

Guide d'utilisation de la version multi-GPU de CUTEFLOW

GRANIT – Groupe de recherche sur les applications numériques en ingénierie
et en technologie à l'École de Technologie Supérieure, Montréal

26 mars 2025

Table des matières

1 Avant-propos	3
1.1 Utilisation des clusters de calcul de <i>Compute Canada</i>	3
1.2 Gestion des versions avec Git	3
2 Fichiers de maillages	4
2.1 Conversion des fichiers de maillages vers le format CGNS	4
2.2 Création d'un fichier de maillage depuis Python	6
2.3 Création d'un fichier de maillage avec <i>GMSH</i>	7
2.4 Ajout d'un nombre de Manning sur chaque maille du fichier de maillage	11
2.4.1 Création du fichier "Entree_gen_manning.txt" préliminaire à l'ajout des nombres de Manning	12
2.5 Découpe du maillage en sous domaines : <i>parmetis_ghost</i>	17
3 CuteFlow	18
3.1 Fichiers sources et compilation	18
3.2 Description du fichier donnees.f	19
3.3 Lancement d'une simulation	22
3.3.1 Lancement d'une simulation à partir d'une solution déjà générée	24
3.4 Lancement de plusieurs simulations en simultané	24
3.4.1 Génération des fichiers de données	24
3.4.2 Préparation du dossier <i>base_files</i>	24
3.4.3 Lancement du script <i>multi.sh</i>	25
3.4.4 Lancement du job array	26
4 Traitement dans Paraview des fichiers *.vtk et *.cgns	28
4.1 Tunnel SSH pour la visualisation avec Paraview sur les clusters de calcul en utilisant MobaXterm	28
4.2 Visualisation des fichiers résultats générés par CUTEFLOW sur Paraview	31
4.2.1 Visualisation du fichier <i>rect_6138</i>	31
4.2.2 Visualisation du fichier <i>rect_6138_bump</i>	36
4.2.3 Visualisation du fichier <i>out_rect_95459_bump.cgns</i>	37
4.2.4 Visualisation du fichier <i>mille_700k_2_2.cgns</i>	41
5 Cas Tests	52
5.1 Cas de bris de barrage fictifs	52
5.1.1 Solutions exactes et de référence	52
5.1.2 Simulation sur <i>mesh_6138</i> - Test 1	53
5.1.3 Simulation sur <i>mesh_6138</i> - Test 2	54
5.1.4 Simulation sur <i>mesh_6138</i> - Test 3	56
5.2 Cas de bris de barrage fictif avec bump	58
5.3 Cas du bris de barrage circulaire	60
5.4 Cas de la rivière des Milles Îles	65
5.5 Cas de l'archipel de Montréal	68
5.5.1 Manning constant	68
5.5.2 Manning variable	70
6 Calibration du nombre de Manning avec CuteFlow	73
6.1 Préparation des données et résultats numériques	74
6.1.1 Zonage du maillage et affectation des nombres de Manning	74
6.1.2 Ajout des nombres de Manning aux fichiers de maillage (<i>mesh.cgns</i>)	76

6.1.3	Lancement des calculs : job array	78
6.2	Post-traitement des fichiers solutions <i>out_*.cgns</i> et extraction des données	79
6.2.1	Visualisation des fichiers solutions générés par CuteFlow	79
6.2.2	Extraction des données	79
6.3	Création et entraînement du modèle d'ensemble	82
6.4	Optimisation des paramètres	83
7	Calibration des coefficients de Manning : cas tests	87
7.1	Écoulement dans un canal divergent	87
7.1.1	Initialisation et paramétrage	87
7.1.2	Résultats	88
7.2	Écoulement dans une rivière avec obstacle non submersibles	88
7.2.1	Initialisation et paramétrage	88
7.2.2	Résultats	89
7.3	Cas de la rivière des Mille - Îles	90
7.3.1	Initialisation et paramétrage	91
7.3.2	Résultats	91

1 Avant-propos

On présente dans la suite le guide d'utilisation du code *CuteFlow* et des codes de préé/post-traitement associés. Ce code est hébergé sur Github <https://github.com/ETS-GRANIT/CuteFlow>.

CuteFlow est un code de calcul parallèle par éléments-volumes finis développé dans le **GRANIT** à l'ETS. Il est dédié à la résolution des équations de St-Venant et est principalement utilisé pour simuler les ruptures de barrages, les écoulements dans les rivières, les inondations et la détermination des côtes de crues.

Depuis sa version initiale, *CuteFlow* a connu des évolutions majeures. La première version séquentielle sur CPU a été développée dans un premier temps par Azzeddine Soulaïmani et Youssef Loukili [6] puis par Jean-Marie Zokagoa [9]. Le code a par la suite été porté sur GPU en CUDA Fortran par Arun Kumar Suthar [7]. Finalement, le code a été porté sur une architecture multi-GPU par Vincent Delmas [4]. Les avancées les plus récentes comprennent le développement d'un module de calibration des paramètres physiques de simulation (coefficient de Manning) par Igor Metcheka [5].

1.1 Utilisation des clusters de calcul de *Compute Canada*

Pour tout ce qui concerne l'utilisation des clusters de calculs, l'utilisateur peut se référer au wiki officiel de *Compute Canada* https://docs.computecanada.ca/wiki/Compute_Canada_Documentation. Sur ce wiki, on peut entre autres trouver les informations relatives à la connexion aux clusters de calculs, au stockage de données, au lancement de job et à l'allocation interactive de noeuds.

Pour plus de facilité d'utilisation des clusters de calculs, il est préférable d'être à l'aise avec un éditeur en ligne de commande comme vim, emacs ou nano et d'être capable de séparer un terminal en plusieurs terminaux, par exemple, à l'aide de tmux <https://docs.computecanada.ca/wiki/Tmux> (tout cela est installé par défaut sur les clusters de calculs).

Dans la suite, les exemples sont donnés en partant du principe que l'utilisateur est connecté sur un des clusters de calculs de *Compute Canada*, mais le code peut être téléchargé sur n'importe quel système. Pour compiler et lancer le code, il faudra cependant avoir les différentes librairies nécessaires installées et compilées de la bonne façon.

1.2 Gestion des versions avec Git

On utilise le logiciel Git [1] pour gérer les versions du code. Un tutoriel est présent sur le [site officiel](#), un autre est présent sur [OpenClassrooms](#). On présente simplement comment cloner le dossier du code et comment le tenir à jour des modifications qui y sont faites. Git permet évidemment de faire bien plus de choses, notamment, remonter dans l'historique des fichiers ou permettre à des personnes de travailler dans différentes branches du même code.

Pour cloner le projet sur GRAHAM dans l'espace scratch de l'utilisateur :

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

# Generer une cle d'authentification sur le cluster avec
ssh-keygen -t rsa -b 4096

# Voir la cle avec
cat ~/.ssh/id_rsa.pub
```

Le système demandera ensuite de créer un mot de passe associé à la clé. Il faudra le rentrer pour cloner le repo Git. La clé sera stockée dans **home/User Name/.ssh/id_rsa.pub** ou similaire.

Copier ensuite cette clé sur le GitHub ici <https://github.com/settings/keys>. Le git-clone à faire sera alors comme suit :

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

Un dossier [CuteFlow/](#) contenant les fichiers importants du code sera alors créé, l'opération peut durer quelques minutes.

Si l'utilisateur détruit par inattention certains fichiers du code, il peut toujours re-synchroniser son dossier local avec GitHub en faisant,

```
# Reset le dossier par rapport au dernier téléchargement
git reset --hard

# Télécharge les changements depuis le dossier source sur GitHub
git pull
```

Attention : de cette façon toutes les modifications apportées par l'utilisateur aux fichiers gérés par Git seront écrasées. Cela n'aura pas d'impact sur les nouveaux fichiers et dossiers que l'utilisateur aura pu créer. Il y a d'autres façons de faire pour conserver une partie des modifications, voir <https://git-scm.com/docs/gittutorial>.

2 Fichiers de maillages

2.1 Conversion des fichiers de maillages vers le format CGNS

On peut convertir différents types de fichiers de maillages vers le format CGNS en utilisant par exemple Python. Un code convertissant les anciens fichiers de maillages utilisés par CuteFlow vers le format CGNS est présent dans le dossier [CuteFlow/src/cute_to_cgns](#).

Avec le code [cute_to_cgns.py](#), on lit les anciens fichiers de maillages, en particulier, les coordonnées des noeuds, la connectivité des éléments et les noeuds d'entrée et sortie, et on utilise la bibliothèque *pyCGNS* pour écrire un fichier au format CGNS. Le code [cute_to_cgns.py](#) est accompagné d'un fichier [Readme.txt](#) qui explique comment créer l'environnement virtuel Python qui permet de lancer le code.

```
# Une fois l'environnement virtuel créé et activé
python3 cute_to_cgns.py mesh.txt multi_entree
```

où, *mesh.txt* est un fichier utilisant l'ancien format de fichier de maillages, et *multi_entree* prend la valeur 0 ou 1 selon si le fichier de maillage contient plusieurs entrées. Suite à l'exécution du code, un fichier de maillage ***mesh.cgns*** sera généré.

Le code [cute_to_cgns.py](#) a par exemple été utilisé pour convertir [Mille_Iles_mesh_743968_elts.txt](#) vers le nouveau 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 connectivites
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](#) utilisant l'ancien format *.txt lu par CuteFlow

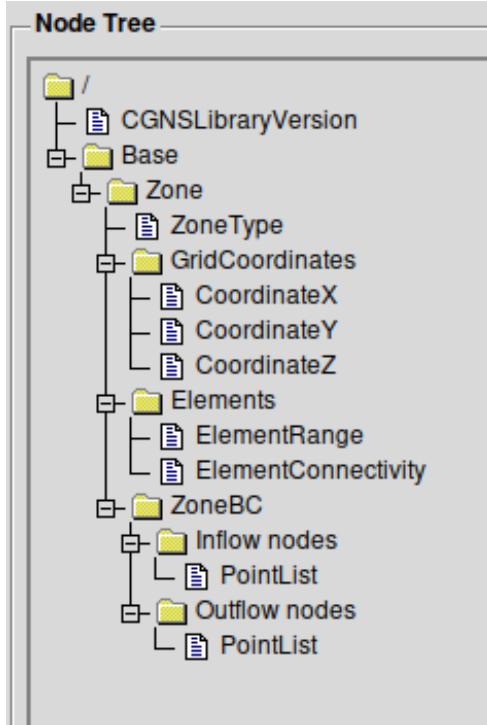


FIGURE 2 – Structure du nouveau fichier [Mille_Iles_mesh_743968_elts.cgns](#) au format *.cgns

Dans l’ancien format de fichier présenté dans la Figure 1, on distingue plusieurs sections : Une table de coordonnées contenant les noeuds numérotés en commençant à 1 avec leurs coordonnées x , y , z et un nombre de Manning qui est maintenant obsolète (un nombre de Manning défini sur une zone doit maintenant être ajouté comme présenté dans la section 2.4) ; Une table de connectivité où les éléments sont numérotés en commençant à 1 et où pour chaque élément (triangulaire) on liste les sommets de l’élément ainsi qu’un nombre de Manning qui est aussi obsolète ; Une liste des noeuds d’entrée ; Une liste des noeuds de sortie ; Une liste des noeuds de murs qui peut être oubliée. Pour chaque section, la première ligne correspond au nombre de lignes suivantes à lire. Dans le fichier [Mille_Iles_mesh_743968_elts.txt](#) on peut voir qu’il y a 374619 noeuds, 743968 éléments, 266 noeuds d’entrée, 342 noeuds de sortie et, 4708 noeuds de murs. Ce fichier sera converti par le code [cute_to_cgns.py](#) vers le fichier [Mille_Iles_mesh_743968_elts.cgns](#) dont la structure est présentée sur la Figure 2.

Le code [cute_to_cgns.py](#) sera amené à être modifié et doit servir de base pour la conversion de différents types de fichiers de maillages vers le format CGNS lisible par *CuteFlow*.

2.2 Création d’un fichier de maillage depuis Python

On peut créer des fichiers de maillages au format CGNS depuis Python en utilisant la bibliothèque *pyCGNS*. Un code est donné en exemple dans le dossier [CuteFlow/src/gen_mesh/](#). Le code *gen_mesh.py* est accompagné d’un fichier *Readme.txt* qui explique comment créer un environnement virtuel pour pouvoir lancer le code.

Dans le code *gen_mesh.py*, on commence par créer une liste de points dans un domaine rectangulaire et on utilise *scipy.Delaunay* pour générer la triangulation. On peut ensuite utiliser la bibliothèque *pyCGNS* pour créer la structure du fichier CGNS en commençant par créer une base puis une zone et y ajouter les coordonnées des noeuds et la connectivité des éléments avant de sauvegarder le fichier au format CGNS. On peut trouver la documentation complète de *pyCGNS* sur le lien, <https://pycgns.github.io/index.html> en particulier les modules **MAP** et **PAT**.

Une fois l’environnement virtuel créé et activé, on peut lancer le code [gen_mesh.py](#) de la façon suivante :

