487	1.55431	1.55339	1.55375	...	...	...	...	...	...	...	...	...	...	...
22	1.55243	1.54866	1.55431	1.55032	1.55429	...	...	...	...	...	...	...	...	...	...	...
23	1.55394	1.55429	1.55426	1.55016	1.55429	...	...	...	...	...	...	...	...	...	...	...
24	1.55412	1.55429	1.55426	1.54579	1.54376	...	...	...	...	...	...	...	...	...	...	...
25	1.55411	1.55429	1.55426	1.54562	1.54371	...	...	...	...	...	...	...	...	...	...	...

Figure 126: Structure of the final file containing all the solutions for all simulation cases

6.3 Creation and training of the ensemble model

The overall model can be built from measurements of different hydraulic variables (water height, momentum, flow speed) and even by combining these. It is built from a set of trained and averaged neural networks. Currently, this step is carried out locally, on the user's personal computer. It can be carried out with Matlab software (using the integrated libraries) or in Python, at the user's discretion.

In our example, the overall model is built from water height measurements. To do this, a set of 10 forward propagation neural networks, each composed of four layers was trained by initializing the weights randomly. The Matlab code used to build the neural networks is provided below:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name of file : Train_dataset.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Clear memory
clc; clear all; close all;

% Read data
Inputs = xlsread('Inputs_16.xlsx',A2:D17); % Fichier des echantillons de manning utilises
Outputs = xlsread('Outputs_223.xlsx','A2:P224'); % Fichier des resultats numeriques
x1 = Inputs(:,:)'; % Echantillons de Manning
y1 = Outputs(:,:); % Hauteur d'eau

% Normalize Manning coefficients
Max_x1 = max(x1,[],all);
Min_x1 = min(1x.[],all);
save('max_manning_data.mat','Max_x1');
save('min_manning_data.mat','Min_x1');

% For calculating the objective function
Max_y1 = max(y1,[],'all');
Min_y1 = min(y1,[],'all');
save('max_y_data.mat','Max_y1');
save('min_y_data.mat','Min_y1');

% Normalization of the Manning value matrix
for i = size(x1,2)
```

```
        x(:,i) = (x1(:,i)-Min_x1)/(Max_x1-Min_x1);
    end
    % Creating the overall model
    M = 10 ; % The number of neural networks in the model
    for m = 1:M
        % Create the neural network
        net1 = feedforwardnet([212 108 212 108], 'trainscg');
        net1.trainParam.epochs=1000;
        net1.trainParam.min_grad= 1e-9; %% Minimum performance gradient. The default value is 1e-6.
        net1.trainParam.max_fail = 6;
        net1.trainParam.mu = 0.005;
        net1.trainParam.sigma = 5.0e-05;
        net1.trainParam.lambda = 5.0e-7;
        net1 = train(net1, x, y1);