```
#Une fois l'environnement virtuel cree et active
python3 gen_mesh.py
```

Un fichier de maillage de sortie sera alors généré en fonction des paramètres entrés dans le fichier [gen_mesh.py](#). Les valeurs $x0$, $x1$, $y0$ et $y1$ définissent les bornes spatiales du domaine, ny correspond au nombre de points selon l’axe y . Le maillage sera généré de manière la plus homogène possible en utilisant des triangles équilatéraux. Des exemples de maillages générés de cette manière sont présents dans le dossier [CuteFlow/meshes/rect_mesh_convergence/](#).

2.3 Création d'un fichier de maillage avec *GMSH*

On présente dans cette section comment créer un fichier de maillage simple à l'aide du logiciel *gmsh* <https://gmsh.info/>. On aura besoin de la dernière version de *gmsh* pour pouvoir générer des fichiers au format CGNS pour la lecture avec *CuteFlow*. On pourra avoir besoin du logiciel CGNSview https://cgns.github.io/CGNS_docs_current/cgnstools/cgnsview/index.html pour modifier à la main les fichiers de maillages si besoin.

On crée dans cette section un maillage simple de deux étages rectangulaires connectés par un canal. On commence par ajouter les points sur le contour du maillage en allant dans la section "Geometry->Elemental Entities->Add->Points". Pour cet exemple, nous ajoutons les points de coordonnées (0, 0, 10), (0, 10, 10), (10, 10, 10), (10, 0, 10), (10, 7, 10), (10, 3, 10), et (20, 0, 0), (20, 3, 0), (20, 7, 0), (20, 10, 0), (30, 10, 0), (30, 0, 0).



FIGURE 3 – Ajout des points dans le domaine

On peut ensuite relier les points par des lignes en allant dans "Geometry->Elemental Entities->Add->Lines".

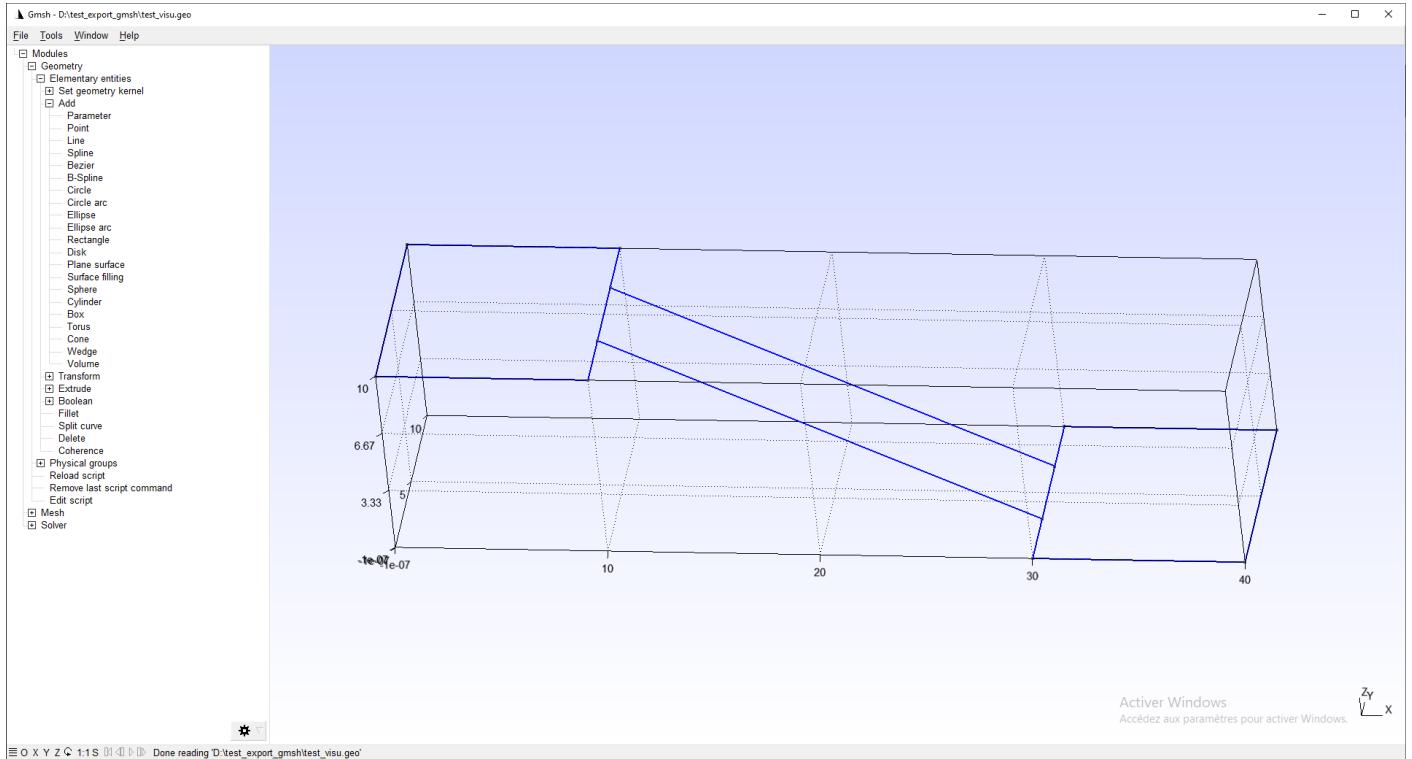


FIGURE 4 – Ajout des lignes dans le domaine

On peut ensuite créer des surfaces planes sur chaque section du maillage en allant dans "Geometry->Elemental Entities->Add->Plane Surfaces".

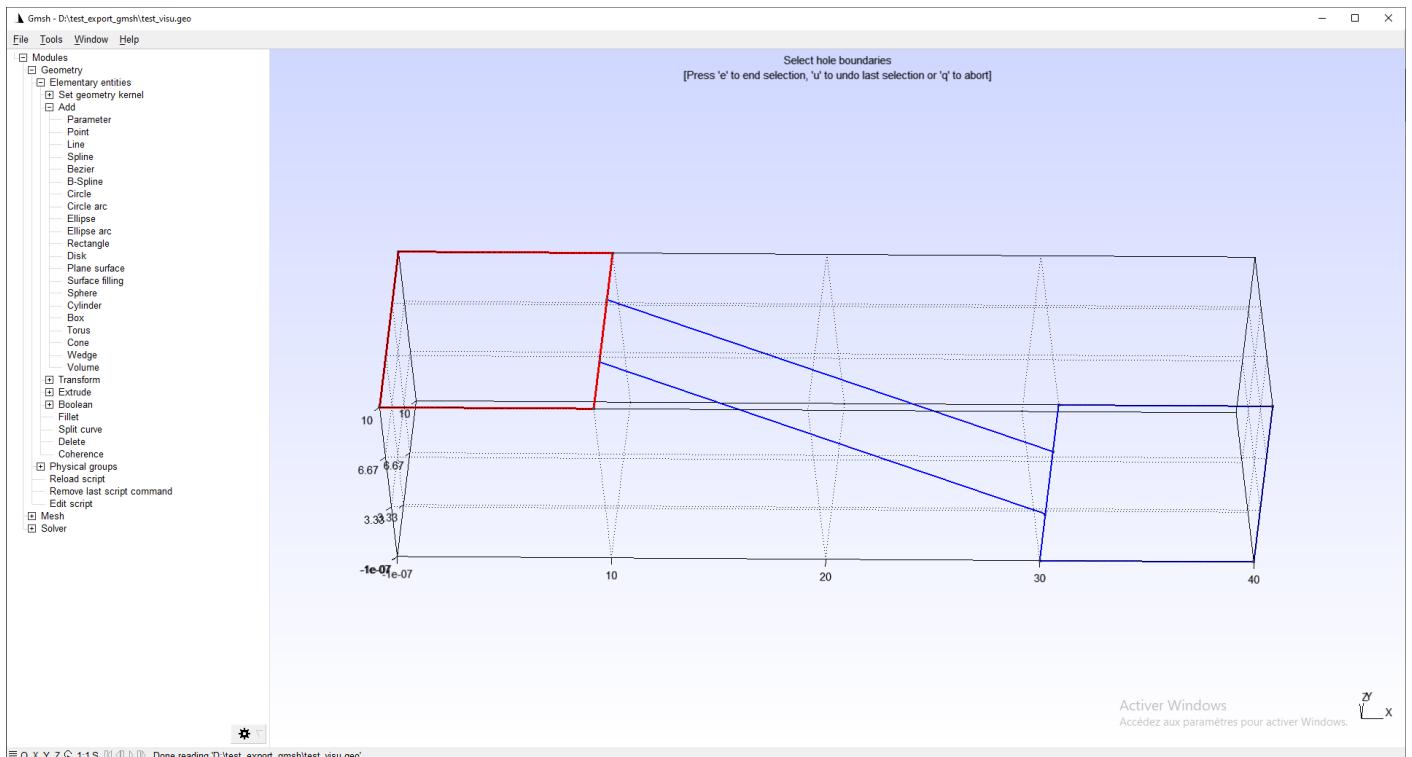


FIGURE 5 – Ajout de la surface 1

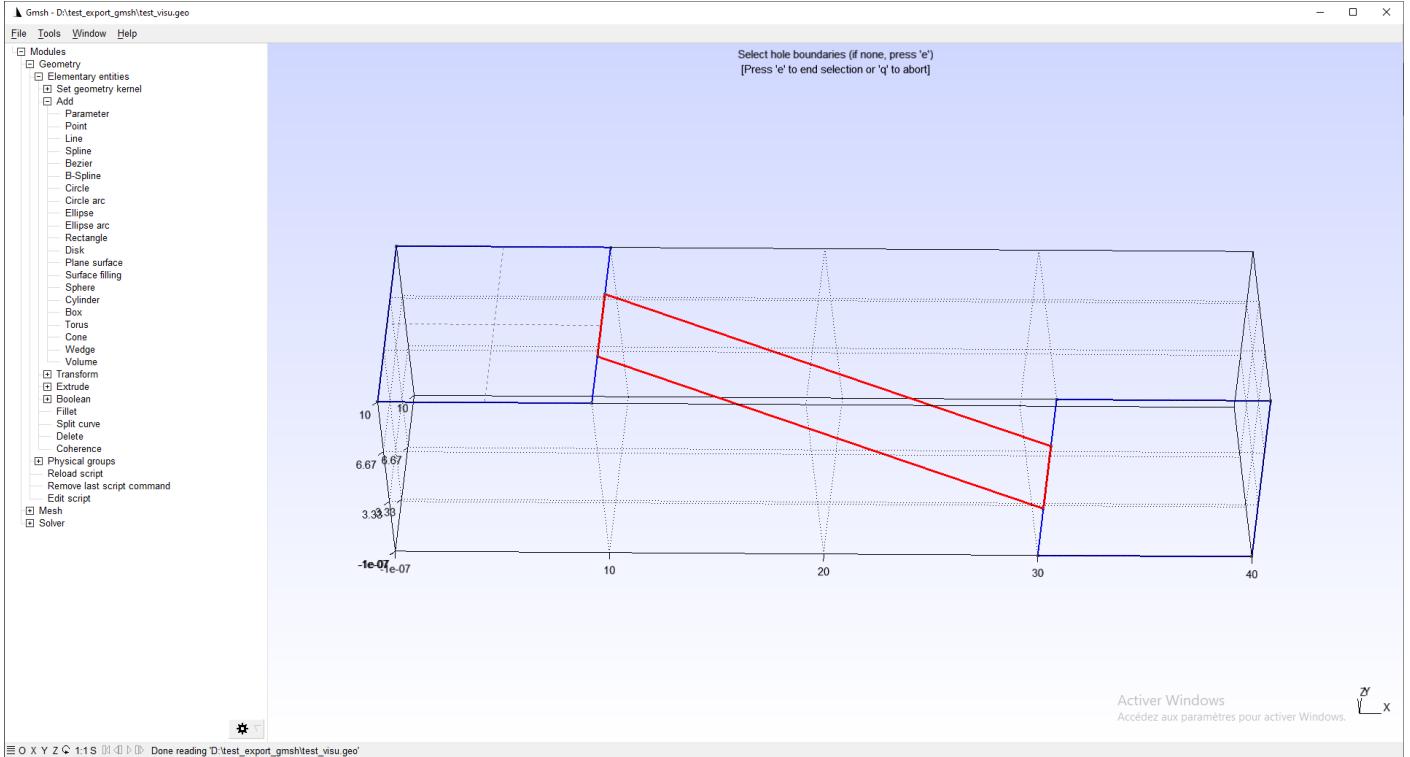


FIGURE 6 – Ajout de la surface 2

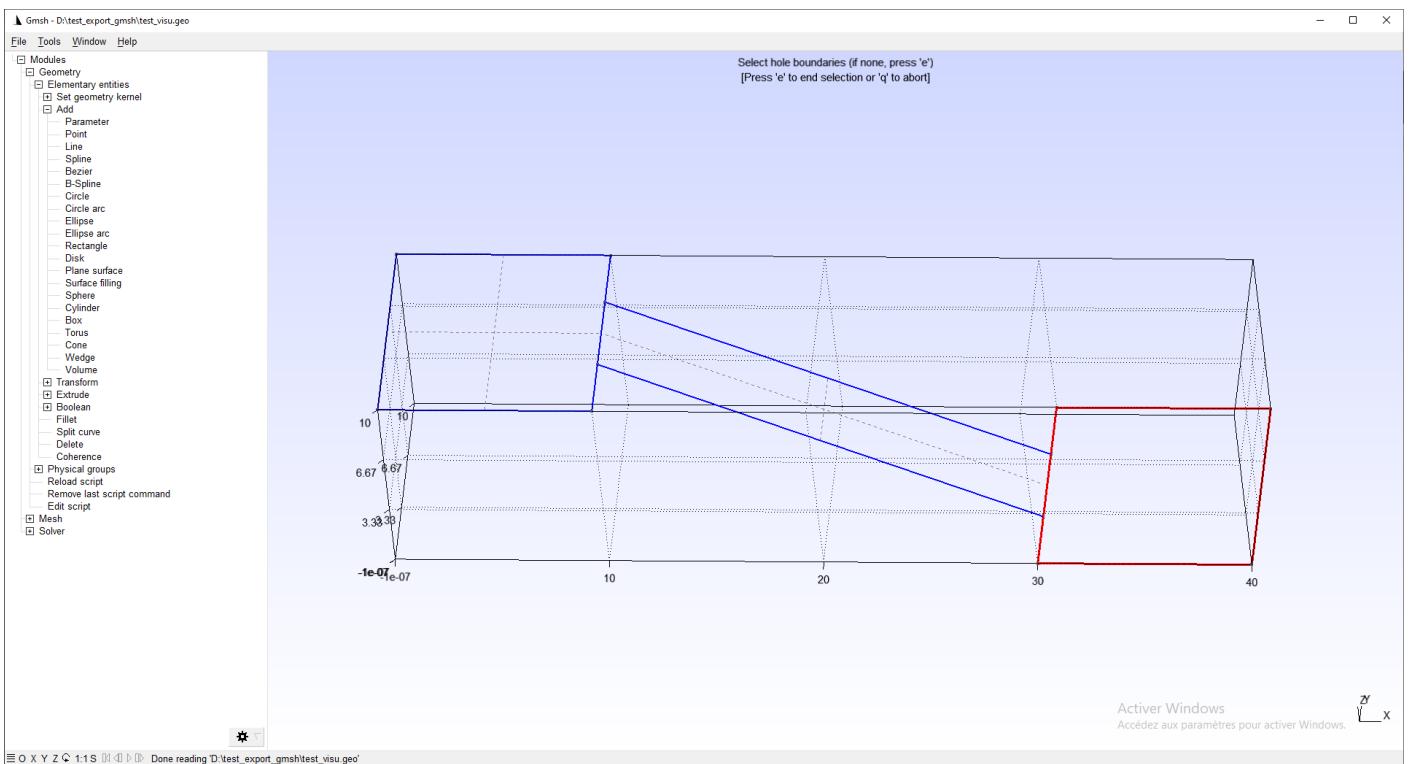


FIGURE 7 – Ajout de la surface 3

Si on va maintenant dans "Mesh->2D" un maillage séparé des trois surfaces sera créé. C'est un problème car on veut réunir les trois surfaces en une seule. Pour cela, il va falloir aller dans "Mesh->Define->Comound->Surface" et sélectionner les trois surfaces à combiner.

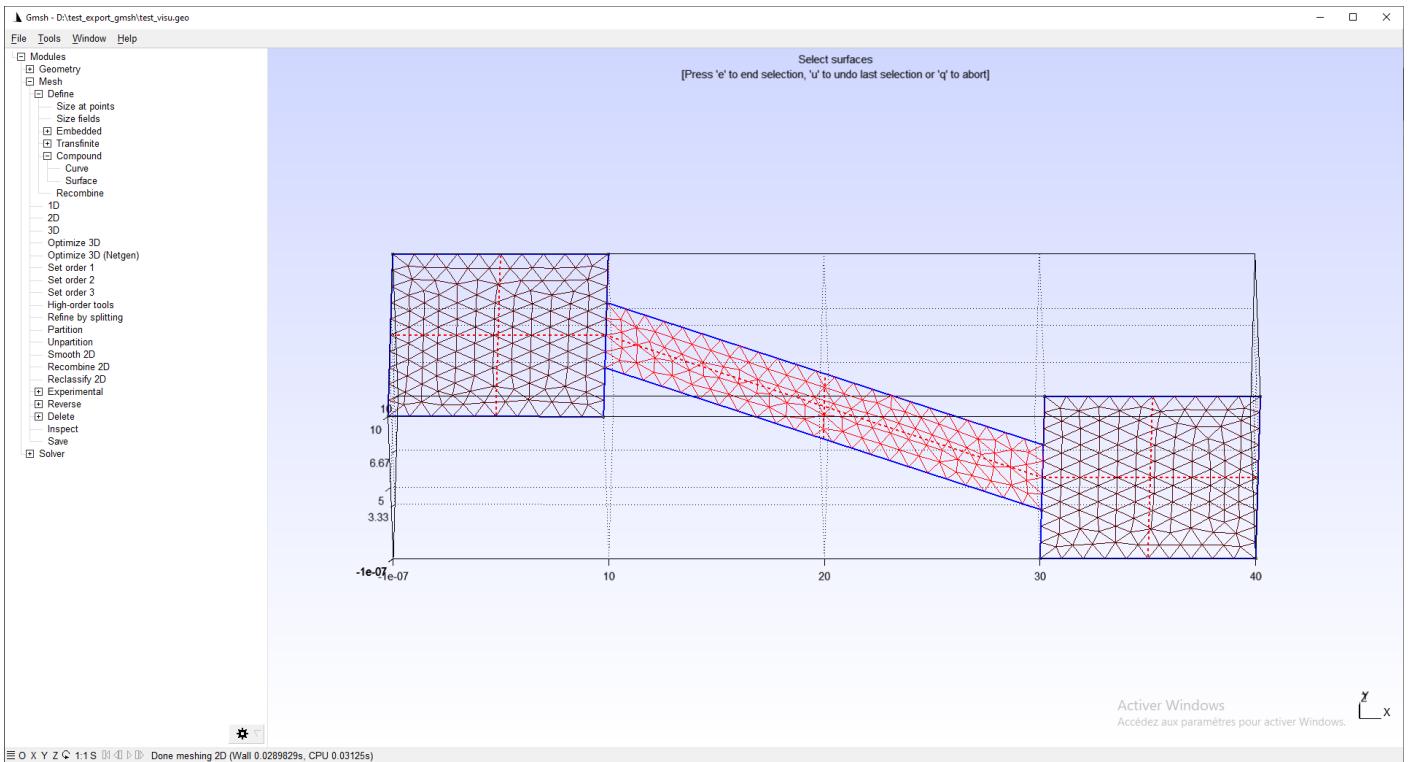


FIGURE 8 – Création de la surface commune

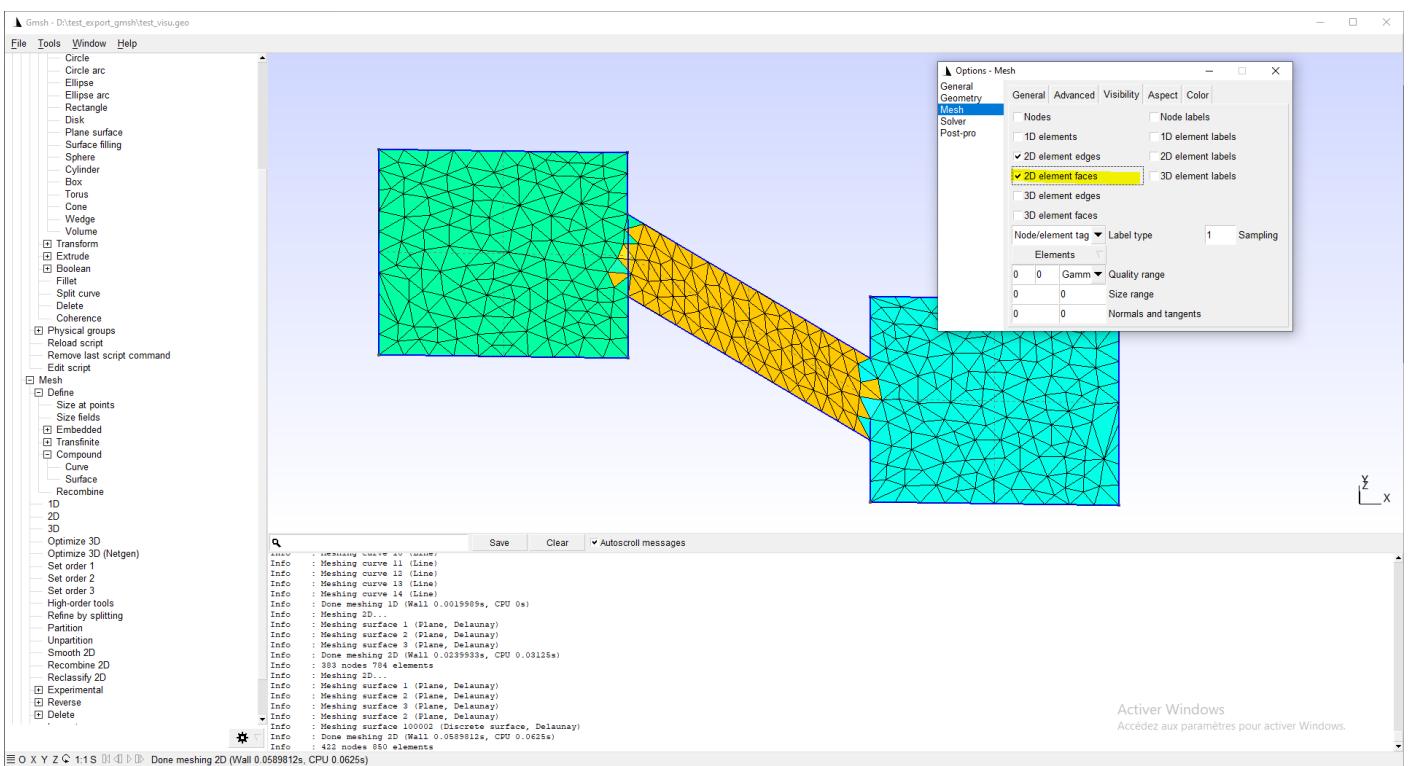


FIGURE 9 – Visualisation de la surface

On peut voir que les trois surfaces sont combinées en une seule, cependant, on voit que la surface ne passe pas exactement par le bord des deux étages. Pour arranger cela, il faut aller dans "Mesh->Define->Embedded->Curve" et intégrer à la surface les lignes intérieures.

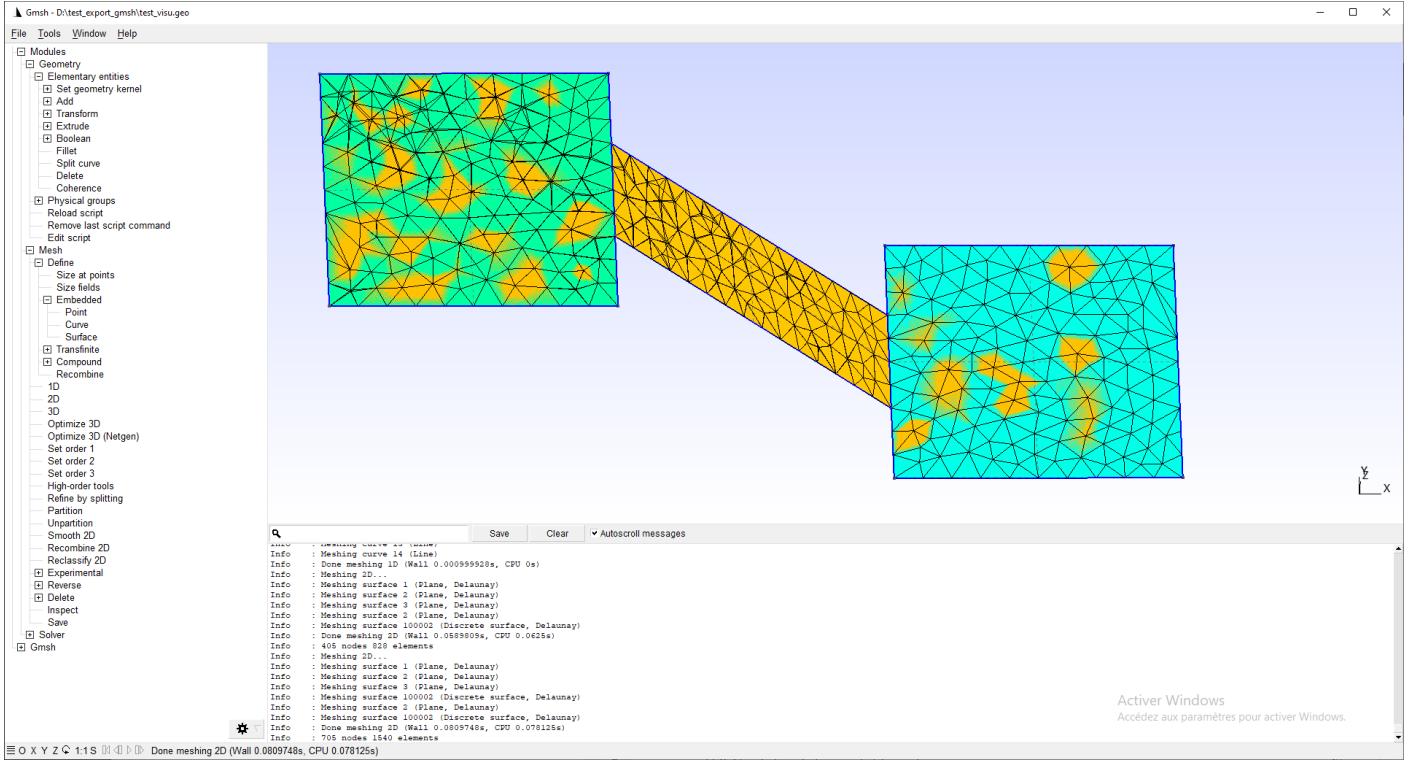


FIGURE 10 – Correction de la surface pour passer par les lignes intérieures

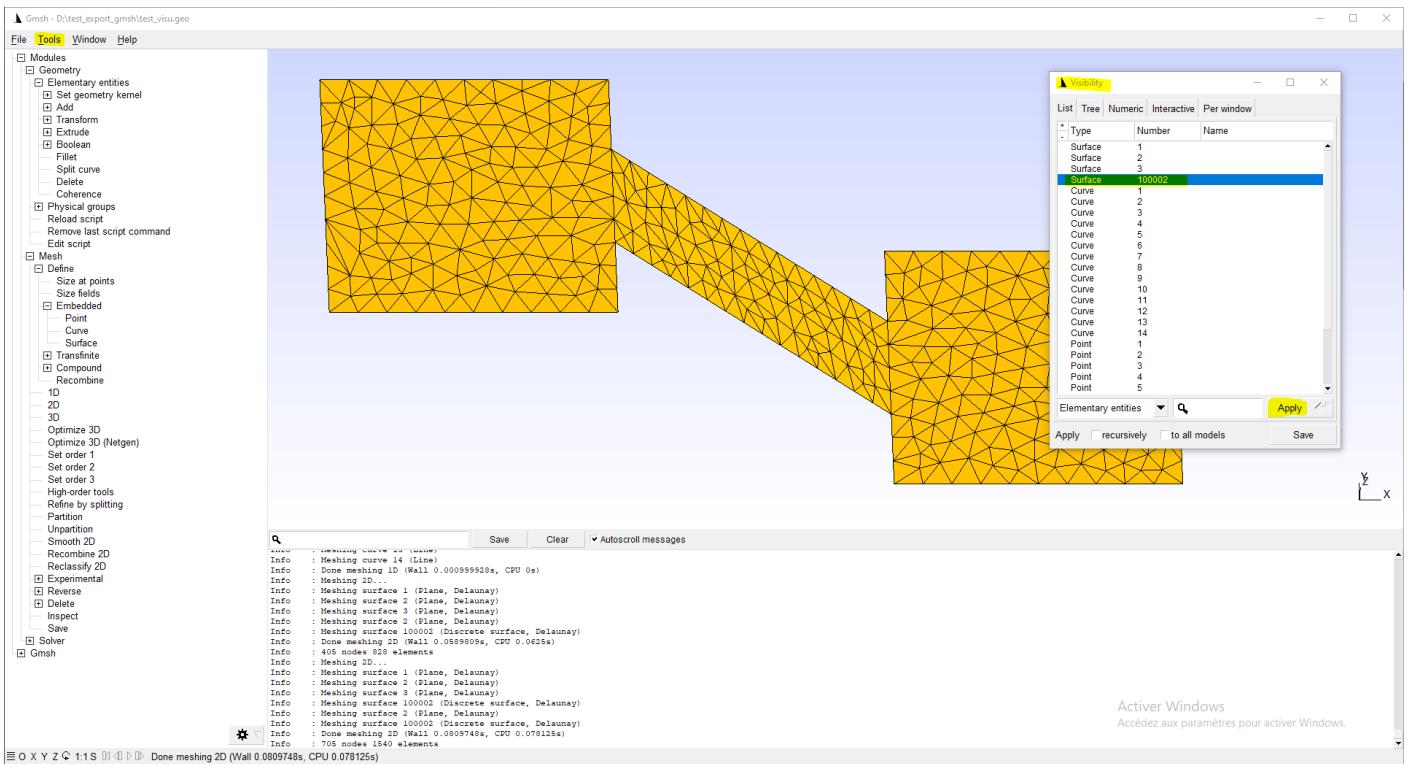


FIGURE 11 – Visualisation uniquement de la surface recombinée

À ce moment-là, le maillage est prêt à être exporté au format CGNS et lu soit par le code de décomposition de domaine, soit par *CuteFlow*.

2.4 Ajout d'un nombre de Manning sur chaque maille du fichier de maillage

La majorité du temps, le nombre de Manning est défini comme constant à l'aide du paramètre *is_override_manning* mis à 1. Si l'utilisateur souhaite définir des nombres de Manning différents pour chaque maille, la démarche est plus complexe. On utilise un code Python pour écrire un nombre de Manning sur chaque maille du fichier de maillage. Ce code peut être

trouvé dans le dossier [CuteFlow/src/gen_manning/](#). Un fichier [Readme.txt](#) explique rapidement la procédure pour créer un environnement virtuel qu'on détaille ici.

On crée l'environnement virtuel dans lequel on pourra lancer le code python de la façon suivante :

```
# Creation de l'environnement virtuel

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
```

Après création de l'environnement virtuel, on met à jour les bibliothèques installées avec le code `upgrade_bib.py` :

```
# Mise a jour des bibliotheques installees
python3 upgrade_bib.py
```

```
# Pour sortir de l'environnement
deactivate

# Pour re-entrer dans l'environnement
module load gcc cgns python/3.8.10
source ENV/bin/activate
```

Une fois l'environnement activé, on utilise le code ainsi :

```
# Creer les fichiers mesh.vtk et mesh.cgns
python3 gen_manning.py mesh.cgns
```

Le fichier `mesh.vtk` peut être ouvert dans Paraview afin de visualiser les nombres de Manning. Une fois le fichier `mesh.cgns` généré, il peut être utilisé avec `CuteFlow`.

Note : La modification du nombre de Manning peut rendre la simulation avec `CuteFlow` difficile en provoquant des instabilités. Il ne faut pas utiliser des nombres de Manning inférieurs à 0 ou supérieurs à 0.5. Pour réduire les instabilités, on peut diminuer le nombre CFL ou augmenter la tolérance sec/mouillé tolisec.

Dans le code `gen_manning.py`, on spécifie les index (numéros) des mailles du domaine avec les nombres de Manning qu'on souhaite leur attribuer. Les numéros des mailles sont définis dans la liste `cell_index` et les valeurs du coefficient de Manning dans la liste `manning_value`. Ces données sont lues dans le code à partir d'un fichier texte, `Entree_gen_manning.txt` dont la structure et la procédure de création sont détaillés ci-dessous.

La variable `average_manning`, quant à elle permet de définir le nombre de Manning prééminent dans le domaine, ou encore le nombre de Manning à assigner aux cellules non identifiées dans le fichier `Entree_gen_manning.txt`.

2.4.1 Crédit du fichier "Entree_gen_manning.txt" préliminaire à l'ajout des nombres de Manning

Le fichier `Entree_gen_manning.txt` contient des informations sur les numéros de cellules et les valeurs du coefficient Manning respectives. Il sert de fichier d'entrée au code `gen_manning.py` et se présente comme suit :

```
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 du fichier `Entree_gen_manning.txt`

Lorsqu'on dispose déjà de ces informations, dans le cas des petits domaines d'étude par exemple, ce fichier peut être édité directement à partir d'un éditeur de texte comme Notepad. Dans le cas d'un maillage réel, on peut récupérer les identifiants des cellules d'intérêt dans le domaine en utilisant Paraview.

On présente dans la suite un exemple de création du fichier *Entree_gen_manning.txt* pour le maillage *mesh_6072.cgns*. L'objectif ici est de définir 4 zones selon les valeurs du coefficient de Manning.

La première étape consiste à identifier les différentes zones du maillage et à définir un nombre de Manning pour chacune d'elles. La figure ci-dessous illustre un exemple de cartographie de Manning que l'on souhaite obtenir pour le maillage considéré.

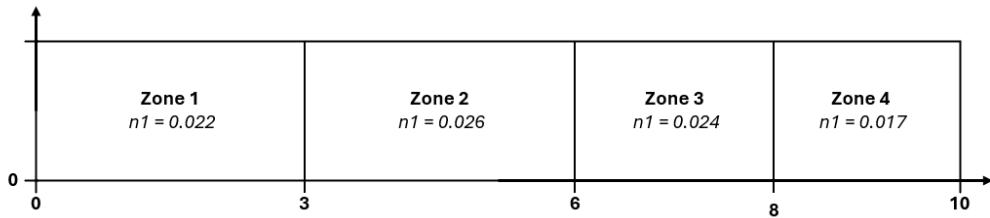


FIGURE 13 – Cartographie (zonage) du maillage *mesh_6072.cgns* en fonction des valeurs des coefficients de Manning

Par la suite, ouvrir le fichier de maillage dans paraview, sélectionner les cellules en respectant la cartographie établie et extraire les identifiants des cellules en définissant les zones comme suit :

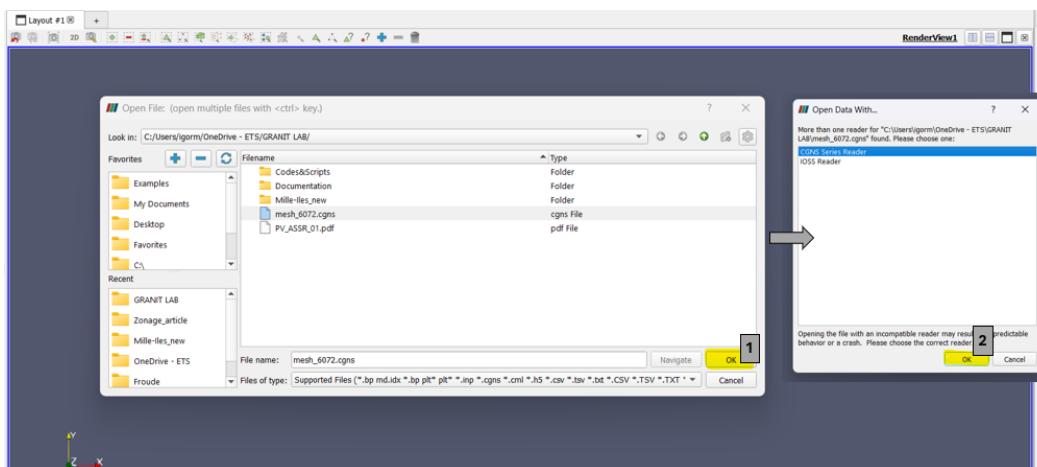


FIGURE 14 – Ouverture du fichier *mesh_6072.cgns* et choix du lecteur de fichier (cgns series reader)

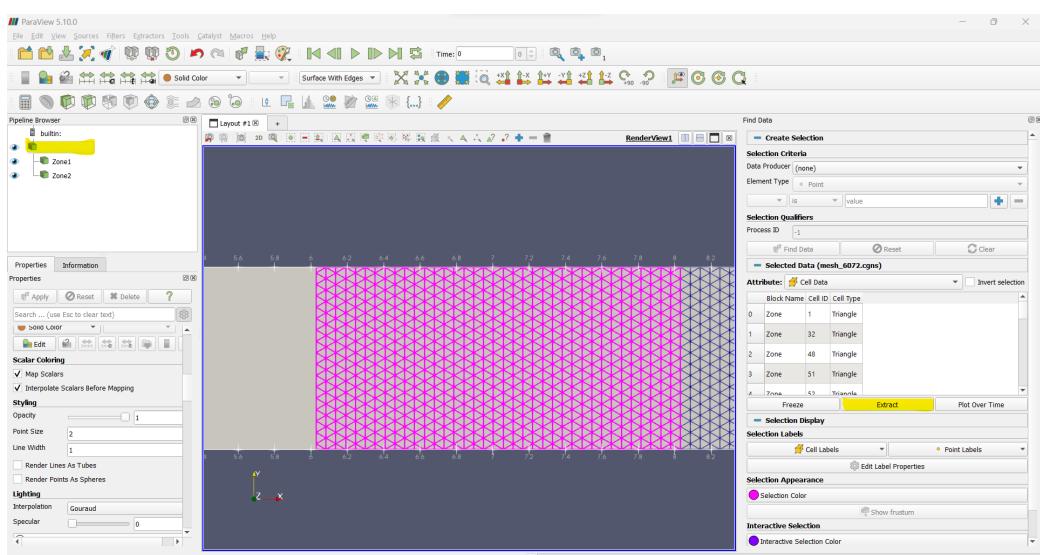


FIGURE 16 – Sélection et extraction des cellules du maillage avec les outils de sélection

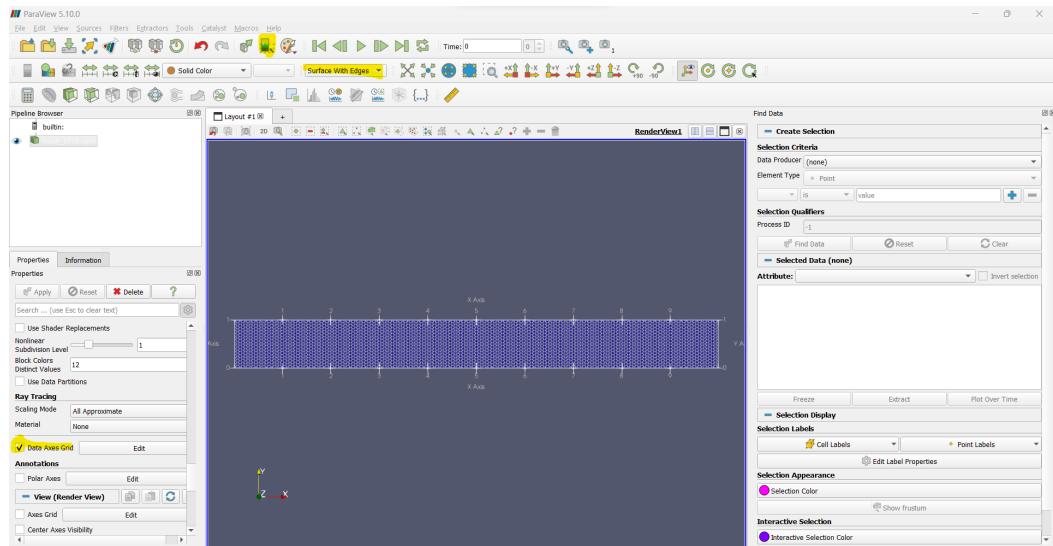


FIGURE 15 – Ouverture de la fenêtre de sélection des données

Après cette étape, on exporte les identifiants (IDs) des cellules du maillage dans des fichiers Excel pour chaque zone définie. Pour ce faire, ouvrir une nouvelle fenêtre en mode SpreadSheetView, puis exporter les données comme suit :

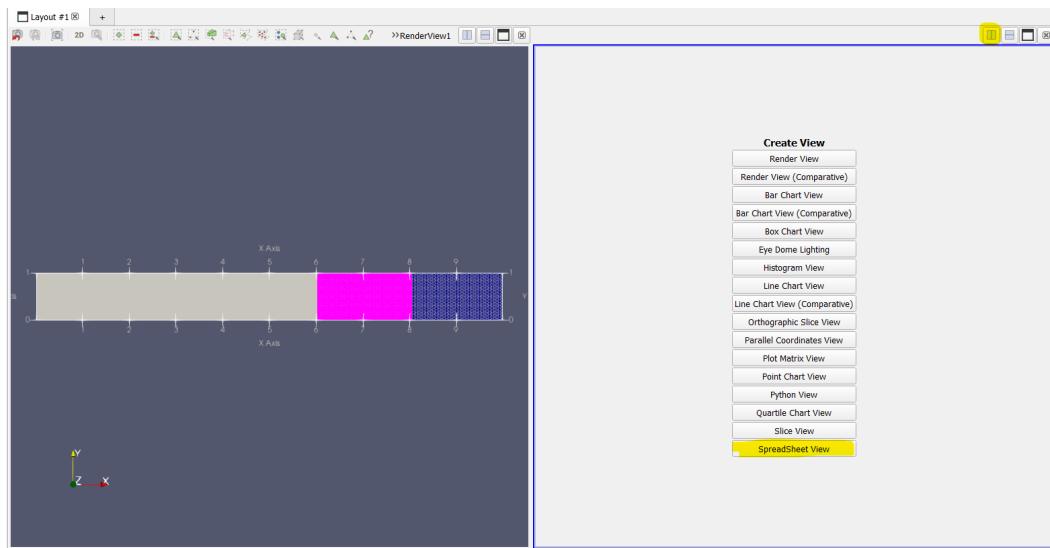


FIGURE 17 – Ouverture de la fenêtre *SpreadSheetView*

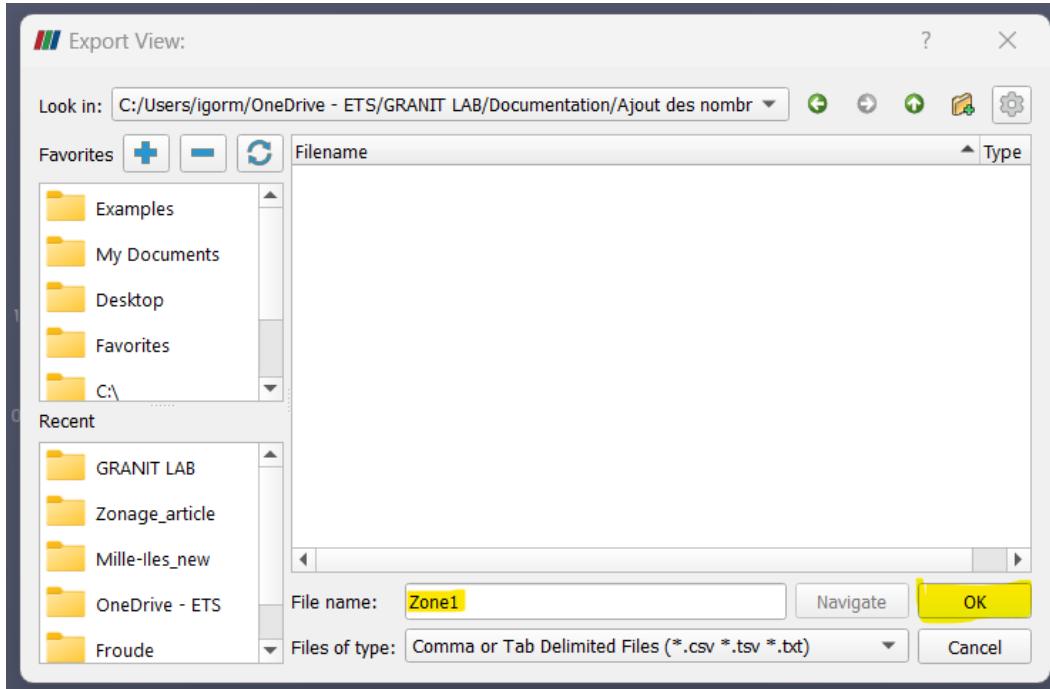


FIGURE 19 – Enregistrer les fichiers contenant les Cells_IDs de chaque zone dans un dossier de votre choix. (Exemple pour la zone1)

SpreadSheetView1		
Showing	Zone1	Attribute:
	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 – Sélectionner la zone et les informations à exporter puis cliquer sur le bouton **exporter**

Une fois que toutes les zones ont été définies et les Cells IDs correspondant extraits, nous rassemblons ces informations dans un même fichier Excel, en respectant la structure suivante :

	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 – Structure des données dans le fichier Excel de Zonage

Dans le cas du fichier mesh_6072.cgns, nous avons enregistré ce fichier sous le nom "**Zones_mesh6072.xlsx**". Une fois le fichier de zonage au format excel créé, nous pouvons l'importer dans notre espace de travail sur les clusters de Compute Canada et utiliser le code *gen_Entree_gen_manning.py* pour générer le fichier texte *Entree_gen_manning.txt*.

Nous allons réaliser cet exemple dans le dossier *CuteFlow/Calibration_manning/Exemple1*

```
# Depuis le dossier CuteFlow
cd Calibration_manning

# Creer du dossier Exemple1
mkdir Exemple1
cd Exemple1

# Telecharger le fichier Zones_mesh6072.xlsx dans le dossier courant Exemple1
# Copier le fichier de maillage dans le dossier courant
cp ../../meshes/rect_mesh_convergence/mesh_6072.cgns .

# En etant dans le dossier Exemple1, copier les codes pythons necessaires
cp ../code/Zonage.py .
cp ../code/gen_Entree_gen_manning/gen_Entree_gen_manning.py .

# Creer le fichier Manning.txt contenant dans l'ordre (telque defini dans le fichier Excel de zonage) les valeurs de
# Manning respectives pour les zones extraites
nano Manning.txt
0.022,0.026,0.024 #n1, n2, n3 respectivement

# Une fois l'environnement virtuel active, ajouter les valeurs de Manning au fichier Excel de zonage
python3 Zonage.py Zones_mesh6072.xlsx

# Creer le fichier Entree_gen_manning.txt
python3 gen_Entree_gen_manning.py Output_zonage.xlsx

# Depuis le dossier Exemple1
cp ../../src/gen_manning/gen_manning.py .

# Ajouter les valeurs de Manning aux mailles du fichier de maillage
python3 gen_manning.py mesh_6072.cgns
```

On peut vérifier que le maillage a bien été modifié en ouvrant le fichier *mannings.vtk* :

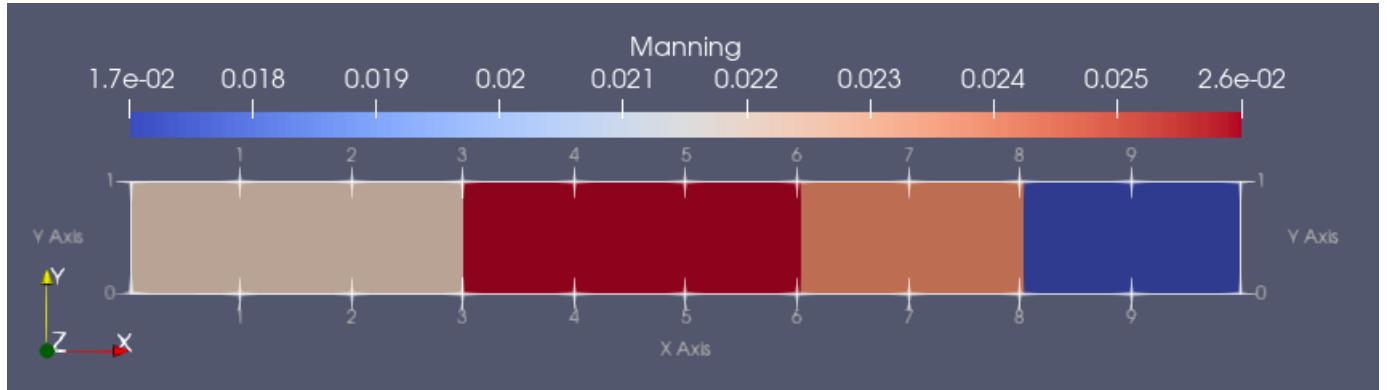


FIGURE 21 – Nombres de Manning modifiés sur le fichier mesh_6072.cgns

Nota :

- Les valeurs exportées dans le fichier Excel de zonage pour construire le nouveau maillage, sont les vtkOriginalCellIds extraits depuis Paraview
- Pour **m** zones de manning souhaitées, il est suffisant de sélectionner et exporter les données des cellules sur **m-1** zones ; la valeur du coefficient de Manning pour la dernière zone étant définie dans la variable **average_manning** du script gen_manning (average_manning = 0.017 dans notre exemple).
- Le fichier **Manning.txt** est indispensable au fonctionnement du script de zonage **Zonage.py**.

2.5 Découpe du maillage en sous domaines : parmetis_ghost

Pour pouvoir utiliser *CuteFlow* en parallèle, il est nécessaire de décomposer les fichiers de maillages en autant de sous-domaines que l'on veut utiliser de GPUs pour faire la résolution. De plus, il est nécessaire d'ajouter une ou plusieurs couches de mailles fantômes à chaque sous-domaine pour assurer la qualité de la solution à travers les sous-domaines. Pour cela, un code de prétraitement parallèle permettant de décomposer un maillage au format CGNS en plusieurs sous-domaines et d'y ajouter de multiples couches de mailles fantômes a été développé [4]. Il suffit de donner à ce programme un fichier au format CGNS, le nombre de sous-domaines voulus, et, le nombre de couches de mailles fantômes souhaité (une couche pour une méthode d'ordre 1 et deux couches pour une méthode d'ordre 2). En sortie, un unique fichier au format CGNS sera généré contenant toutes les zones et les informations nécessaires pour l'échange mémoire dans *CuteFlow*.

Le code **parmetis_ghost** est disponible sur GitHub <https://github.com/ETS-GRANIT/parmetis-ghostlayers>. On peut le cloner dans le scratch de Graham de la façon suivante :

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

On peut ensuite créer un dossier build et y compiler le code après avoir chargé les bons modules de la façon suivante,

```
cd parmetis-ghostlayers
mkdir build
cd build

#Chargement des modules nécessaires à la compilation
module load StdEnv/2020 gcc/9.3.0 metis/5.1.0 parmetis/4.0.3 cgns/4.1.2

#Création du makefile via cmake
cmake ..

#Compilation du code
make
```

Si on veut compiler le code *parmetis-ghostlayers* sur un autre système, il faudra s'assurer d'avoir installé les bibliothèques *metis*, *parmetis* et *cgns* avec en particulier le support d'écriture de fichier en parallèle pour la bibliothèque *cgns* ([PCGNS CGNS-Github](#)).

À la fin de la compilation, un fichier exécutable *main* est créé. Pour lancer ce code, il faut obligatoirement utiliser MPI avec deux processeurs (pré-requis pour utiliser *parmetis*) de la façon suivante,

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

où, NP est le nombre de processeurs utilisés (au moins 2), FICHIER_MAILLAGE.CGNS est un fichier de maillage au format CGNS contenant une zone à décomposer en plusieurs sous-domaines, NSD est le nombre de sous-domaines voulus, MODE prend soit la valeur 0 ou 1 selon qu'une maille voisine est calculée avec un côté en commun (MODE=0 choix classique) ou un sommet en commun (MODE=1), et, NC est le nombre de couches de mailles fantômes voulues (1 pour une méthode d'ordre 1 et 2 pour une méthode d'ordre 2). Un exemple d'utilisation est donc,

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

qui décomposera le fichier mille.cgns en 8 sous-domaines avec 2 couches de mailles fantômes calculés via les côtés en communs en utilisant 2 processeurs. Il est à noter que NP doit être inférieur ou égal à NSD.

On présente maintenant un exemple de décomposition d'un fichier de maillage qui sera utilisé en section 3.3. On va effectuer cette décomposition dans le dossier **CuteFlow/examples/example1** en utilisant le fichier exécutable *main* créé précédemment.

```
Depuis le dossier de Cuteflow
```

```
cd examples/example1
```

```
#copie de l'executable de parmetis-ghost (le code de decomposition)
cp ~/scratch/parmetis-ghostlayers/build/main .
```

```
#copie du fichier de maillage a decomposer
cp ../../meshes/mille/Mille_Iles_mesh_743968_elts.cgns .
```

```
#Si le fichier de maillage est volumineux il faudra demander un noeud de calcul
salloc --time=0:20:0 --ntasks=4 --account=def-soulaima --mem-per-cpu=4000M
```

```
#Une fois le noeud alloue, lancement de la decomposition de domaine
# 4 process MPI, 4 sous domaines, 2 couches de mailles fantomes
mpirun -n 4 ./main 4 Mille_Iles_mesh_743968_elts.cgns 0 2
```

```
#On peut ensuite renommer le fichier de sortie par exemple avec
mv Mesh_Output_pcgns_ch.cgns Mille_Iles_mesh_743968_elts_4_2.cgns
```

Suite à cette procédure, le fichier *Mille_Iles_mesh_743968_elts_4_2.cgns* contient la décomposition en 4 sous-domaines avec 2 couches de mailles fantôme du fichier de maillage original et peut être utilisé avec le code *CuteFlow* avec 4 GPUs (voir section 3.3).

3 CuteFlow

3.1 Fichiers sources et compilation

Les fichiers sources de *CuteFlow* sont dans le dossier [CuteFlow/src/cuteflow/](#). Un fichier *makefile* est présent dans le dossier [CuteFlow/](#) et peut être copié dans le dossier [CuteFlow/build/](#) pour la compilation.

Note : Pour utiliser la version multi-CPU il faut faire "git checkout multi-cpu-new" pour se mettre sur la branche multi-CPU du code, la compilation se fait ensuite de manière identique.

```
#Pour compiler CuteFlow
cd build/
cp ../makefile .
make
```

Le fichier exécutable *cuteflow* sera alors créé dans le dossier [CuteFlow/bin/](#).

L'utilisateur peut modifier les options de compilations dans le fichier *makefile* qu'il vient de copier dans le dossier [Cuteflow/build/](#).