        % Save the neural network
        reseau_hauteur_xnorm = net1;
        save (sprintf('reseau_hauteur_xnorm_%d.mat', m), 'net1')
        pause
    end
```

The Manning samples file (Inputs_16.xlsx) read by this code looks like this:

	A	B	C	D	E
1	n1	n2	n3	n4	
2	0.02274	0.02408	0.02735	0.0157	
3	0.02005	0.02367	0.02411	0.01734	
4	0.02048	0.02547	0.02702	0.01831	
5	0.02386	0.02387	0.02285	0.01806	
6	0.02316	0.02626	0.02474	0.01762	
7	0.02029	0.02319	0.02539	0.01588	
8	0.02182	0.02487	0.02375	0.0177	
9	0.02338	0.02196	0.02669	0.01701	
10	0.02202	0.0253	0.02311	0.0169	
11	0.02152	0.02247	0.02607	0.01799	
12	0.02146	0.02602	0.0244	0.01542	
13	0.0224	0.02272	0.02513	0.01865	
14	0.02403	0.02506	0.02571	0.01635	
15	0.02067	0.02179	0.02339	0.01647	
16	0.02107	0.02434	0.02642	0.01662	
17	0.02303	0.02292	0.02384	0.016	
18					

Figure 127: Structure of the file ***Inputs_16.xlsx*** of Manning values

The individual outputs of each neural network will then be averaged to obtain the output solution of the ensemble model.

6.4 Parameter optimization

Manning parameter optimization requires carefully defining the objective function that will be minimized in order to determine the optimal solutions. It consists of comparing the output solution of the ensemble model (predicted result) to the reference solution (field data) by optimizing the objective function or loss function. The loss function used in this approach is the mean square error (in English MSE) defined in the code below:

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Name of file : Objective_function.m
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    function [f,ypredict_ens,y_measure,Sigma_ens] = Objective_function(x_new)
```

```
% Note
% x_new must be normalized before being called by PSO or AG
% M corresponds to the number of neural networks in the ensemble model
% Npts corresponds to the number of measurement points

M=10;
Npts = 223;
Ym = zeros(Npts,M);

% Reading the observed values (Case_solution)
y_measure = xlsread('Case_solution.xlsx', 'A2:A224'); % Hauteur d'eau

% Loop on neural networks
nets = cell(1, 10);
for m = 1:M
    % Load neural networks
    filename_ANN = sprintf('reseau_hauteur_xnorm_%d.mat', m);
    loaded_net = load(filename_ANN);
    net_fieldnames = fieldnames(loaded_net);
    ANN{m}=loaded_net.(net_fieldnames{1});

    % Output prediction using ANN
    ypredict = ANN{m}(x_new');

    % Registration in the Ym matrix of the ensemble model
    Ym(:,m) = ypredict ;
end

% Calculating the average value of Ypredict
ypredict_ens = mean(Ym,2);

% Calculation of the variance of the ensemble model
Sigma_ens = 0;
for m = 1:M
    % Calculation of the variance of the current neural network
    Sigma_m = 0;
    y_m = Ym(:,m);
    for k=1:Npts
        Sigma_m = Sigma_m + (y_m(k,:)-ypredict_ens(k,:))^2;
    end
    Sigma_m = Sigma_m/Npts;

    % Addition to Sigma_ens
    Sigma_ens = Sigma_ens + Sigma_m ;
end

% Model variance (optional)
Sigma_ens =Sigma_ens/M ;

% OBJ: Calculation of the squared error (MSE)
f= (norm(ypredict_ens(:,1) -y_measure(:,1)))^2;
f = f/Npts;
end
```

Once the ensemble model has been constructed and the objective function defined, the calibration can be carried out by the genetic algorithm or by particle swarm optimization (PSO). The Matlab codes used are provided below:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name of file : GA.m
% Main genetic algorithm optimization program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
clc;
close all;
clear all;
tic;

load ('max_manning_data.mat','Max_x1');
load ('min_manning_data.mat','Min_x1');

% Upper and lower bounds of search intervals for Manning values
lb = [0.0198 0.0235 0.0216 0.0153];
ub = [0.0242 0.0285 0.0264 0.0187];

% Normalization of bounds
lb_norm = (lb-Min_x1)/(Max_x1-Min_x1);
ub_norm = (ub-Min_x1)/(Max_x1-Min_x1);

fun = @Objective_function;
opts = optimoptions(@ga, ...
       'PopulationSize', 25, ...
        'MaxGenerations',200, ...
        'MutationFcn',{@mutationadaptfeasible,1,0.1},...
         'CrossoverFcn', {@crossoverheuristic, 0.8},...
           'SelectionFcn',@selectionroulette, ...
            'FunctionTolerance', 1e-6, ...
             'Display','iter', ...
              'PlotFcn', @gaplotbestf );
 numberOfVariables = 5;
 [x,fval,exitflag,output] = ga(fun,numberOfVariables,[],[],[],[],lb_norm,ub_norm,[],opts)

 % Values of mannings calibrated by optimization
x_abs = x*(Max_x1-Min_x1)+Min_x1;

% Exact/desired Manning values
Manning_exacte = [0.022, 0.026, 0.024, 0.017];

% Table of results
T_Column = {'Manning desires','Manning calibres','Erreur (%)'};
n1 = [Manning_exacte(:,1);x_abs(:,1);(100/Manning_exacte(:,1))*(abs(Manning_exacte(:,1)-x_abs
    ↪ (:,1)))];
n2 = [Manning_exacte(:,2);x_abs(:,2);(100/Manning_exacte(:,2))*(abs(Manning_exacte(:,2)-x_abs
    ↪ (:,2)))];
n3 = [Manning_exacte(:,3);x_abs(:,3);(100/Manning_exacte(:,3))*(abs(Manning_exacte(:,3)-x_abs
    ↪ (:,3)))];
n4 = [Manning_exacte(:,4);x_abs(:,4);(100/Manning_exacte(:,4))*(abs(Manning_exacte(:,4)-x_abs
    ↪ (:,4)))];
n5 = [Manning_exacte(:,5);x_abs(:,5);(100/Manning_exacte(:,5))*(abs(Manning_exacte(:,5)-x_abs
    ↪ (:,5)))];