Pour la compilation du code *cuteflow* il faut spécifier la *Compute Capability* cible des GPU pour lesquels on compile. La *Compute Capability* par défaut dans le *makefile* est 6.0 ce qui fonctionne pour la compilation sur CEDAR et GRAHAM avec les GPUs NVIDIA P100. Pour BELUGA et l'utilisation des GPUs NVIDIA V100 en général, il faut spécifier une *Compute Capability* de 7.0. C'est possible en faisant,

```
make CC=70
```

Si on veut être sûr que le code est bien recompilé, on peut forcer la recompilation de tous les fichiers avec l'option "-B" de la façon suivante :

```
#Pour des GPUs NVIDIA P100  
make -B CC=70
```

Note : Lorsque l'on demande des ressources de calcul, si on ne spécifie pas quel type de GPU utiliser, il est possible que le choix se fasse automatiquement et que des GPUs qui ne correspondent pas à la Compute Capability choisie soient utilisés. Il est donc fortement recommandé de spécifier exactement quel type de GPU utiliser lors du lancement d'une simulation avec `-gres=gpu[/:type] :number]` (voir https://docs.alliancecan.ca/wiki/Using_GPUs_with_Slurm).

Le fichier `makefile` chargera les modules nécessaires à la compilation du code sur les clusters de calculs de *Compute Canada* avec la commande :

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

Pour pouvoir compiler le code sur un autre système, il faudra installer et compiler les différentes versions des bibliothèques *nvhpc*, *cuda*, *openmpi* et *cgns*. En particulier, il faudra s'assurer que la bibliothèque *openmpi* soit compilée avec le support pour CUDA ([CUDA-Aware OpenMPI](#)) et que la bibliothèque CGNS, soit compilée pour la gestion de fichiers des maillages en parallèle ([PCGNS-doc CGNS-Github](#)).

Liste des bibliothèques utilisés pour la compilation du code *CuteFlow* en version multi-GPU (branche "master") :

- [nvhpc/20.7](#)
- [cuda/11.0](#)
- [openmpi/4](#)
- [cgns/4.1.2](#)

Liste des bibliothèques utilisés pour la compilation du code *CuteFlow* en version multi-CPU (branche "multi-cpu-new") :

- [gcc/9.3.0](#)
- [openmpi/4.0.3](#)
- [cgns/4.1.2](#)

3.2 Description du fichier donnees.f

Lors d'une simulation, le code *CuteFlow* lit les paramètres de la simulation dans le fichier **donnees.f** présent dans le dossier courant. On décrit dans cette section chaque paramètre de ce fichier qui est donné en aperçu en Figure 22. Un exemple peut être trouvé dans le dossier [CuteFlow/src/cuteflow/donnees.f](#).

1. GP (real) : La constante gravitationnelle (9.81 m.s^{-2})
2. is_override_manning (0/1) : Utilisé pour forcer un nombre de Manning (coefficient de frottement) spécifique plutôt que de lire celui des fichiers de maillages.
3. override_manning (real) : Nombre de Manning qui s'applique si is_override_manning est à 1.
4. is_cgns (0/1) : Prend la valeur 1 si le maillage utilisé est au format CGNS, et la valeur 0 si on utilise les anciens formats de maillages.
5. meshfile_path (string) : Chemin d'accès au fichier de maillage. Si le fichier de maillage est dans le dossier parent, `meshfile_path="../"`.
6. meshfile (string) : Nom du fichier de maillage.
7. elt_bound (0/1) : Utilisé pour ne pas avoir à re-calculer la connectivité des éléments d'un fichier de maillage. Si une simulation précédente a déjà été lancée, et que les fichiers `*boundary*` ont été créés alors `elt_bound` peut prendre la valeur de 1 pour accélérer marginalement le lancement des simulations. Si on utilise un maillage décomposé en plusieurs sous-domaines, le gain de temps sera négligeable et la valeur 0 peut-être systématiquement adoptée sans problème.
8. nombre_entrees (integer) : Nombre d'entrées du maillage.
9. nombre_sorties (integer) : Nombre de sorties du maillage.
10. eqlin_barrage (0/1) : Utilisé pour spécifier si la condition initiale est un problème de Riemann autour d'une droite. Si la valeur est à 1 alors les coordonnées de deux points de la droite seront lus dans les paramètres suivants.
11. x1eqbar (real) : Position *x* du premier point de la droite.
12. y1eqbar (real) : Position *y* du premier point de la droite.

```

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

  ! Données du maillage
  is_cgns=1,
  meshfile_path='',
  ! meshfile= 'mille_700k_2.cgns',                         ! Fichier de maillage
  meshfile= 'manning_Mille_Iles_mesh_481930_elts.cgns'
  elt_bound=0,                                              ! 1 si le fichier boundary_table existe déjà
  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
  xleqbar=274946., x2eqbar=274752.,                      ! Abscisses des deux points du barrage
  yleqbar=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 à partir d'un fichier, fichier avec solution ax éléments
  solinit=0,                                              ! 1 pour initialiser à 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.,
  outlet='fixedheight',                                    ! Débit aux entrées, séparer les débits pas des ,
  outlet_name="Outflow nodes"                            ! Type de sortie : {fixedheight, transm}
  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,                                         ! Tolérance sec/mouillé
  friction=1,                                              ! 1 pour prendre en compte la friction
  fricimplic=1,                                            ! 0 -> explicite, 1 -> I-dt/2*I, 2 -> I-dt*B
  TS=1000, CFL=0.4,                                       ! Temps maximal de simulation, nombre CFL
  is_dt_constant=0, constant_dt=1e-4,                    ! Tolérance relative entre débit entrée et débit sortie
  tol_reg_perm=1.0E-14,                                     ! Fréquence de print dans outfile.[0-9]
  freqaffich=10000,                                         ! Sauvegarde fichiers T3S à la fin

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

```

FIGURE 22 – Aperçu du fichier **donnees.f** édité avec nano

13. x2eqbar (real) : Position x du second point de la droite.
14. y2eqbar (real) : Position y du second point de la droite.
15. H_AMONT (real) : Hauteur de la **surface libre** en amont de la droite.
16. H_AVAL (real) : Hauteur de la **surface libre** en aval de la droite.
17. U_AMONT (real) : Vitesse selon x en amont de la droite.
18. U_AVAL (real) : Vitesse selon x en aval de la droite.
19. V_AMONT (real) : Vitesse selon y en amont de la droite.
20. V_AVAL (real) : Vitesse selon y en aval de la droite.
21. plan (0/1) : Utilisé pour spécifier l'initialisation de la solution selon un plan. Les coordonnées d'un point appartenant au plan et d'un vecteur orthogonal au plan sont lues dans la suite. Si $x_{plan} = 0.$, $y_{plan} = 0.$, $z_{plan} = 1.$, $x_{point} = 0.$, $y_{point} = 0.$, et, $z_{point} = 10.$, on a un plan orthogonal à l'axe z qui passe par le point $(0, 0, 10)$.
22. xplan (real) : Projection selon x d'un vecteur orthogonal au plan
23. yplan (real) : Projection selon y d'un vecteur orthogonal au plan
24. zplan (real) : Projection selon z d'un vecteur orthogonal au plan
25. xpoint (real) : Coordonnée x d'un point du plan.
26. ypoint (real) : Coordonnée y d'un point du plan.
27. zpoint (real) : Coordonnée z d'un point du plan.
28. solimit (0/1) : Utilisé pour spécifier si on initialise à partir d'un fichier solution déjà généré.
29. fich_sol_init (string) : Nom du fichier solution préexistant.
30. inlet (inflow/transm) : Spécifie le type de condition aux limites d'entrée. *inflow* pour un débit entrant, *transm* pour une condition transmissive.
31. inlet_name (string) : Nom des conditions aux limites d'entrées séparées par des virgules. Utilisé avec les fichiers CGNS lorsque plusieurs entrées sont présentes pour que chaque process soit au courant de toutes les entrées qui existent. (Dans les fichiers de maillages de l'archipel de Montréal, le nom des entrées est simplement *InflownodesX* où X est un nombre de 1 à 7, ce qui donne, *inlet_name = "Inflownodes0", "Inflownodes1", "Inflownodes2", "Inflownodes3", "Inflownodes4", "Inflownodes5", "Inflownodes6"*).
32. debiglob (real) : Débits des entrées du domaine séparés par des virgules.
33. outlet(fixedheight/transm) : Spécifie le type de condition aux limites de sortie. *fixedheight* pour une hauteur de sortie fixe, *transm* pour une condition transmissive.
34. outlet_name (string) : Nom des conditions aux limites de sorties séparées par des virgules.
35. H_sortie (real) : Hauteurs de la **surface libre** au niveau des sorties du domaine séparés par des virgules (non testé pour plusieurs sorties).
36. IFLUX (1/2/3/4) : Choix du schéma numérique utilisé. 1 pour HLLC classique, 2 pour HLLC façon riadh (meilleur pour la gestion sec/mouillé), 3 pour WAF façon Loukili, 4 MUSCL façon Toro. Pour les choix 3 et 4, 2 couches de mailles fantômes sont nécessaires lors de la décomposition en sous-domaines.
37. ilimiteur (1/2/3/4) : Choix du limiteur dans les méthodes WAF et MUSCL. 1 Superbee style, 2 Leer Style, 3 Minmod/Albada Style. Le limiteur 3 est souvent celui qui est le plus stable, le limiteur 1 est celui qui donne les meilleurs résultats sur des cas tests simples mais (comme le limiteur 2) il peut créer des vitesses sur des bathymétries complexes (reconstruction hydrostatique d'ordre 2 nécessaire).
38. iupwind (1/2) : Façon de rechercher les mailles upwind pour l'ordre 2. Doit prendre la valeur 1 pour avoir des résultats corrects, la valeur 2 est un test en cours qui ne donne pas de résultats stables.
39. tolisec (real) : Valeur de la tolérance sec/mouillé. Une valeur entre $10e - 4$ et $10e - 8$ est habituelle. Une valeur trop grande ou trop petite peut être la cause de divergences du code.
40. friction (0/1) : Active ou non la gestion des frottements dans le code (via le Manning).
41. fircimplic (0/1) : Active la semi-implicitation des termes de friction. À laisser à 1.
42. TS (real) : Temps final de la simulation.
43. CFL (real) : Nombre CFL entre 0 et 1. Plus la valeur est proche de 1 plus des divergences peuvent apparaître. (La distance caractéristique est 1.8 fois la valeur du rayon du cercle inscrit dans la plus petite maille du maillage).
44. is_dt_constant (0/1) : Utilisé pour mettre une valeur fixe du pas de temps (Non conseillé).
45. constant_dt (real) : Si *is_dt_constant* est à 1, spécifie la valeur du pas de temps pour la simulation.
46. tol_reg_perm (real) : Tolérance de sortie liée au régime permanent. Si l'écart relatif entre le débit de sortie et le débit d'entrée est plus petit que la tolérance pendant un certain temps, alors la simulation stoppera. Une valeur de $1e - 15$ permet à cette condition de sortie de ne jamais être atteinte.
47. freqaffich (integer) : Nombre d'itérations entre deux affichages de l'état de la simulation dans le fichier de sortie.
48. sol_z_offset (real) : Anciennement utilisé pour rehausser artificiellement les solutions de sorties du code. Une valeur de 0 permet de n'avoir aucun impact.
49. solrestart (0/1) : Si une valeur de 1 est adoptée, le code générera une solution finale à partir de laquelle il sera possible de redémarrer.

50. solvisu (0/1/2/3/4/5) : Spécifie le type de fichier de sortie. 0 pour aucune sortie, 1 pour toutes les sorties, 2 pour une sortie au format vtk, 3 pour une sortie au format cgns, 4 pour une sortie de solution sur chaque noeud du maillage, 5 pour une solution sur chaque élément du maillage (à partir de laquelle on peut redémarrer).
51. sortie_finale_bluekenue (0/1) : Sortie au format T3S lisible par Bluekenue.

3.3 Lancement d'une simulation

Dans cette section, on présente comment lancer une simulation depuis le dossier **examples/example2**. Cette simulation utilisera 4 GPUs et s'effectuera sur les fichiers du maillage que l'utilisateur a déjà décomposé dans le dossier **examples/example1** (voir section [2.5](#)).

```
#Depuis le dossier Cuteflow
cd examples/example2

# Copie du fichier de maillage dans le dossier courant
cp ../example1/Mille_Iles_mesh_48930_elts_4_2.txt .

# Copie du fichier de donnees dans le dossier courant
cp ../../src/cuteflow/donnees.f .

# Copie de l'executable de CUTEFLOW dans le dossier courant
cp ../../bin/cuteflow .

# Copie du fichier de lancement du code dans le dossier courant
cp ../../scripts/4_gpu.sh .
```

À ce stade, le résultat de la commande *ls* dans le dossier **example2** devrait être le suivant :

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

L'utilisateur doit ensuite modifier les paramètres de simulation dans le fichier **donnees.f** en se reportant à la section [3.2](#).

Il y a ensuite deux façons de lancer une simulation : en lançant un job différé ou en demandant un noeud interactif. On peut demander un noeud interactif de la façon suivante,

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

On demande avec cette commande deux nœuds de calculs avec 2 CPUs et 2 GPUs (P100) pour 1 heure. Ici, on spécifie comme groupe *rrg-soulaima-ac* uniquement lorsque l'on est sur GRAHAM pour utiliser l'attribution de ressources qui a été donnée au groupe de recherche. On peut aussi utiliser le groupe *def-soulaima* mais on aura une priorité et une quantité de ressources allouées moins importante.

Il se peut qu'il faille attendre un petit moment pour que le noeud de calcul nous soit alloué. Une fois que le noeud de calcul nous a été alloué, on peut lancer la simulation avec les commandes suivantes,

```
#Lancement de la simulation
mpirun --mca pml ob1 -n 4 ./cuteflow
```

De cette façon, tous les process MPI vont faire une sortie directement dans le terminal, ce n'est souvent pas ce qu'on veut, on peut alors remplacer la dernière commande par :

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

De cette façon, aucun process ne fera de sortie dans la console, ils feront chacun leurs sorties dans les fichiers **outfile.X** avec X leur numéro de process. On peut monitorer l'avancement de la simulation dans un de ces fichiers avec la commande :

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

```

1 Device name:Tesla P100-PCIE-12GBthread id : 0
2 Compute capability : 6.0
3          0 lecture du maillage en cours
4 id      0 ,      122191 nodes
5 id      0 ,      241151 elens
6 id      0 ,      210 noeuds d'entree
7 id      0 ,      0 noeuds de sortie
8 id      0 ,      2741 noeuds de murs
9 id      0 ,      185 mailles fantomes a recep
10 id     0 ,      185 mailles fantomes a envoyer
11 id     0 ,      1 bloc fantomes a receptionner
12      240967    185      1
13 id     0 ,      1 bloc fantomes a envoyer
14      240782    185      1
15          0 lecture du maillage reussie
16 gridx, gridy (for set_boundary) =      15072      15072
17 =====
18 Parameters
19 =====
20 meshfile      =
21 0_Mille_Iles_mesh_481930_elts.txt
22 cotemin      = 0.1529265774476539
23 elt_bound     = 0
24 H_AMONT      = 30.00000000000000
25 U_AMONT      = 0.00000000000000
26 V_AMONT      = 0.00000000000000
27 H_AVAL       = 29.00000000000000
28 U_AVAL       = 0.00000000000000
29 V_AVAL       = 0.00000000000000
30 iFlux        = 2
31 FricImplic   = 1
32 jauge_s snapshots = 100
33 nbrjauge     = 0
34 nbrcoupes    = 0
35 solrestart   = 0
36 restart_s snapshots = 10
37 solvtk       = 1
38 vtk_s snapshots = 10
39 tol_reg_perm = 1.000000000000001E-015
40 tol          = 9.999999747524271E-007
41 tolisec      = 1.000000000000000E-008
42 tolaffiche   = 0.100000014901161
43 freqaffich   = 1000
44 dry_as_wall  = 0
45 local time step = 0
46 nombre_entree = 1
47 debitglob    1 = 800.000000000000
48 longueur_entree 1 = 1672.082325559227
49 nombre_sortie = 1
50 H_SORTIE     1 = 29.00000000000000
51 =====
52 =====
53 ***** FULL_FV *****
54
55 FULL-ORDER : VOLUMES FINIS
56
57 Loop over time starts
58 -----
59 nt =      1000
60 prochain tsolvtk = 90.00000000000000      3
61 tc = 70.39981172647879      Secondes => 1.173330195441313
62 Minutes
63
64 debit_entree 1 = 799.999999999999
65 debit_sortie = 0.00000000000000 M3/S
66 -----
67 nt (end of loop on time) = 1295
68 Time taken by time loop (parallel) = 3412.849 ms
69 =====
70 ====== FIN DE LA SIMULATION =====
71 =====
72
73 DUREE DU CALCUL : 3.420150041580200      Secondes
74 =====
75 =====
NORMAL | outfile.0

```

FIGURE 23 – Exemple 2 : fichier **outfile.0**

qui permettra de voir tous les ajouts qui sont faits dans les fichiers de manières interactive.

Si on demande un nœud interactif, on ne devrait rien voir dans le terminal une fois la commande

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

lancée, si des erreurs sont présentes, c'est dans les fichiers *outfile.X* qu'on les verra.

Une fois la simulation terminée, on peut regarder avec un éditeur de texte le fichier **outfile.0** par exemple pour s'assurer que tout s'est bien passé. Dans ce cas, avec le fichier de données montré plus haut, on obtient pour le fichier **outfile.0** de la figure 23.

Alternativement, on peut soumettre le code à l'ordonnanceur, après avoir modifié le fichier **4_gpu.sh** pour correspondre à ses besoins, avec la commande suivante :

```
# En etant dans le dossier examples/example2/
sbatch 4_gpu.sh
```

On peut regarder si le code est lancé ou à quel moment il le sera avec la commande

```
squeue -u $USER
```

De la même façon que précédemment, on peut suivre l'avancement en monitorant les fichiers `outfile.[0-9]` qui servent de sortie standard pour chaque process MPI.

Une fois la simulation terminée, selon les paramètres du fichier `donnees.f`, plusieurs fichiers résultats seront générés et pourront être visualisés à l'aide de Paraview (voir section 4).

3.3.1 Lancement d'une simulation à partir d'une solution déjà générée

Une simulation peut être redémarrée à partir d'une solution enregistrée sur les éléments. Il suffit de mettre le paramètre `solrestart` à 1 dans le fichier `donnees.f` pour déclencher l'enregistrement de la solution dans les fichiers appropriés en fin de simulation. Un fichier solution `*_solution_elements_restart.txt` sera enregistré pour chaque sous-domaine et sera utilisé pour faire repartir la simulation.

Une fois les fichiers `*_solution_elements_restart.txt` générés, si on veut faire repartir la simulation sur ces fichiers il faudra les renommer puis mettre leur nom de base en paramètre dans `fich_sol_init` dans le fichier `donnees.f`.

Dans le cas de l'exemple 2, on peut par exemple mettre `solrestart = 1` dans le fichier `donnees.f`. Une fois une première simulation finie, les fichiers `*_solution_elements_restart.txt` seront générés. Si on veut pouvoir réenregistrer la solution sur les éléments une nouvelle fois, il faut renommer les fichiers générés, en faisant par exemple

```
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
```

Il faudra alors ensuite mettre le paramètre `solinit = 1` et `fich_sol_init = 'sol_example2.txt'` pour initialiser la solution avec ces fichiers.

3.4 Lancement de plusieurs simulations en simultané

Cette section présente comment on peut lancer plusieurs simulations en simultané en utilisant des `job_arrays`. L'objectif est ici d'avoir un script bash qui va créer un dossier pour chaque simulation qu'on veut lancer en y copiant le fichier de paramètre correspondant. On pourra ensuite utiliser un `job_arrays` pour lancer toutes les simulations simultanément.

Dans la suite de la section, on prend comme exemple le lancement de plusieurs simulations dans le dossier `examples/example3` en utilisant les fichiers de maillage qui ont été découpés en section 2.5.

3.4.1 Génération des fichiers de données

On a besoin d'un moyen de générer les fichiers de données automatiquement à partir des valeurs de `debitglob`, `hamont`, `haval` et `Manning`. C'est le but du script `gen_donnees` qui est utilisé à l'intérieur du script `multi`. On n'a pas normalement à l'utiliser tout seul, mais on décrit quand même son utilisation simple.

Pour générer un fichier de donnée on fait,

```
#Pour debit_global=1000 hamont=31 haval=30 manning=0.04  
./gen_donnees 1000 31 30 0.04
```

Le fichier de données `donnees.f` sera alors créé et il contiendra bien les paramètres passés en argument. Attention, dans la configuration actuelle, la valeur de `haval` sera aussi utilisée comme `hsortie`. Ce script peut être trouvé dans le dossier `scripts` et l'utilisateur sera sûrement amené à le modifier pour l'utilisation qu'il désire en faire. Ce script sera appelé par le script `multi.sh` avec en argument chacune des lignes du fichier d'input, on explique cela dans la suite.

3.4.2 Préparation du dossier `base_files`

Avant de lancer le script `multi.sh` il faut préparer les fichiers dont le code aura besoin. Pour cela, il faut créer un dossier `base_files` dans lequel on va mettre ces fichiers. Les fichiers nécessaires sont : le fichier de maillage, l'exécutable `cuteflow`. On procédera alors de la façon suivante,

```

#Depuis le dossier CUTEFLOW_CUDA_MPI
cd examples/example3

#Copie des scripts dans le dossier courant
cp ../../scripts/multi.sh .
cp ../../scripts/gen_donnees.sh .

#Creation du dossier base_files et copie des fichiers de maillages
mkdir base_files
cp ./example1/Mille_Iles_mesh_743968_elts_4_2.cgns base_files/
cp ../../bin/cuteflow base_files/

```

Suite à ces opérations, le résultat de la commande *tree* lancée dans le dossier **examples/example3** devrait être conforme à la figure 24. (On note que le dossier **example3** devait déjà contenir le fichier *INPUT_MONTE_CARLO.dat*)

```

delmasv@gra283:example3$ tree
.
└── base_files
    └── cuteflow
        └── Mille_Iles_mesh_743968_elts_4_2.cgns
    └── gen_donnees.sh
    └── INPUT_MONTE_CARLO.dat
    └── multi.sh

1 directory, 5 files

```

FIGURE 24 – Sortie de la commande *tree*

3.4.3 Lancement du script *multi.sh*

Le script *multi.sh* lit un fichier d’input et crée pour chaque ligne (pour chaque cas de simulation) un dossier *multi_*[0-9]*. Il copie dans ce dossier les fichiers du dossier *base_files* et génère les fichiers de donnée grâce au script *gen_donnees*. Le fichier *INPUT_MONTE_CARLO.dat* nous sert de fichier d’input pour lancer plusieurs simulations dans le dossier **examples/example3**.

1	5			
2	800.00	29.000000	29.000000	0.020000
3	1000.00	30.000000	29.000000	0.020000
4	1200.00	31.000000	29.000000	0.020000
5	1400.00	32.000000	29.000000	0.020000
6	1600.00	33.000000	29.000000	0.020000

FIGURE 25 – Fichier *INPUT_MONTE_CARLO.dat*

On voit dans ce fichier que 5 simulations seront lancées, pour comprendre ce à quoi chaque colonne correspond, il faut aller voir le script *gen_donnees.sh* qui génère à partir de chaque ligne le fichier *donnees.f* correspondant. Dans cet exemple, la première colonne correspond au débit d’entrée, la seconde à la hauteur amont, la troisième à la hauteur aval et à la hauteur de sortie, et la dernière correspond au nombre de Manning global dans le domaine.

Dans le dossier **examples/example3**, on peut lancer ce script avec la commande

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

```
delmasv@cedar1:example3$ ./multi.sh INPUT_MONTE_CARLO.dat
Cas 1 800.00 29.000000 29.000000 0.020000
Cas 2 1000.00 30.000000 29.000000 0.020000
Cas 3 1200.00 31.000000 29.000000 0.020000
Cas 4 1400.00 32.000000 29.000000 0.020000
Cas 5 1600.00 33.000000 29.000000 0.020000
```

FIGURE 26 – Lancement du script *multi.sh*

Les différents dossiers *multi_*[0-9]* seront créés les uns à la suite des autres. On peut ensuite vérifier les paramètres de simulation, par exemple dans *multi_1*. Si un paramètre doit être changé, par exemple le temps final de simulation, plutôt que de le changer à la main dans chacun des dossiers de simulation, il suffit de le changer dans le script *gen_donnees.sh* puis de relancer le script *multi.sh* comme indiqué précédemment. De cette façon la modification s'appliquera dans tous les dossiers.

On peut vérifier la bonne structure du dossier **examples/example3** suite à l'exécution du script *multi.sh* en exécutant la commande *tree* dans ce dossier, la sortie devrait être identique à la figure 27.

```
delmasv@gra283:example3$ tree
.
├── base_files
│   └── cuteflow
│       └── Mille_Iles_mesh_743968_elt_4_2.cgns
├── donnees.f
└── gen_donnees.sh
    └── INPUT_MONTE_CARLO.dat
    └── multi_1
        ├── cuteflow
        │   └── donnees.f
    └── multi_2
        ├── cuteflow
        │   └── donnees.f
    └── multi_3
        ├── cuteflow
        │   └── donnees.f
    └── multi_4
        ├── cuteflow
        │   └── donnees.f
    └── multi_5
        ├── cuteflow
        │   └── donnees.f
    └── multi.sh
    └── donnees.f

6 directories, 16 files
```

FIGURE 27 – *tree* une fois le script *multi.sh* lancé.

3.4.4 Lancement du job array

On a créé dans la partie précédente la structure des dossiers pour lancer toutes les simulations spécifiées dans le fichier *INPUT_MONTE_CARLO.dat*, il faut maintenant lancer toutes ces simulations sur le cluster de calcul. Dans ce but, on utilise un job array. Un job array est un moyen rapide de lancer plusieurs jobs avec les mêmes paramètres, la description en est faite sur le wiki de Compute Canada https://docs.computecanada.ca/wiki/Job_arrays. Le fichier que nous allons utiliser dans ce cas est *array.sh* présent dans le dossier *script*.

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --gres=gpu:2
#SBATCH --mem=4000M
#SBATCH --time=00:10:00
#SBATCH --account=def-soulaime
#SBATCH --array=1-5

module load pgf/19.4 cuda/10.0.130 openmpi/3.1.2
cd multi
#SBATCH --array=1-5
mpirun --mca pml ob1 -o ./cuteflow > outfile.SOMPI_COMM_WORLD_RANK
```

FIGURE 28 – Fichier *array.sh*

Alternativement on peut utiliser *array_group.sh* ou le nom du groupe a été changé pour utiliser l'attribution de ressources qui a été donné au groupe de recherche.

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --gres=gpu:2
#SBATCH --mem=4000M
#SBATCH --time=00:10:00
#SBATCH --account=def-soulaime
#SBATCH --array=1-5

cd multi
#SBATCH --array=1-5
mpirun --mca pml ob1 -o ./cuteflow > outfile.SOMPI_COMM_WORLD_RANK
```

FIGURE 29 – Fichier *array_cedar.sh*

Ce fichier décrit un job qui utilise un nœud de calcul avec 4 GPU pour une durée de 10 minutes. Un paramètre important est le paramètre array qui décrit le nombre de jobs à lancer. Dans cet exemple, on lance 5 jobs qui seront numérotés de 1 à 5. On se sert de ce numéro qui sera stocké dans la variable d'environnement SLURM_ARRAY_TASK_ID pour se déplacer dans le dossier adéquat avant de lancer le code. Pour soumettre ce job array à l'ordonnanceur, il suffit ensuite de faire

```
sbatch array.sh
```

Pour lancer les 5 simulations du dossier **examples/example3** il faut donc faire,

```
#Depuis le dossier CUTEFLOW_CUDA_MPI
```

```
cd examples/example3
```

```
#Copie du script array.sh
```

```
cp ../../scripts/array.sh .
```

```
#Lancement du job array
```

```
sbatch array.sh
```

Les 5 simulations seront alors lancées sur le cluster de calcul. On peut savoir si les simulations sont bien lancées avec la commande *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 – Commande *squeue -u \$USER*

Dans le cas de la figure 30 les 5 simulations se sont toutes lancées et il reste 9 min 49 s de temps maximal pour leur exécution. Si dans la colonne **ST** (Status) il est indiqué **PD** (Pending) c'est que les simulations ne se sont pas encore lancées et il faut simplement attendre.

4 Traitement dans Paraview des fichiers *.vtk et *.cgns

Le logiciel Paraview [3] est installé par défaut sur les grappes de calcul de Compute Canada et permet une utilisation client/serveur ce qui permet de visualiser des solutions très lourdes sans avoir à télécharger les données sur son ordinateur personnel. Un tutoriel détaillé est présent sur le site officiel https://www.paraview.org/Wiki/The_ParaView_Tutorial et l'utilisation en mode client/serveur sur les grappes de calcul de Compute Canada est faite sur leur wiki officiel <https://docs.computecanada.ca/wiki/ParaView>. Si l'utilisateur préfère télécharger les fichiers solutions localement, il peut sauter la section 4.1.

4.1 Tunnel SSH pour la visualisation avec Paraview sur les clusters de calcul en utilisant MobaXterm

Pour faire de la visualisation à distance avec Paraview, il va falloir lancer un serveur Paraview sur un noeud de calcul du cluster et s'y connecter depuis son ordinateur. Pour s'y connecter la seule façon de procéder et d'utiliser un tunnel SSH pour forwarder le port 11111 local sur le port 11111 du noeud de calcul sur lequel on a lancé le serveur Paraview. De cette façon en faisant se connecter Paraview sur le port 11111 de sa machine locale, on se connectera en fait sur le noeud de calcul voulu.

Avant de faire un tunnel SSH, il faut demander un noeud de calcul sur le cluster et lancer un serveur Paraview dessus. Pour ce faire, deux scripts sont présents dans le dossier `scripts/`. Le premier `request_visu.sh` permet simplement de faire une commande qui demande un noeud de calcul avec un certain nombre de CPU et de mémoire, ici 16 CPU avec 12 Go de mémoire. Le script `load_para_cpu.sh` permet ensuite de lancer un serveur Paraview utilisant les 16 CPU demandés lors de l'étape précédente (on peut évidemment demander plus de CPU ou de mémoire si besoin).

```
#Depuis le dossier CUTEFLOW_CUDA_MPI
cd scripts/
./request_visu.sh 2
```

Une fois le noeud de calcul attribué, faire

```
#Depuis le dossier CUTEFLOW_CUDA_MPI/scripts/
./load_para_cpu.sh
```

Cela peut prendre un peu de temps mais on devrait avoir un affichage proche de la figure suivante,

```
delmasv@cedar1:~$ cd ~/scratch/test_doc/CUTEFLOW_CUDA_MPI/
delmasv@cedar1:CUTEFLOW_CUDA_MPI$ cd scripts/
delmasv@cedar1:scripts$ ./request_visu.sh 2
salloc: Granted job allocation 49547676
salloc: Waiting for resource configuration
salloc: Nodes cdr[767-768] are ready for job
delmasv@cdr767:scripts$ ./load_para_cpu.sh
Waiting for client...
Connection URL: cs://cdr767.int.cedar.computeCanada.ca:11111
Accepting connection(s): cdr767.int.cedar.computeCanada.ca:11111
```

FIGURE 31 – Lancement du serveur Paraview sur un noeud de calcul

Une fois arrivé au point de la figure 31 le serveur Paraview est lancé sur un noeud de calcul du cluster. L'important ici est de repérer quel est le noeud de calcul sur lequel le serveur est lancé. On voit dans l'image que le noeud de calcul s'appelle `cdr767`, il est indiqué dans la partie droite de la variable PS1 (à droite du @) et c'est aussi la première partie de l'URL de connections que nous n'allons pas utiliser dans la suite. Sur CEDAR les noeud s'appellent avec `cdr` en préfixe, sur BELUGA c'est `blg` et sur GRAHAM c'est `gra`. Une fois le noeud de calcul repéré, on peut passer à l'étape suivante qui consiste à créer le tunnel SSH entre notre ordinateur et ce noeud sur le cluster de calcul. Pour ce faire, on utilise MobaXterm. MobaXterm propose un visuel qui montre très bien ce qu'il se passe (voir Figure 32).

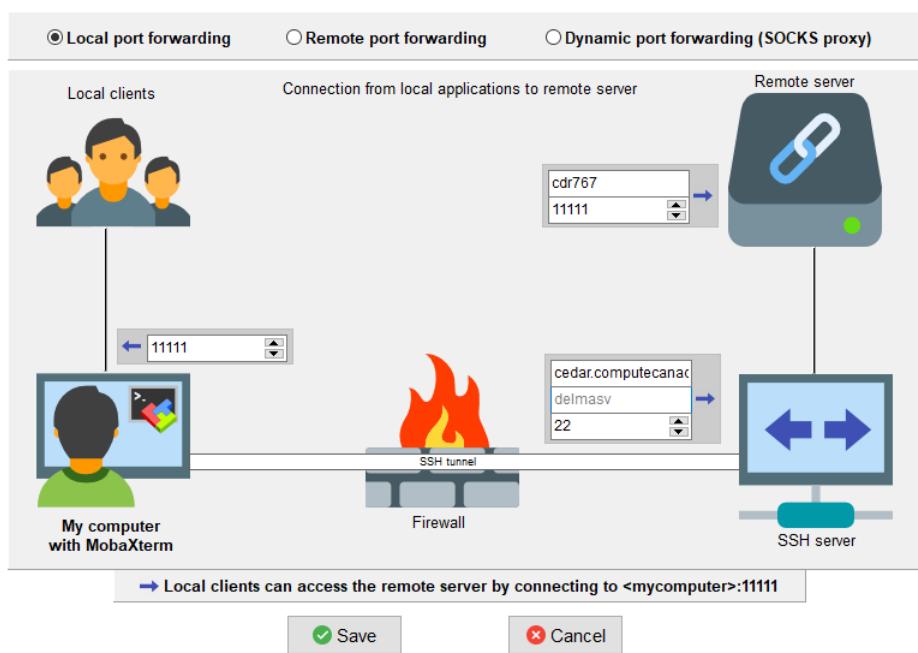


FIGURE 32 – MobaXterm tunnel SSH pour Paraview

Sur cette figure, on voit qu'on connecte le port 11111 de notre ordinateur via le serveur SSH *cedar.computecanada.ca* sur le port 11111 du noeud de calcul *cdr767*. Il faut paramétriser cette fenêtre comme indiqué sur la figure en changeant simplement le nom du noeud de calcul sur lequel le serveur Paraview a été lancé. Une fois ces paramètres rentrés, il suffit de lancer le tunnel en cliquant sur le signe lecture dans MobaXterm (voir figure suivante).

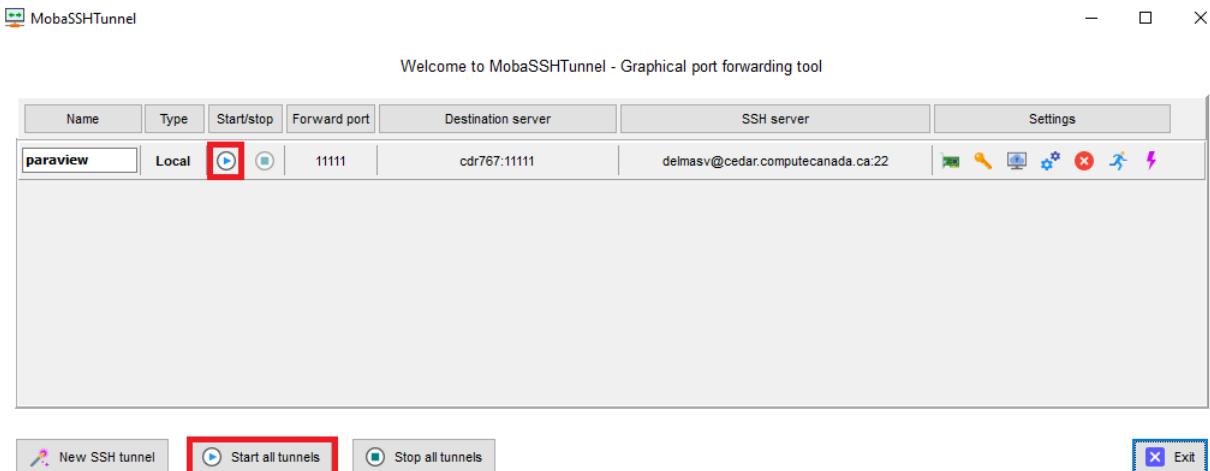


FIGURE 33 – Lancement du tunnel SSH dans MobaXterm

Une fois que le tunnel SSH a été lancé, il suffit de lancer Paraview sur son ordinateur local. Il faut cependant s'assurer d'utiliser la même version de Paraview que celle présente sur les clusters de calculs (loadé par le script *load_para_cpu.sh*) qui est la version 5.5.2 de Paraview téléchargeable gratuitement sur <https://www.paraview.org/download/>. Une fois Paraview lancé, si c'est la première fois il faut créer la connexion à un serveur de la manière suivante :



FIGURE 34 – Sélection du menu *connex*

Il faut ensuite naviguer dans les fenêtres en suivant ces étapes :

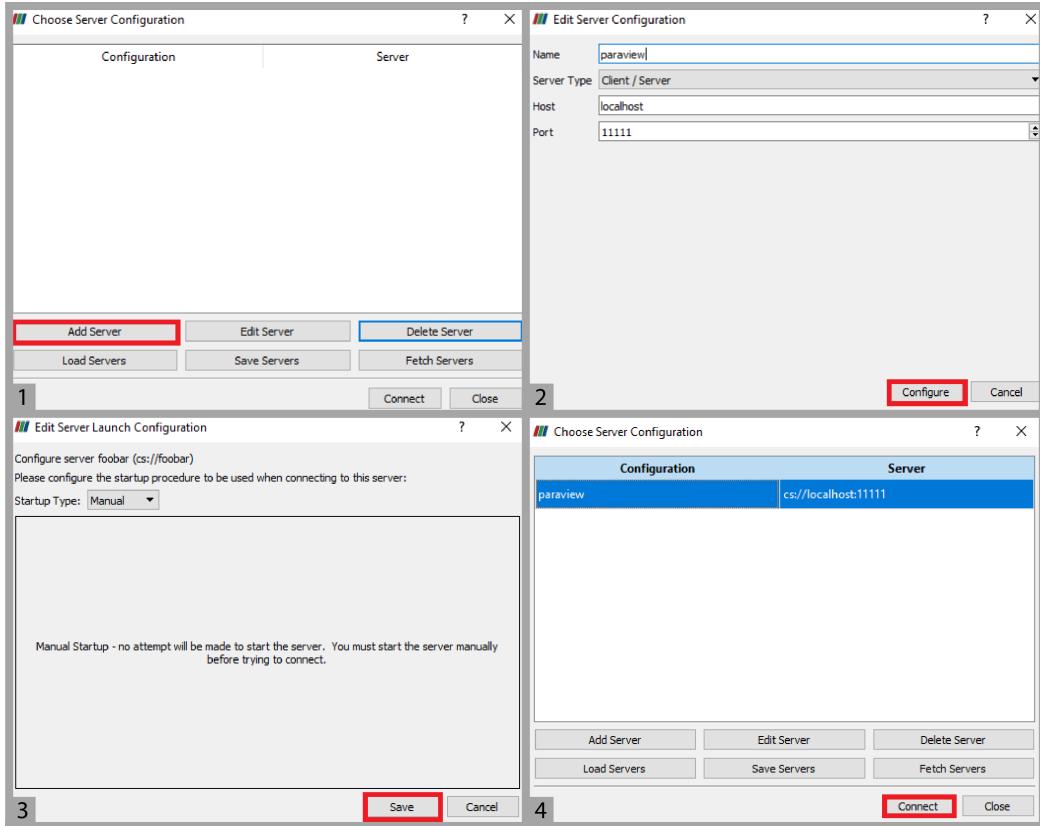


FIGURE 35 – Création de la connexion dans Paraview

À ce stade la configuration est finie, il suffit de cliquer sur connect pour se connecter à son port 11111 local qui est redirigé via le tunnel SSH sur le port 11111 du nœud de calcul sur le cluster sur lequel on a lancé le serveur Paraview un peu plus tôt.

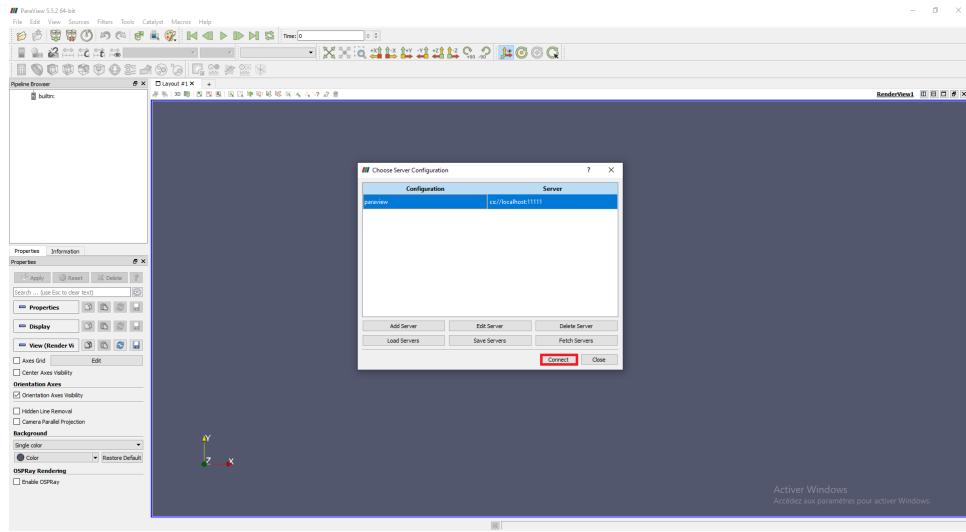
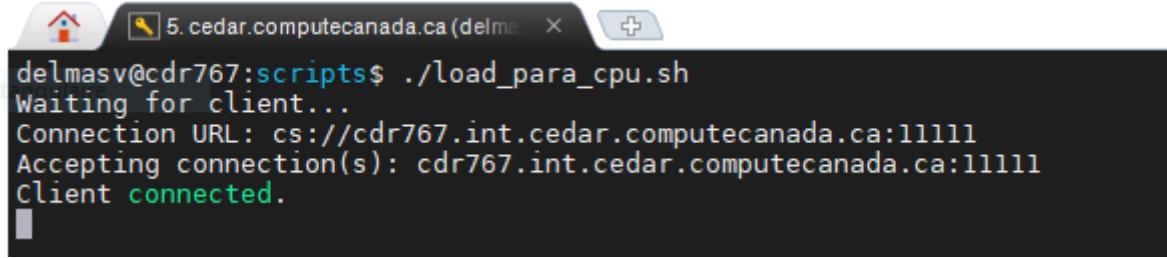


FIGURE 36 – Connexion au port 11111 de sa machine dans Paraview

Si tout se passe bien, il ne devrait pas y avoir d'erreurs une fois qu'on clique sur connect mais l'écran peut rester figer quelques instants. On devrait ensuite voir l'affichage changer dans la partie gauche de Paraview pour correspondre à l'image 38. On peut aussi vérifier que dans le terminal MobaXterm où on avait lancé le serveur Paraview, il devrait être indiqué *Client Connected*.



```

5.cedar.computeCanada.ca (delmasv) ~
delmasv@cdr767:scripts$ ./load_para_cpu.sh
Waiting for client...
Connection URL: cs://cdr767.int.cedar.computeCanada.ca:11111
Accepting connection(s): cdr767.int.cedar.computeCanada.ca:11111
Client connected.

```

FIGURE 37 – Connexion au serveur Paraview réussie

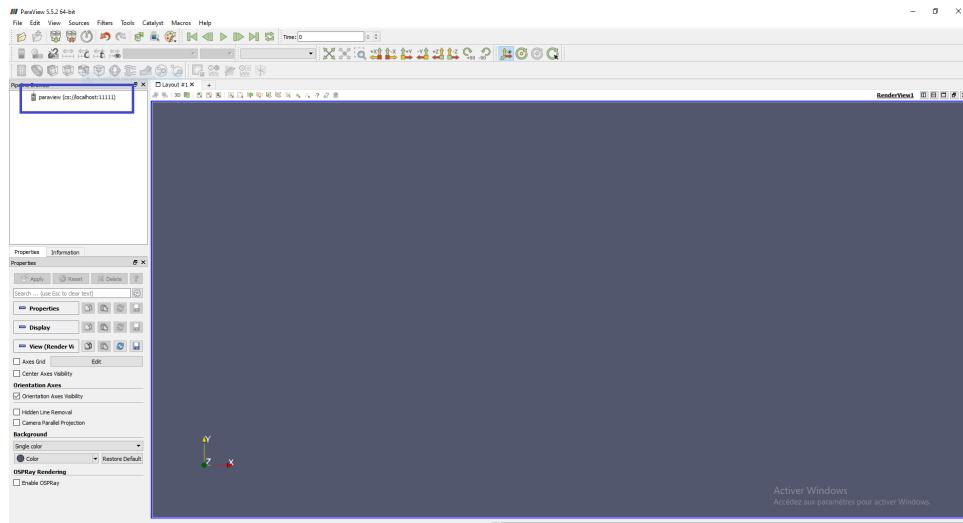


FIGURE 38 – Connexion réussie au serveur distant Paraview

Pour avoir un exemple concret, la section [4.2.2](#) détaille l'ouverture d'un fichier sur le serveur distant.

4.2 Visualisation des fichiers résultats générés par CUTEFLOW sur Paraview

On présente ici une utilisation basique pour visualiser les fichiers out_*.cgns produits par le code *CuteFlow*. On considère que les fichiers out_*.cgns ont déjà été téléchargés localement ou que l'utilisateur s'est déjà connecté au cluster de calcul via un tunnel ssh. Attention à bien lire les fichiers out_*.cgns qui sont les fichiers de sortie de *CuteFlow* et non pas les fichiers de maillages *.cgns qui eux ne contiennent pas la solution temporelle.