T = table(n1,n2,n3,n4,n5,'RowNames',T_Column);

format shortG;
disp(T);

% Quality of results and display (Example)
[f,ypredict_ens,y_measure,Sigma_ens] = fun(x);
disp('La variance du modele est:');
disp(Sigma_ens);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name of file : PSO.m
% Main particle swarm optimization program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Clear Memory
clc;
close all;
clear all

tic;
fun = @Objective_function;
load ('max_manning_data.mat','Max_x1');
load ('min_manning_data.mat','Min_x1');

% Upper and lower bounds
lb = [0.0198 0.0235 0.0216 0.0153];
ub = [0.0242 0.0285 0.0264 0.0187];

% Normalize bounds
lb_norm = (lb-Min_x1)/(Max_x1-Min_x1);
ub_norm = (ub-Min_x1)/(Max_x1-Min_x1);
nvars = 5;

% PSO settings
options = optimoptions('particleswarm','MaxIterations',1000,'SwarmSize',nvars*10,'Display','iter
    ↪ ','PlotFcn',@pswplotbestf);

% Optimization
[x,fval,exitflag,output] = particleswarm(fun,nvars,lb_norm,ub_norm,options);

% Values of mannings calibrated by optimization
x_abs = x*(Max_x1-Min_x1)+Min_x1;

% Exact/desired Manning values
Manning_exacte = [0.022, 0.026, 0.024, 0.017];


% Table of results
T_Column = {'Manning desires','Manning calibres','Erreur (%)'};
n1 = [Manning_exacte(:,1);x_abs(:,1);(100/Manning_exacte(:,1))*(abs(Manning_exacte(:,1)-x_abs
    ↪ (:,1)))];
n2 = [Manning_exacte(:,2);x_abs(:,2);(100/Manning_exacte(:,2))*(abs(Manning_exacte(:,2)-x_abs
    ↪ (:,2)))];
n3 = [Manning_exacte(:,3);x_abs(:,3);(100/Manning_exacte(:,3))*(abs(Manning_exacte(:,3)-x_abs
    ↪ (:,3)))];
n4 = [Manning_exacte(:,4);x_abs(:,4);(100/Manning_exacte(:,4))*(abs(Manning_exacte(:,4)-x_abs
    ↪ (:,4)))];
n5 = [Manning_exacte(:,5);x_abs(:,5);(100/Manning_exacte(:,5))*(abs(Manning_exacte(:,5)-x_abs
    ↪ (:,5)))];

T = table(n1,n2,n3,n4,n5,'RowNames',T_Column);

format shortG;
disp(T);