Les fichiers qui seront utilisés dans cette section se trouvent dans le dossier *Traitement_des_fichiers*.

Note : lors de l'utilisation de Paraview, plusieurs outils peuvent être sélectionnés en cherchant dans les différents menus. Pour plus de simplicité, il est possible d'utiliser le raccourci *Ctrl + espace* pour ouvrir une boîte de recherche dans laquelle tous les outils sont présents. C'est par exemple ce qui est fait sur la figure [41](#) pour rechercher l'outil *Merge Blocks*.

4.2.1 Visualisation du fichier rect_6138

On va commencer par comparer les différents ordres avec une solution exacte sur le fichier *out_mesh_6138*, un domaine rectangulaire de 10m de longs sur 1 de large.

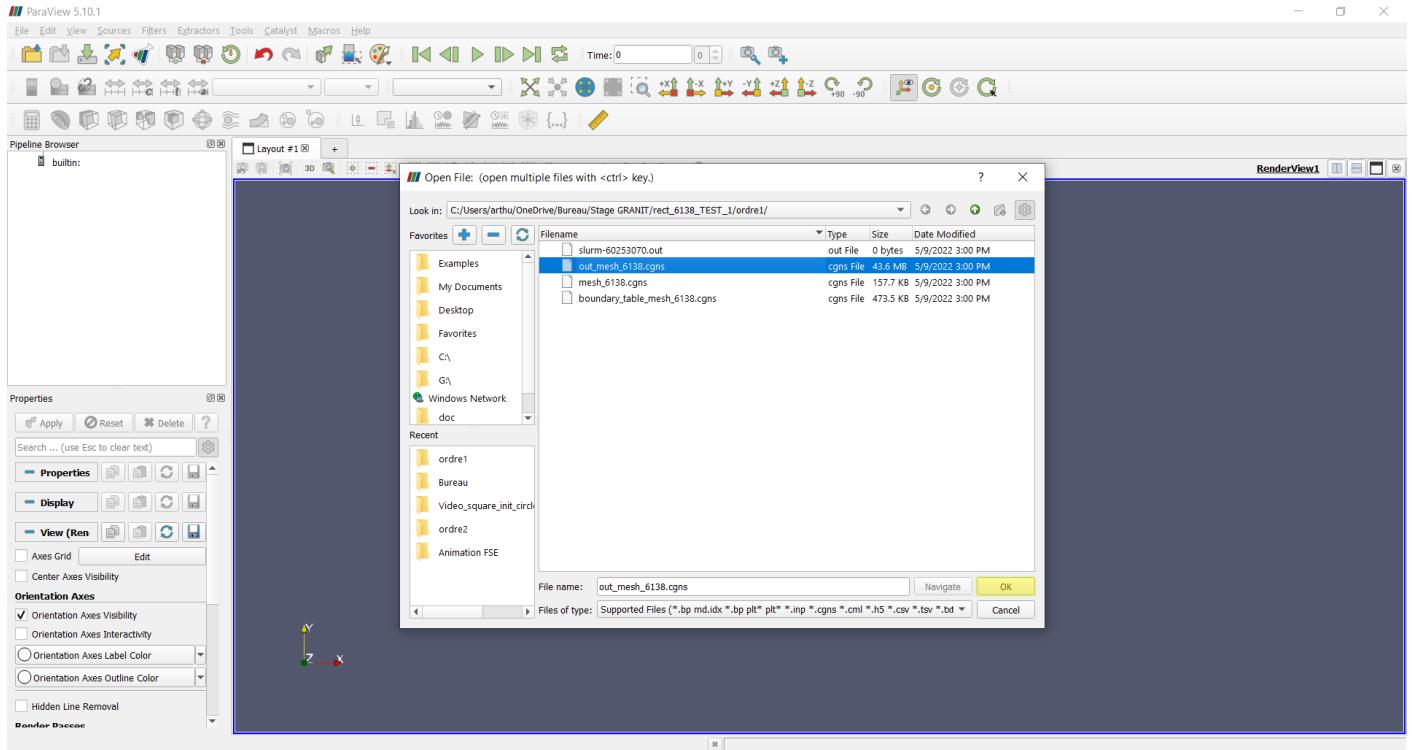


FIGURE 39 – Ouverture du fichier out_mesh_6138 d’ordre 1 avec *File* et *Open...*

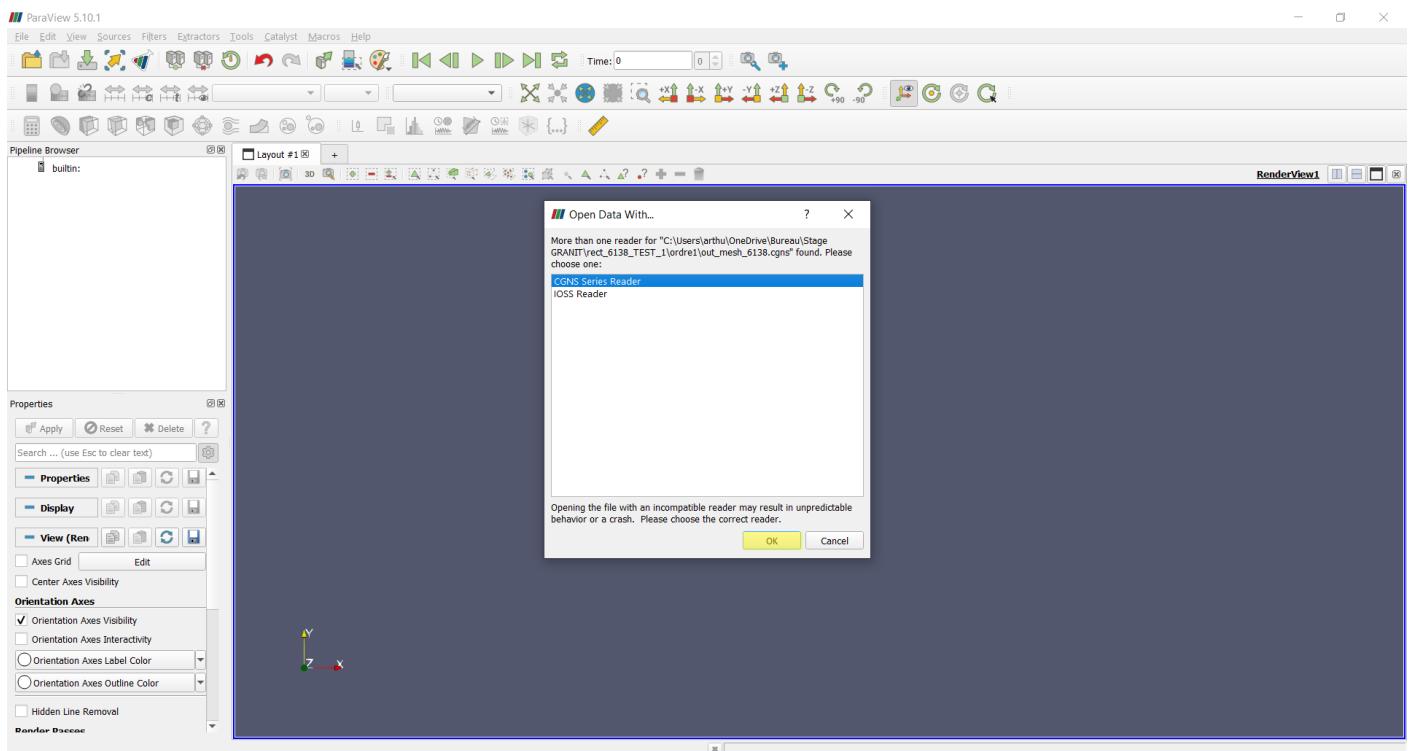


FIGURE 40 – Choix du lecteur

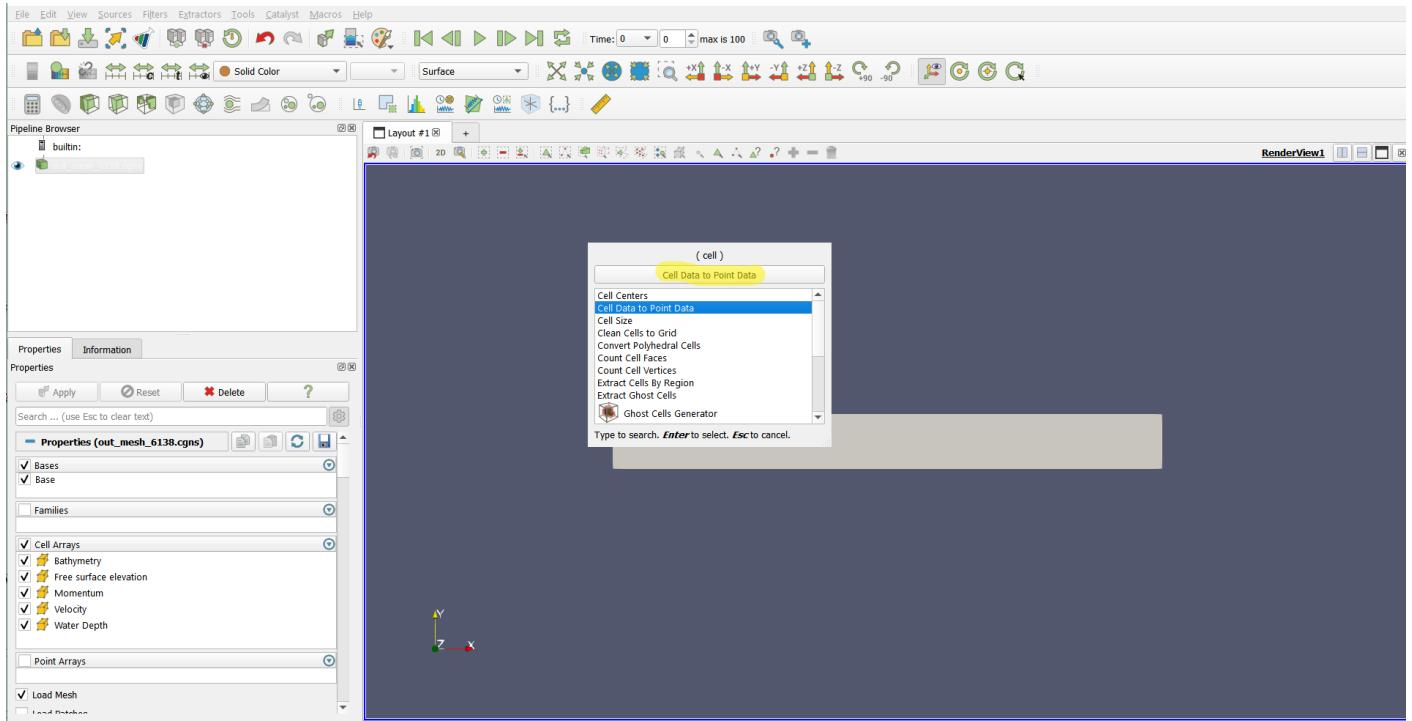


FIGURE 41 – Sélection de l’outil *Cell Data to Point Data* pour éviter le crénelage des solutions tracées

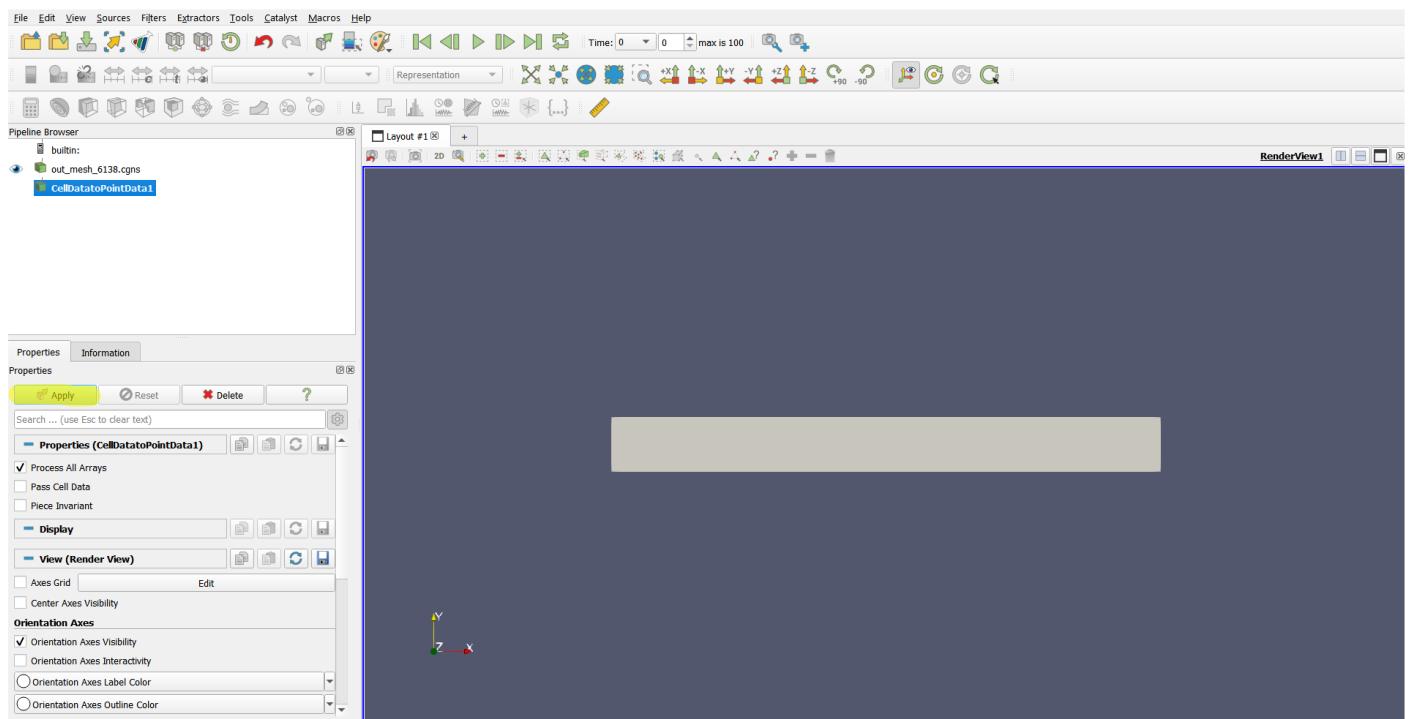


FIGURE 42 – *Apply* pour effectuer l’opération

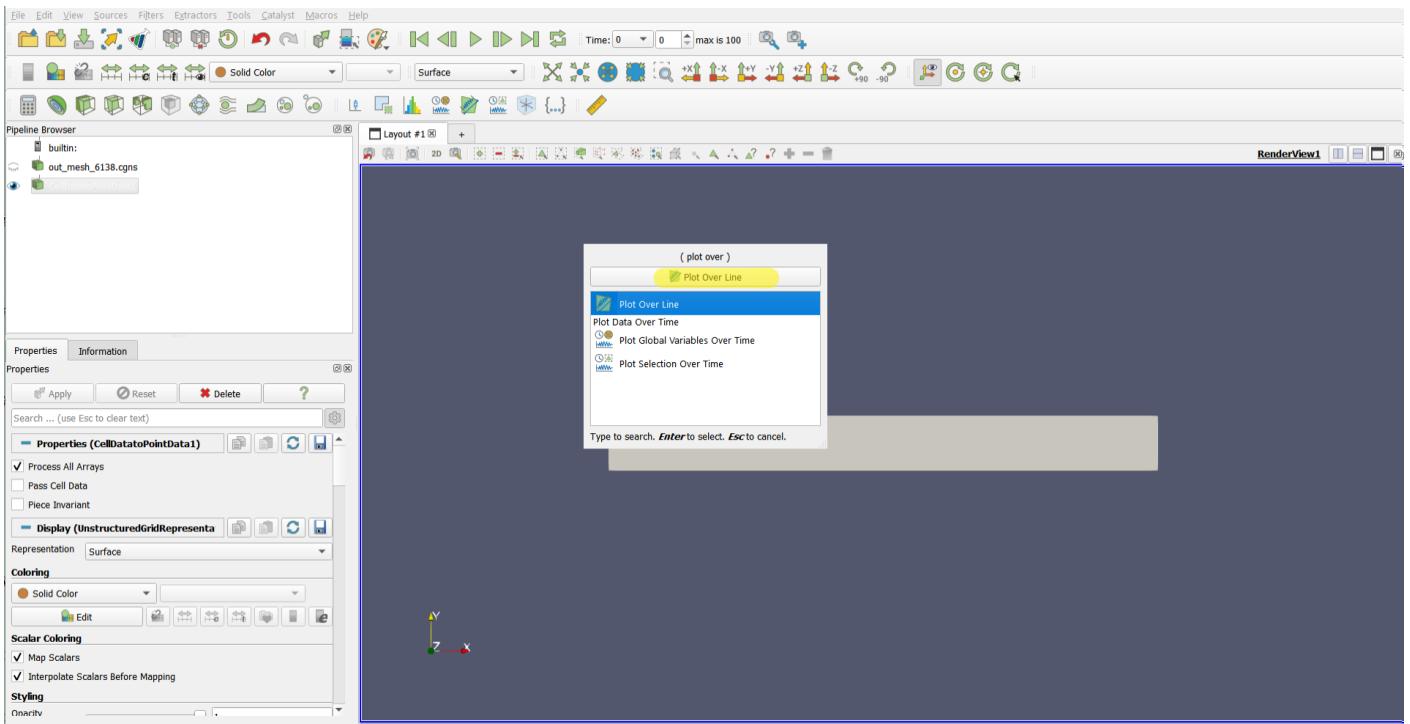


FIGURE 43 – Sélection de l’outil *Plot Over Line*

Note : À partir de maintenant l’utilisation du raccourci Ctrl + espace pour sélectionner les outils ne sera pas mentionnée afin d’alléger le guide et faciliter la lecture mais doit être obligatoirement effectuée par l’utilisateur.

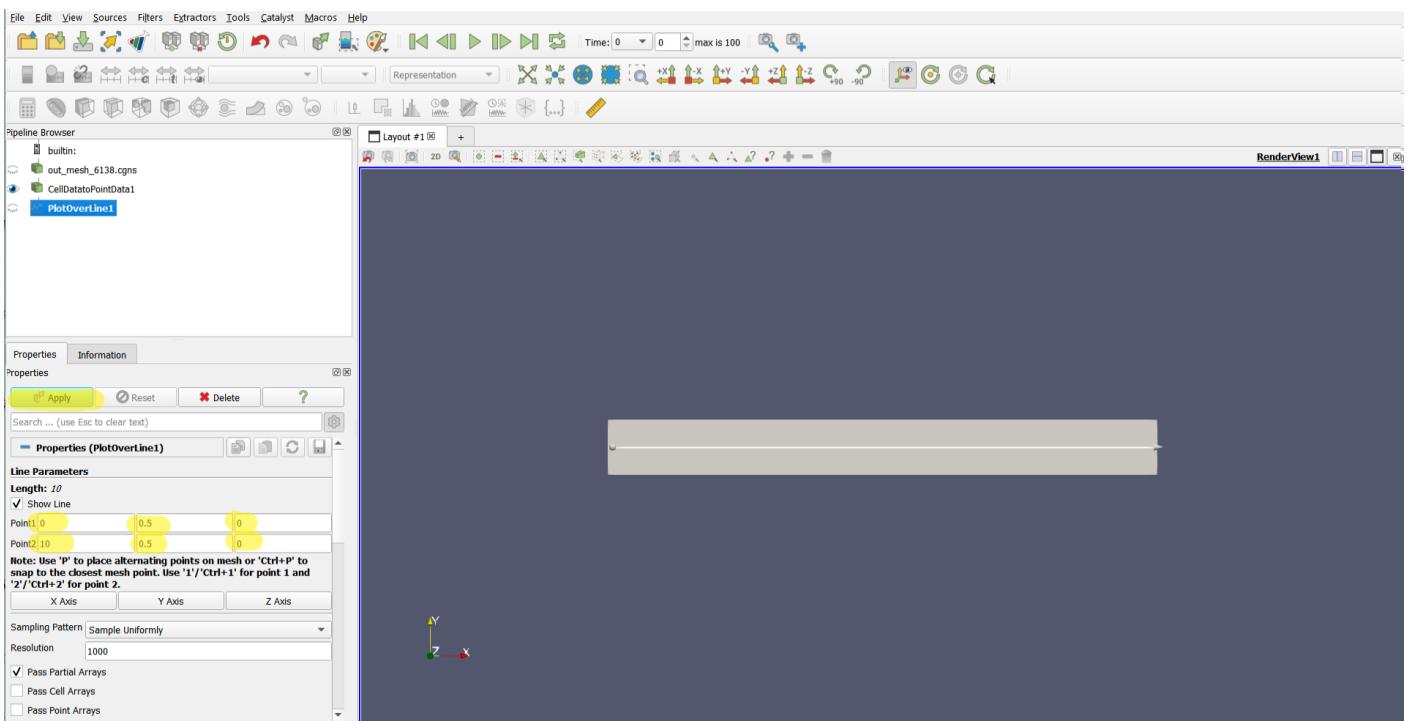


FIGURE 44 – Choix des coordonnées des points de départ et d’arrivée de la ligne, puis *Apply*.

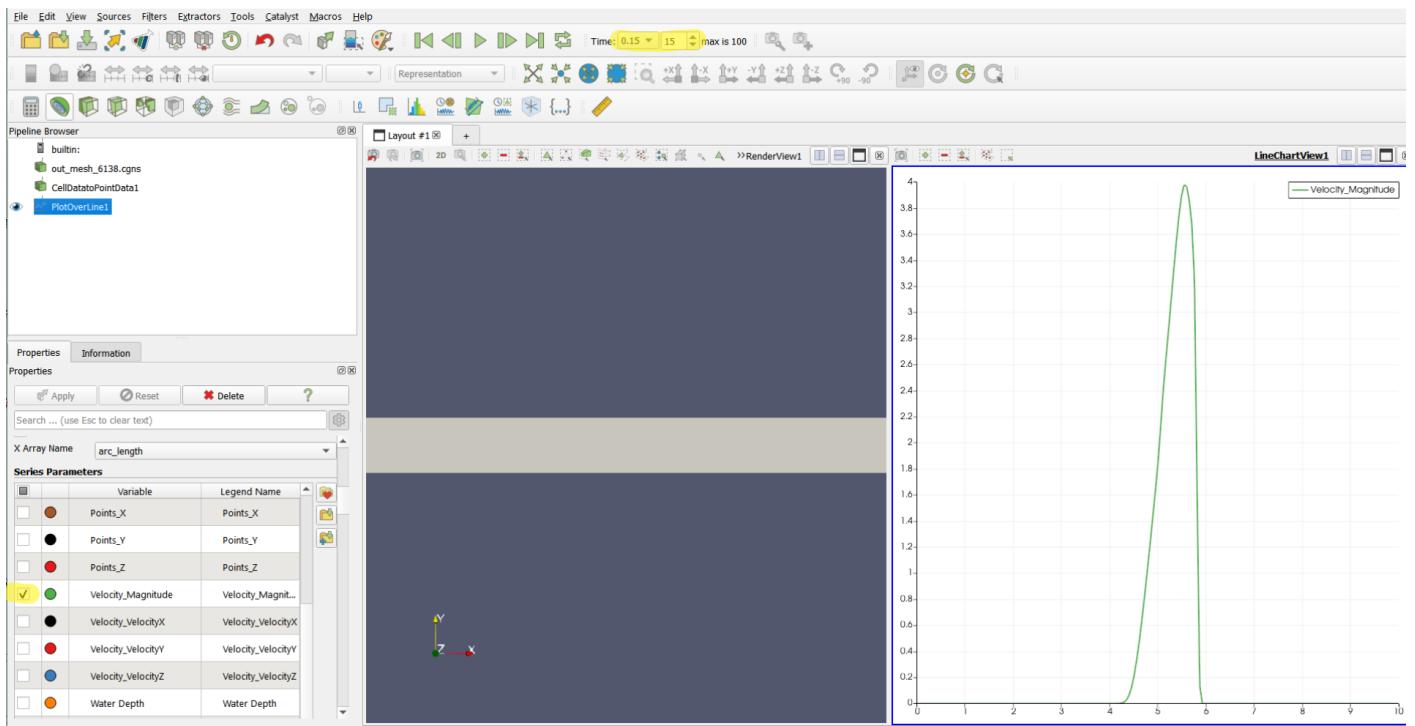


FIGURE 45 – Sélection des attributs de la solutions à charger et du temps de la solution (si le menu de gauche ne s'affiche pas correctement, effectuer un clic gauche sur le graphe a droite).

On va ensuite effectuer à nouveau l'opération sur le fichier du second ordre ainsi que sur la solution exacte en sélectionnant le fichier *exact_sol.vtk.series*.

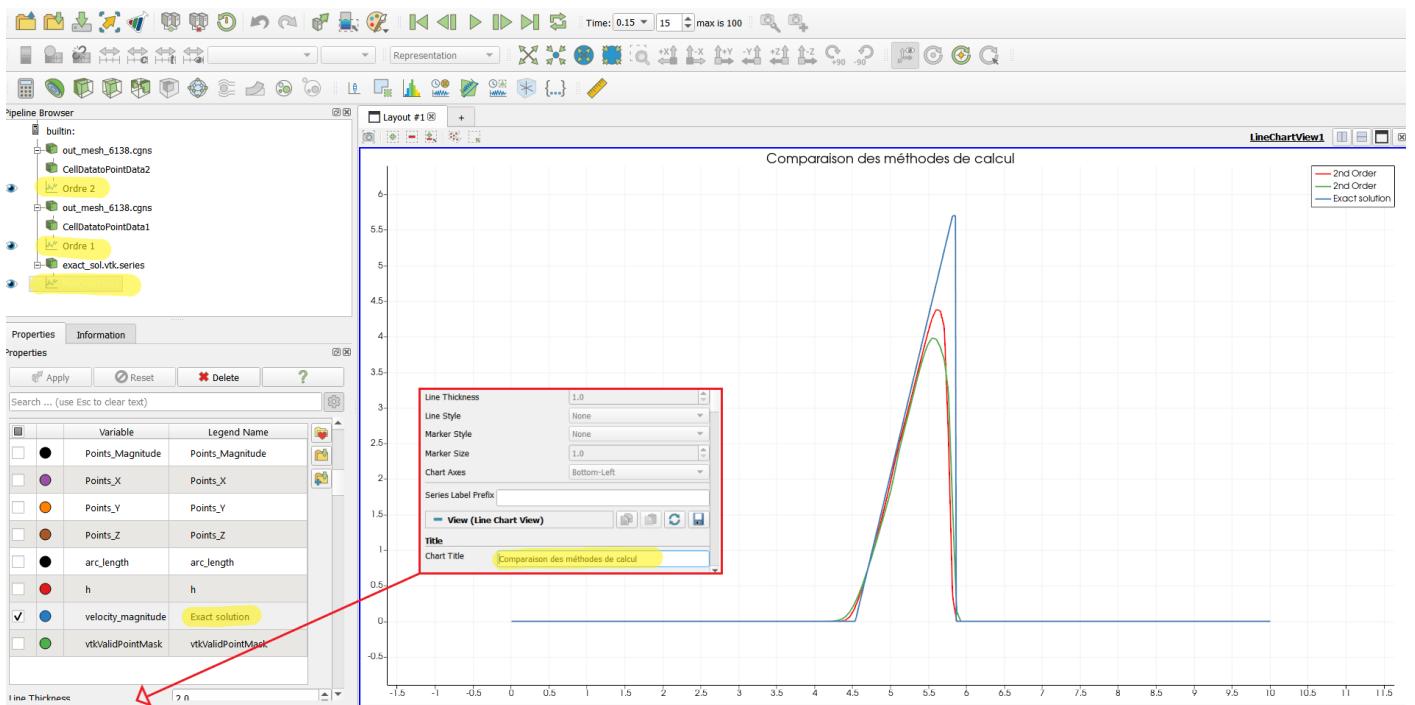


FIGURE 46 – Modification de la légende et du titre du graphe (situé en dessous)

Il faut noter que de nombreux paramètres sont disponibles afin de modifier l'apparence des graphes (échelle verticale et horizontale, légende...). Ces modifications sont simples et ne sont donc pas toutes détaillées dans ce guide.

On peut alors sauvegarder l'état du travail en allant dans *File* en haut à gauche de la fenêtre de travail, puis en sélectionnant *Save State* et en choisissant le fichier de réception. Pour ouvrir à nouveau le projet, il suffit de faire : *File* puis *Load State* et en sélectionnant la sauvegarde créée. Attention à ne pas déplacer ni modifier les fichiers entre temps.

4.2.2 Visualisation du fichier rect_6138_bump

Le fichier `out_rect_6138_bump.cgns` est similaire au précédent à la différence que la surface n'est pas plate. On va effectuer la simulation sur le cluster de calcul afin de montrer pas à pas avec un cas concret une utilisation simple de ce dernier.

On va commencer par accéder au noeud de connexion :

```
#Depuis votre emulateur de terminal linux (type MobaXterm)
ssh VotreNomUtilisateur@graham.computeCanada.ca
VotreMotDePasse
```

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

On va ensuite se connecter par tunnel SSH au noeud de calcul attribué. Pour plus de détails, l'utilisateur peut se référer à la section 4.1.

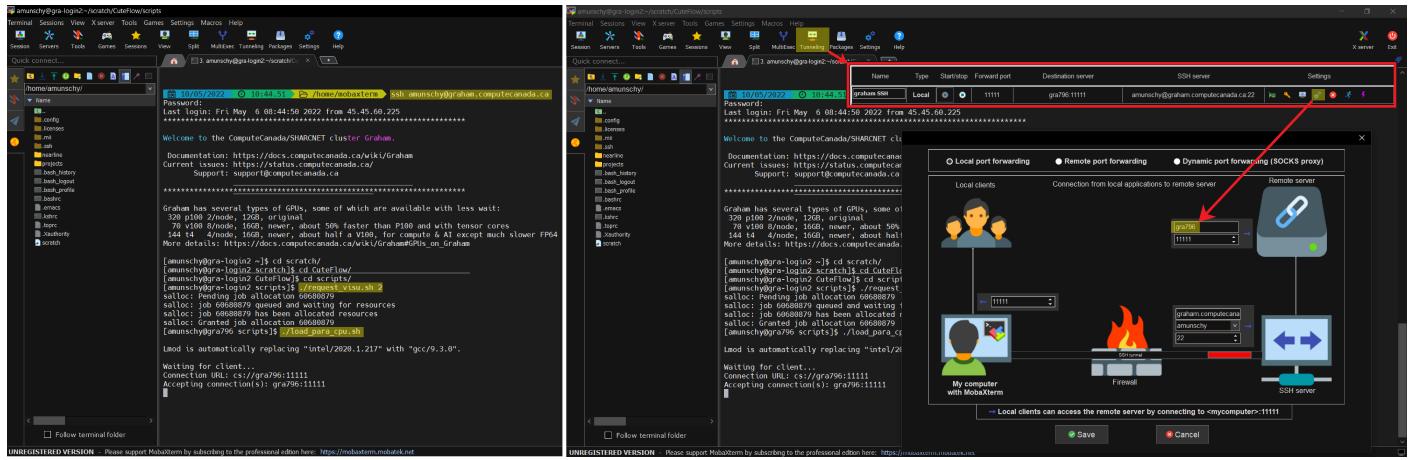


FIGURE 47 – Connexion au noeud de calcul

Une fois la connexion établie, l'utilisateur lance ParaView localement puis se connecte via l'icône *Connect*. La ligne `builtin` dans le *Pipeline Browser* (à gauche) est alors remplacé par *Graham_Paraview* (`cs://localhost:11111`)

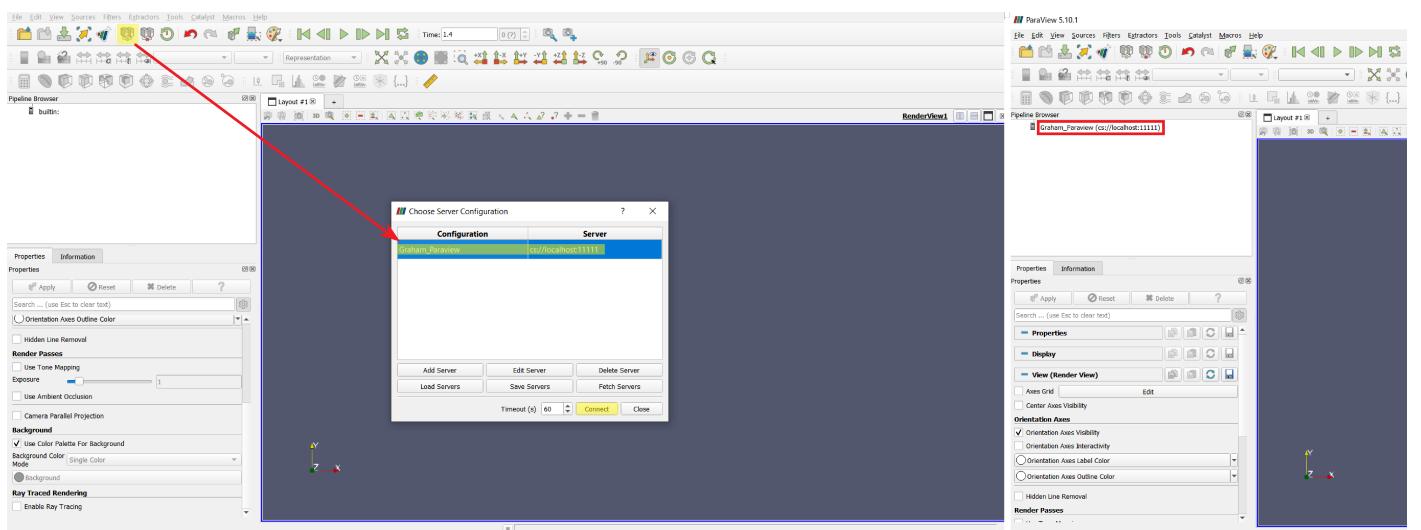


FIGURE 48 – Lancement de ParaView

L'utilisateur va à présent pouvoir ouvrir le fichier `out_rect_6138_bump.cgns` sur le serveur. On va ensuite effectuer la démarche afin de tracer le graphe désiré le long d'une ligne, cette démarche est la même que celle décrite dans la section suivante (section 4.2.3). Voici le genre de graphe simple que l'on peut tirer de ce travail, en comparant ici les hauteurs d'eau (FSE : Free Surface Elevation) :

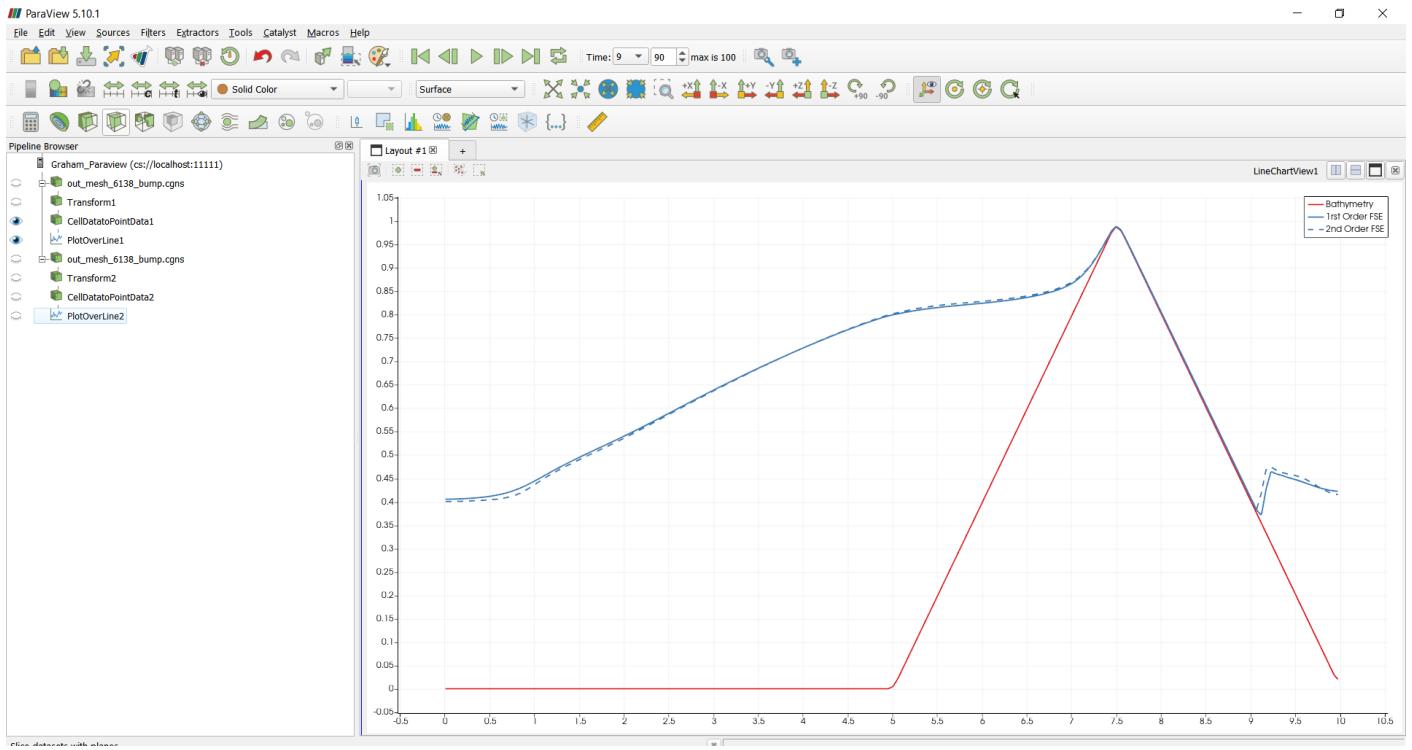


FIGURE 49 – Résultat du travail sur `out_mesh_6138_bump.cgns`

4.2.3 Visualisation du fichier `out_rect_95459_bump.cgns`

On montre dans cette section la marche à suivre pour afficher la solution contenue dans le fichier `out_rect_95459_bump.cgns`. On cherche dans un premier temps à afficher la solution le long d'une ligne qui traverse le domaine.

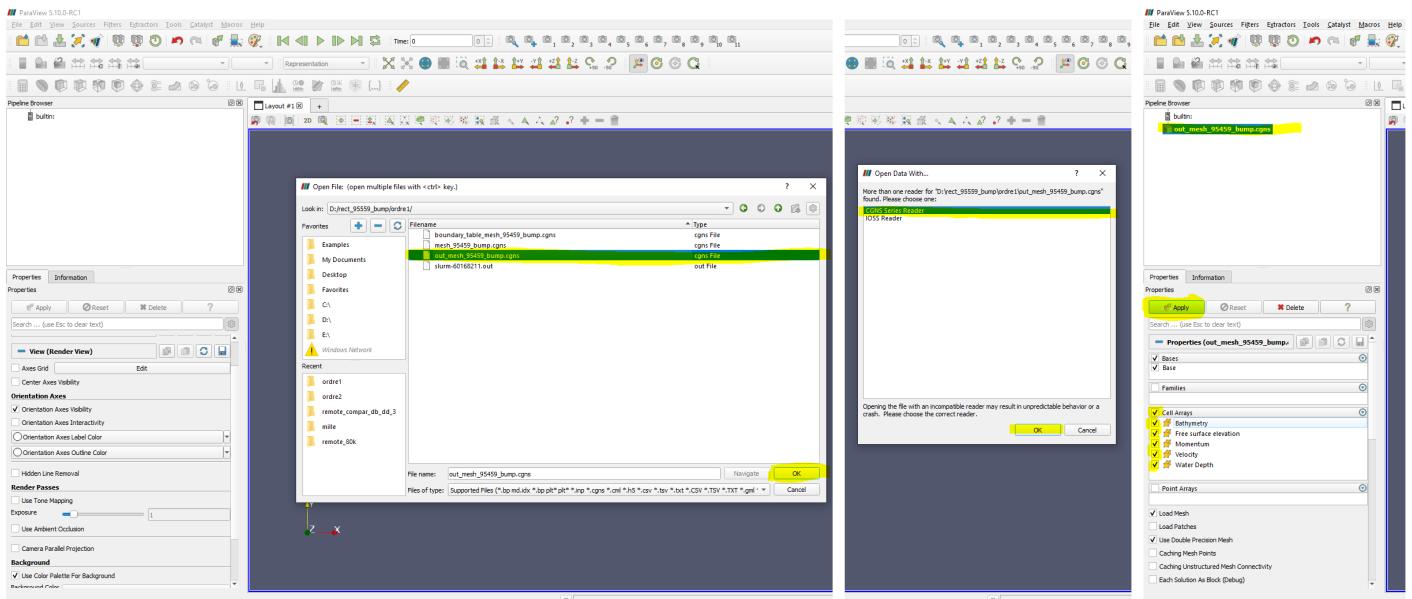


FIGURE 50 – Ouverture du fichier, choix du lecteur et sélection des attributs de la solution à charger

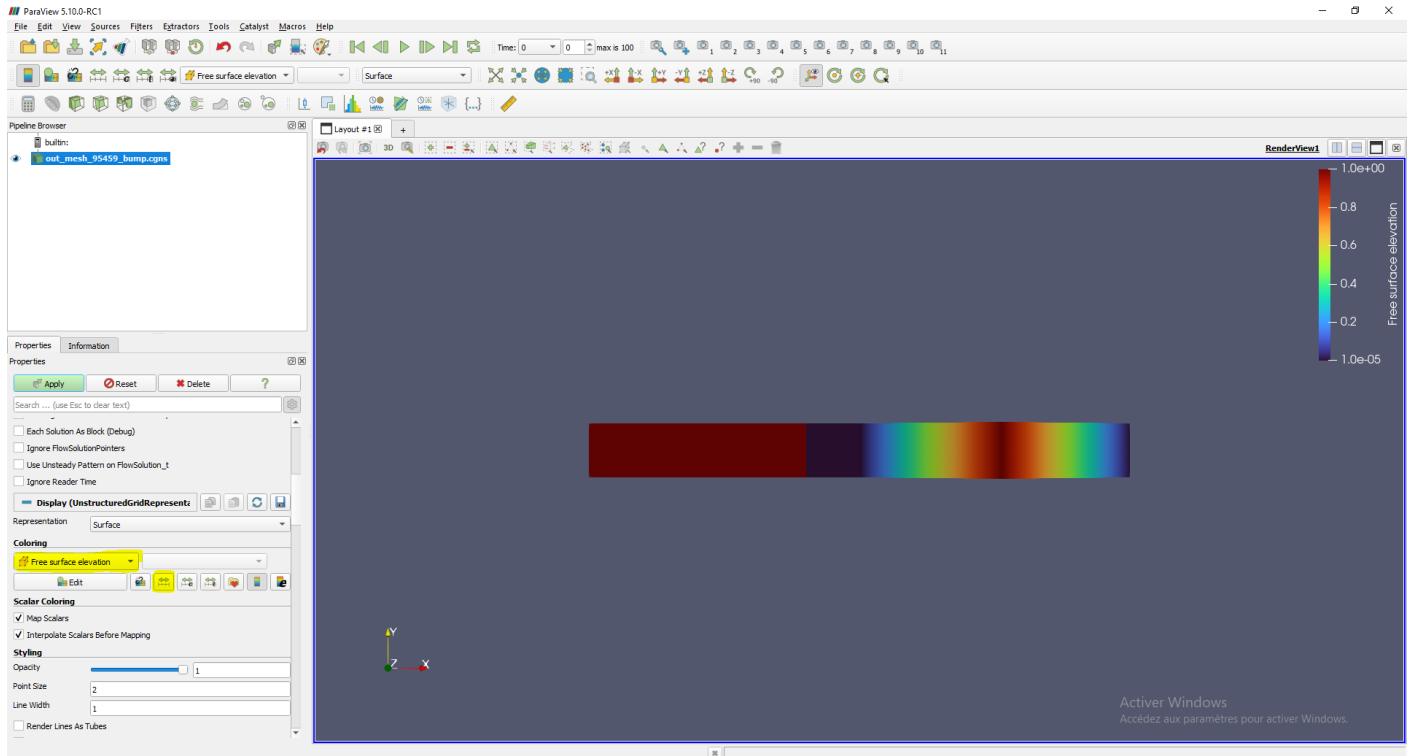


FIGURE 51 – Coloration de la solution en fonction de la hauteur de la surface libre