% Quality of results and display (Example)
[f,ypredict_ens,y_measure,Sigma_ens] = fun(x);
disp('La variance du modele est:');
disp(Sigma_ens);
```

7 Calibration of Manning coefficients: test cases

In this section we present examples allowing us to evaluate the performance and robustness of the methodology and algorithms previously developed for the effective calibration of Manning coefficients, specifically in the context of subcritical flows. For each test, Manning coefficients were derived from the reference values. Sampling was carried out using the Sobol algorithm, with the search intervals defined by taking these reference values as target solutions, and considering a

uniform distribution with a fixed coefficient of variation **cv** at 5 %. The files that will be used in this section are located in the folder **CuteFlow/Calibration_manning/docs**

7.1 Flow in a divergent channel

First, we will evaluate the effectiveness of the procedure on a simple domain with variable geometry. The chosen canal has a total length of 16 meters, and consists of a narrow section 6 meters long by 1 meter wide, as well as a wider section 10 meters long by 2 meters wide. The subdivision of the mesh is represented by the figure below:

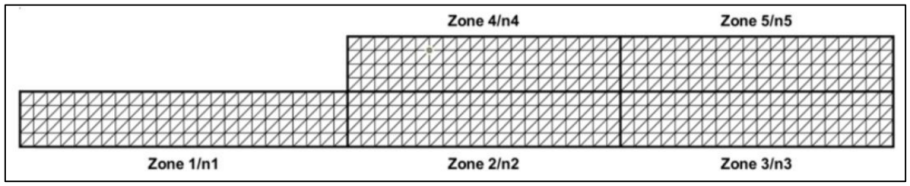


Figure 128: Divergent channel domain and decomposition into 5 subdomains

7.1.1 Initialization and configuration

A preliminary analysis made it possible to identify areas of interest, suitable for locating measurement points. In practice, the hydraulic engineer must have good control of the simulated watercourse.
The results are collected and compared in 108 points, strategically distributed along one horizontal line and three respective vertical lines shown in the figure 129.

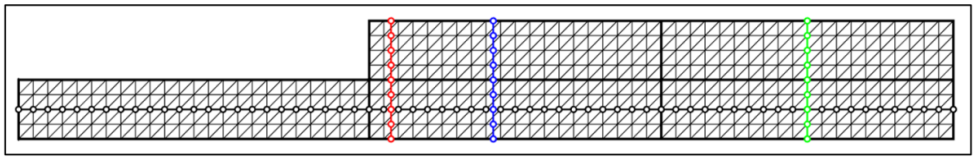


Figure 129: Location of measurement points - Divergent channel

The simulation and calibration parameters are listed in the following table:

Table 1: Calibration Parameters - Divergent Channel

Number of elements	832
Number of mesh nodes	4893
Number of parameters to calibrate	5
Number of samples	256
Overall model size	10
Observation points	108

Zone	Bathymetry	Search interval	Desired parameter
n_1	Concrete	$[1,26-1,54]\times10^{-2}$	$1,40\times10^{-2}$
n_2	Sand	$[1,53-1,87]\times10^{-2}$	$1,70\times10^{-2}$
n_3	Rock cuts	$[2,26-2,75]\times10^{-2}$	$2,50\times10^{-2}$
n_4	Paving stones	$[3,16-3,84]\times10^{-2}$	$3,50\times10^{-2}$
n_5	Gravel	$[2,53-3,07]$	$2,80\times10^{-2}$

The results obtained result from an initialization with a horizontal plane. A flow rate of $350cm^3.s^{-1}$ is imposed at the inlet and a water height $h = 20cm$ is set at the outlet.

7.1.2 Results

The overall model was built from water height measurements. To do this, a set of 10 forward propagation networks was trained by initializing the weights randomly. The individual outputs were averaged to obtain the ensemble output solution. The values obtained for the Manning coefficients, with both models, are listed in the table below. They turn out to be remarkably satisfactory, displaying maximum relative errors of less than 5

Table 2: Calibration results - Divergent channel

Zone	n_1	n_2	n_3	n_4	n_5
Target value	$1,4 \times 10^{-2}$	$1,7 \times 10^{-2}$	$2,5 \times 10^{-2}$	$3,5 \times 10^{-2}$	$2,8 \times 10^{-2}$
Genetic Algorithm (GA)					
Calibrated value	$1,42 \times 10^{-2}$	$1,64 \times 10^{-2}$	$2,42 \times 10^{-2}$	$3,52 \times 10^{-2}$	$2,74 \times 10^{-2}$
Relative error (%)	1,23	3,32	3,26	0,43	2,00
Population Swarm Optimization (PSO)					
Calibrated value	$1,39 \times 10^{-2}$	$1,62 \times 10^{-2}$	$2,62 \times 10^{-2}$	$3,60 \times 10^{-2}$	$2,92 \times 10^{-2}$
Relative error (%)	1,11	4,92	4,64	2,93	4,31

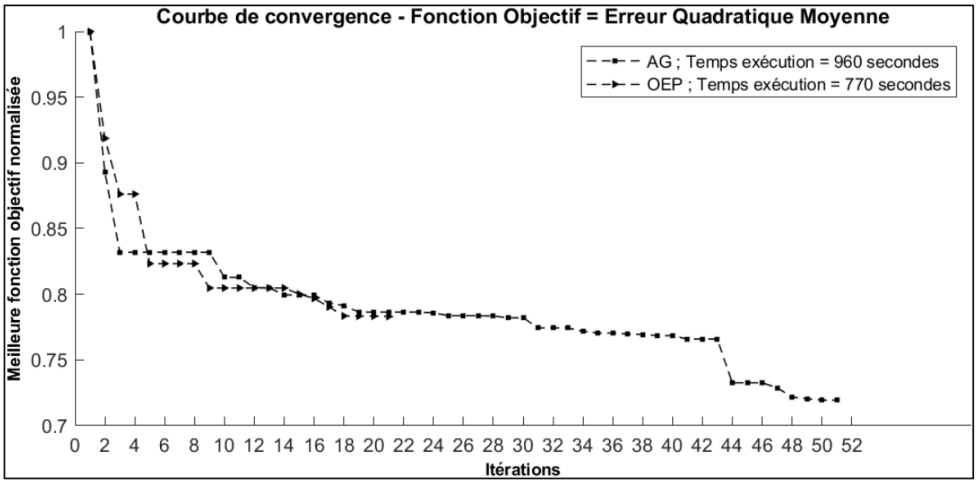


Figure 130: Convergence history of optimization algorithms (AG and PSO) - Divergent channel

The calibration carried out by the genetic algorithm required 960 seconds for 51 iterations, while the calibration by particle swarm optimization required 770 seconds for 21 iterations. These results clearly demonstrate the efficiency and speed inherent in the proposed methodology for the precise identification of Manning coefficients.

7.2 Flow in a river with non-submersible obstacles

The second test is carried out on a river area with stone obstacles. The goal is to evaluate the robustness of the methodology in a context of complex geometry, approaching reality, while remaining relatively simple to reproduce. The dimensions of the canal are 28.5 meters by 20 meters, with a constant bathymetry, depth 4 meters, and the presence of two cylinder-shaped obstacles, each having a diameter of 4 meters. Figure 131 illustrates a decomposition of the domain into 4 subdomains, as well as the position of the 28 measurement points. These measurement points were chosen strategically (i.e., before the obstacles, after the obstacles and around the obstacles).

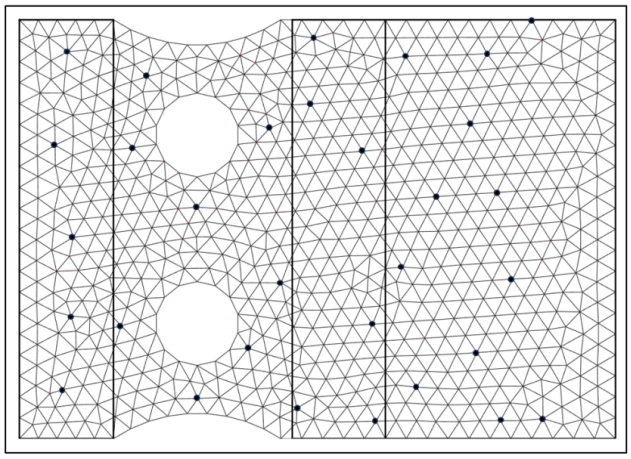


Figure 131: Breakdown of the river with obstacle into 4 subdomains and location of measurement points

7.2.1 Initialization and configuration

For this experimental configuration, a flow rate of $62m^3.s^{-1}$ is imposed at the inlet and the water level fixed at $h = 0.0m$ at the outlet. The flow simulation was initialized from a horizontal plane and calculated over a period of 6 minutes. The initial calibration parameters are presented in the table below:

Table 3: Calibration parameters - River with stone obstacles

Number of elements	1408
Number of mesh nodes	767
Number of parameters to calibrate	4
Number of samples	256
Overall model size	10
Observation points	28

Zone	Bathymetry	Search interval	Desired parameter
n_1	Fine gravel	$[2,16 - 2,64] \times 10^{-2}$	$2,40 \times 10^{-2}$
n_2	Gravel	$[3,16 - 3,84] \times 10^{-2}$	$3,50 \times 10^{-2}$
n_3	Sand	$[1,80 - 2,20] \times 10^{-2}$	$2,00 \times 10^{-2}$
n_4	Paving stones	$[2,71 - 3,29] \times 10^{-2}$	$3,00 \times 10^{-2}$

7.2.2 Results

The results obtained were also generated from a substitution model based exclusively on the water height variable and are summarized below.

Table 4: Calibration results - River with non-submersible stone obstacles

Zone	n_1	n_2	n_3	n_4
Target value	$2,4 \times 10^{-2}$	$3,5 \times 10^{-2}$	$2,0 \times 10^{-2}$	$3,0 \times 10^{-2}$
Genetic Algorithm (GA)				
Calibrated value	$2,38 \times 10^{-2}$	$3,51 \times 10^{-2}$	$2,08 \times 10^{-2}$	$2,97 \times 10^{-2}$
Relative error (%)	0,77	0,33	4,04	1,16
Population Swarm Optimization (PSO)				
Calibrated value	$2,42 \times 10^{-2}$	$3,51 \times 10^{-2}$	$2,09 \times 10^{-2}$	$2,96 \times 10^{-2}$
Relative error (%)	0,64	0,21	4,44	1,53

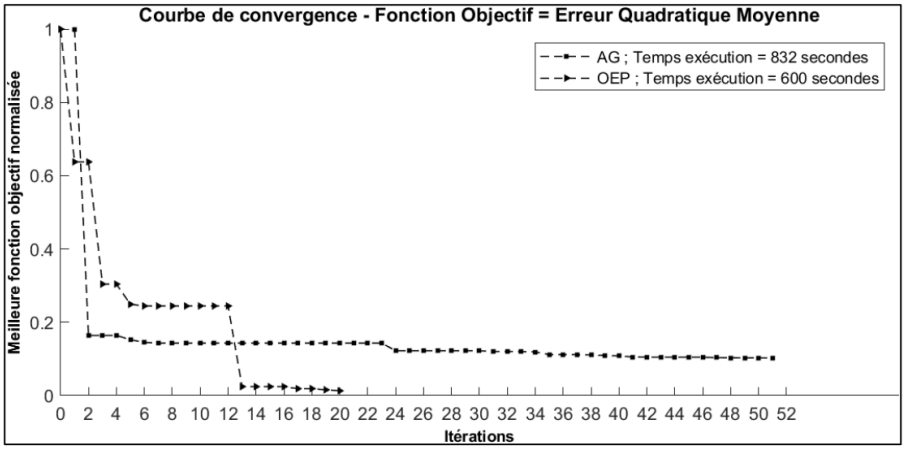


Figure 132: Convergence history of optimization algorithms (AG and PSO) - River with non-submersible stone obstacle

The evolution of the convergence curves of the different approaches, accompanied by their respective execution times, allows us to observe that particle swarm optimization converges more quickly with fewer iterations compared to the genetic algorithm. Furthermore, the results obtained with the two methods demonstrate equivalent satisfaction, with no notable distinction between one and the other.

7.3 Case of the Mille - Îles River

In this section, we evaluate the capacity of the proposed methodology to effectively identify the values of Manning's roughness coefficients on a complex and real bathymetry. The mesh is dense and made up of several elements.

References

[1] URL: https://git-scm.com/. (23/02/2020).

[2] Riadh Ata et al. "A Weighted Average Flux (WAF) scheme applied to shallow water equations for real-life applications". In: *Advances in Water Resources* 62 (2013), pp. 155–172. ISSN: 0309-1708. DOI: https://doi.org/10.1016/j.advwatres.2013.09.019. URL: https://www.sciencedirect.com/science/article/pii/S0309170813001802.

[3] Utkarsh Ayachit. *The ParaView Guide: A Parallel Visualization Application*. Clifton Park, NY, USA: Kitware, Inc., 2015. ISBN: 1930934300.

[4] Vincent Delmas and Azzedine Soulaïmani. "Multi-GPU implementation of a time-explicit finite volume solver using CUDA and a CUDA-Aware version of OpenMPI with application to shallow water flows". In: *Computer Physics Communications* 271 (2022), p. 108190. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2021.108190. URL: https://www.sciencedirect.com/science/article/pii/S0010465521003027.

[5] Igor Gildas Metcheka Kengne. *Calibration du coefficient de frottement de Manning par des algorithmes d'optimisation et des modèles réduits de réseaux de neurones*. 2023.

[6] Youssef Loukili and Azzeddine Soulaïmani. "Numerical Tracking of Shallow Water Waves by the Unstructured Finite Volume WAF Approximation". In: *International Journal for Computational Methods in Engineering Science and Mechanics* 8 (May 2007). DOI: 10.1080/15502280601149577.

[7] Arun Suthar and Azzeddine Soulaïmani. *Internship report parallelization of shallow water equations solver: CuteFlow*. July 2018.

[8] E. F. Toro et al. "A flux-vector splitting scheme for the shallow water equations extended to high-order on unstructured meshes". In: *International Journal for Numerical Methods in Fluids* n/a.n/a (). DOI: https://doi.org/10.1002/fld.5099. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.5099. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.5099.

[9] Jean-Marie Zokagoa and Azzeddine Soulaïmani. "Modeling of wetting–drying transitions in free surface flows over complex topographies". In: *Computer Methods in Applied Mechanics and Engineering* 199.33 (2010), pp. 2281–2304. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2010.03.023. URL: http://www.sciencedirect.com/science/article/pii/S0045782510001003.