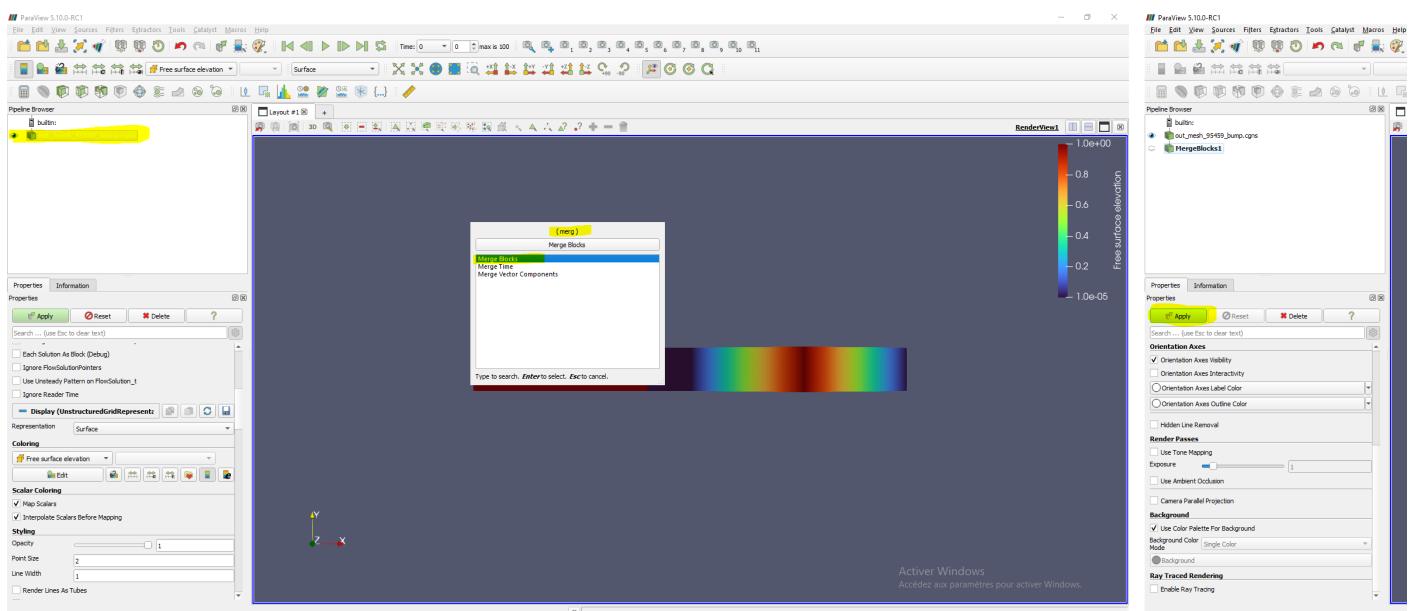


FIGURE 52 – Sélection de l'outil *Merge Block* pour éviter des artefacts de visualisation lorsque plusieurs sous-domaines sont présents. *Apply* pour effectuer l'opération

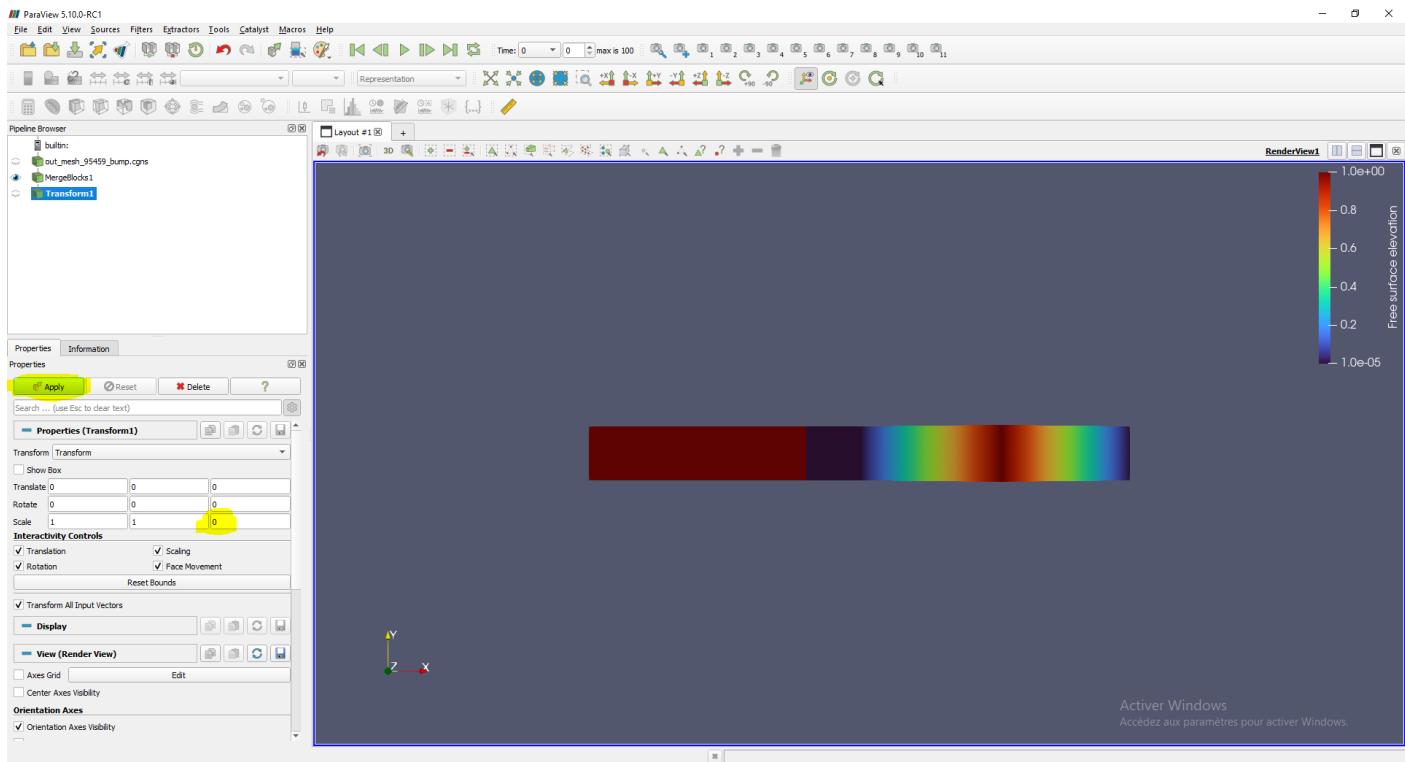


FIGURE 53 – Utilisation de *Transform* pour aplatisir le bump. Permet d'éviter des problèmes lors de la projection sur une ligne.

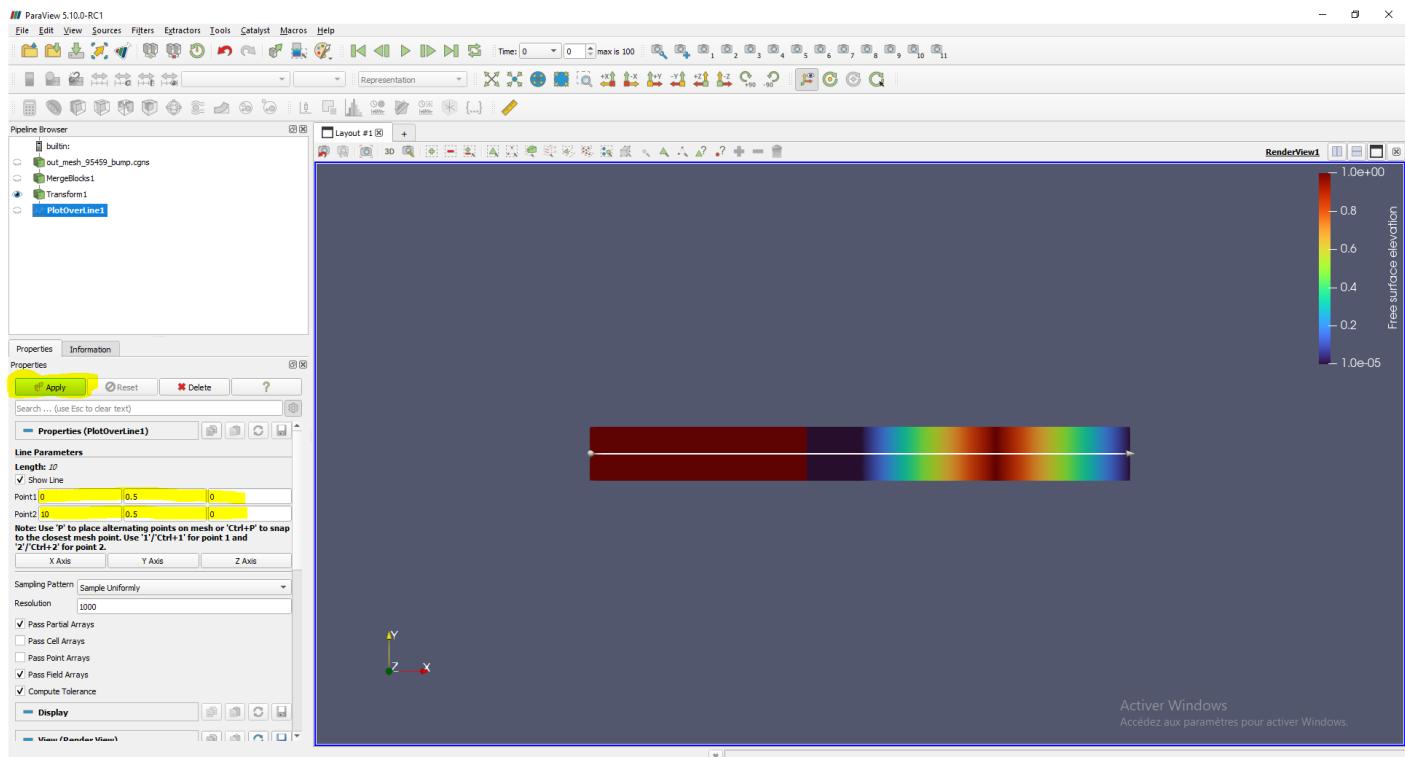


FIGURE 54 – Choix des coordonnées des points de départ et d'arrivée de la ligne avec l'outil *Plot over line*

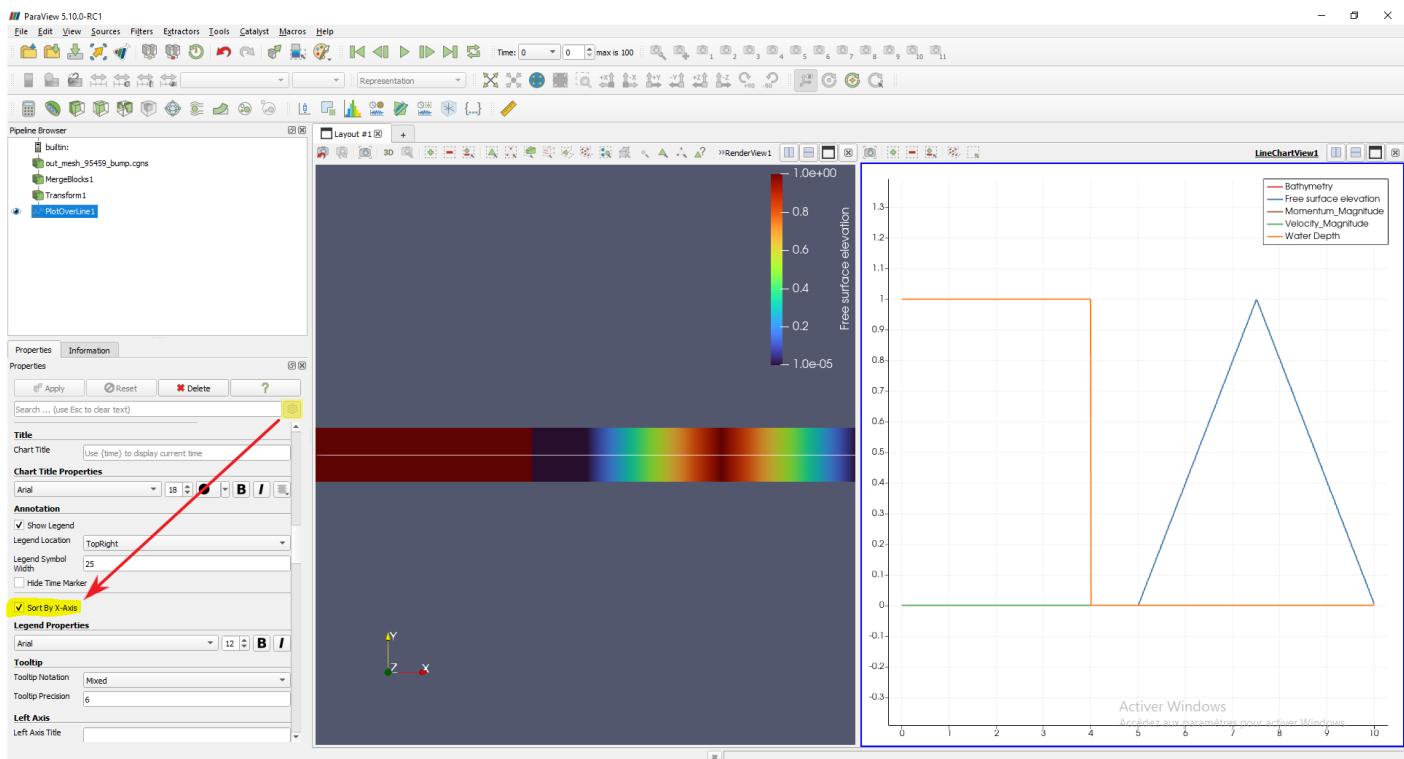


FIGURE 55 – Sélection du mode de paramètre avancé et du paramètre *Sort by X-axis* pour éviter d'avoir des points dans le mauvais ordre (particulièremment utile pour les maillages réels)

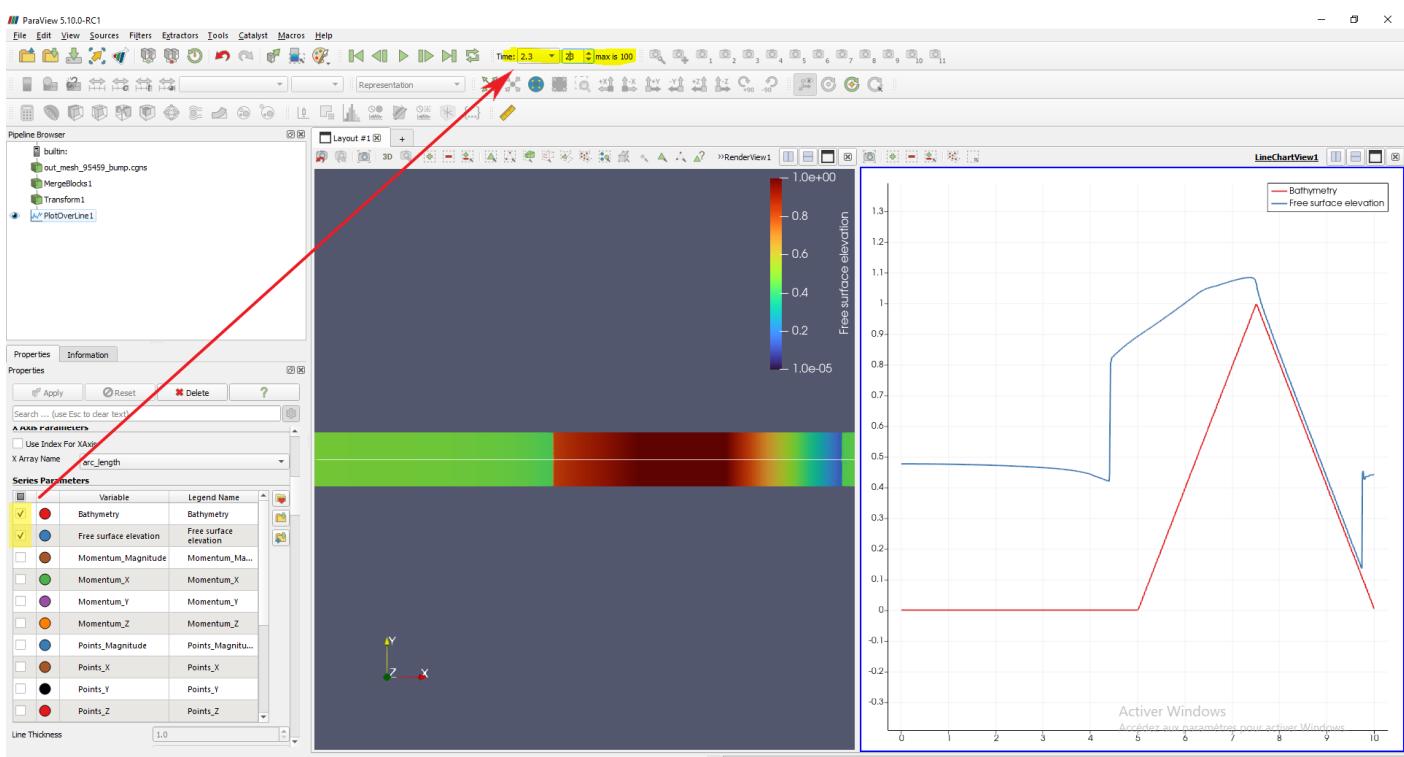


FIGURE 56 – Sélection des attributs de la solutions à charger et du temps de la solution

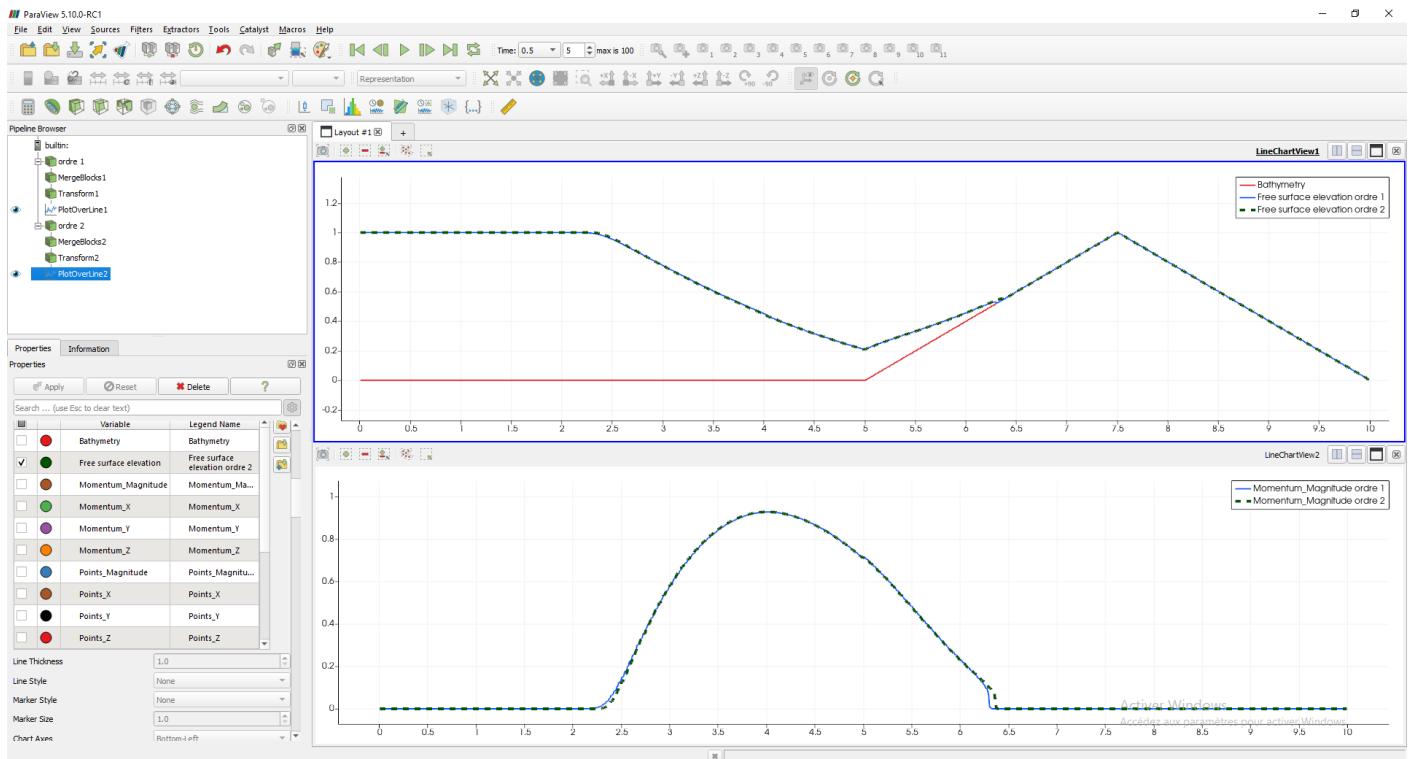


FIGURE 57 – En effectuant le même travail pour un autre fichier de solution, ici la solution d'ordre 2, on peut afficher les deux solutions sur le même graphe.

4.2.4 Visualisation du fichier mille_700k_2_2.cgns

On présente ici comment visualiser la solution sur un domaine réel à partir du fichier mille_700k_2_2.cgns. On montre en particulier comment séparer les zones sèches et mouillées et comment afficher la solution le long d'une ligne brisée dans un premier temps. Dans un second temps, on va chercher à afficher les lignes d'inondation.

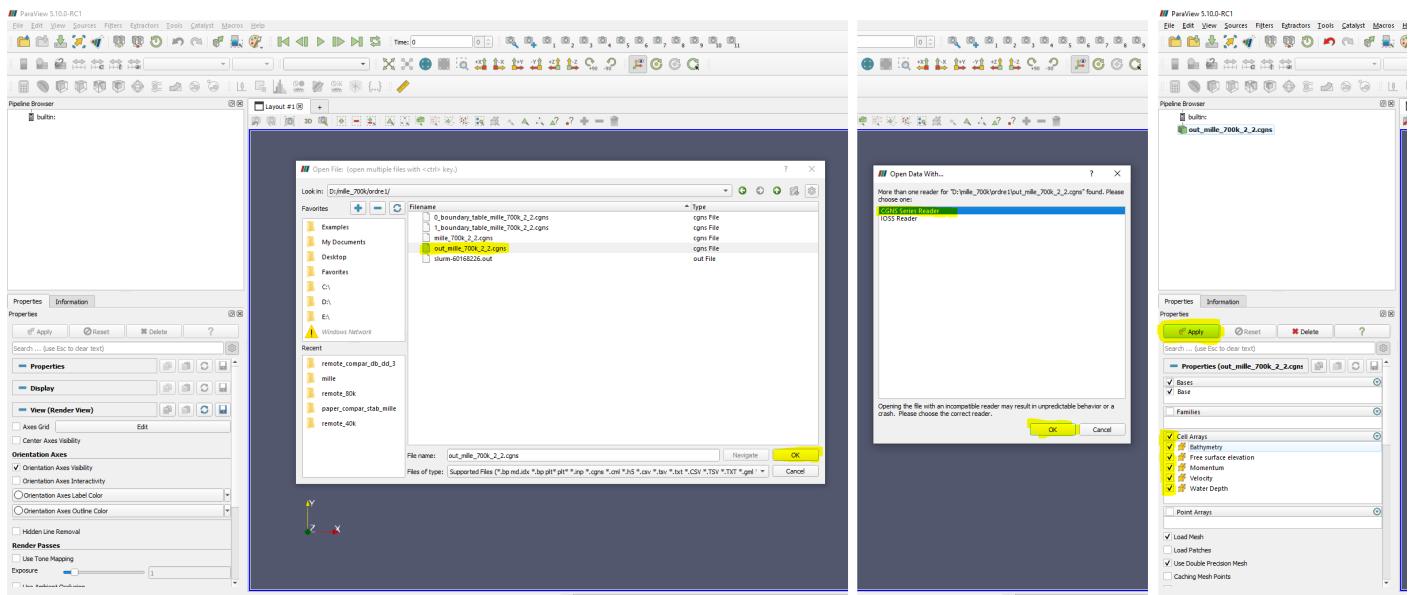


FIGURE 58 – Ouverture du fichier out_mille_700k.cgns, choix du lecteur et sélection des attributs de la solution à charger

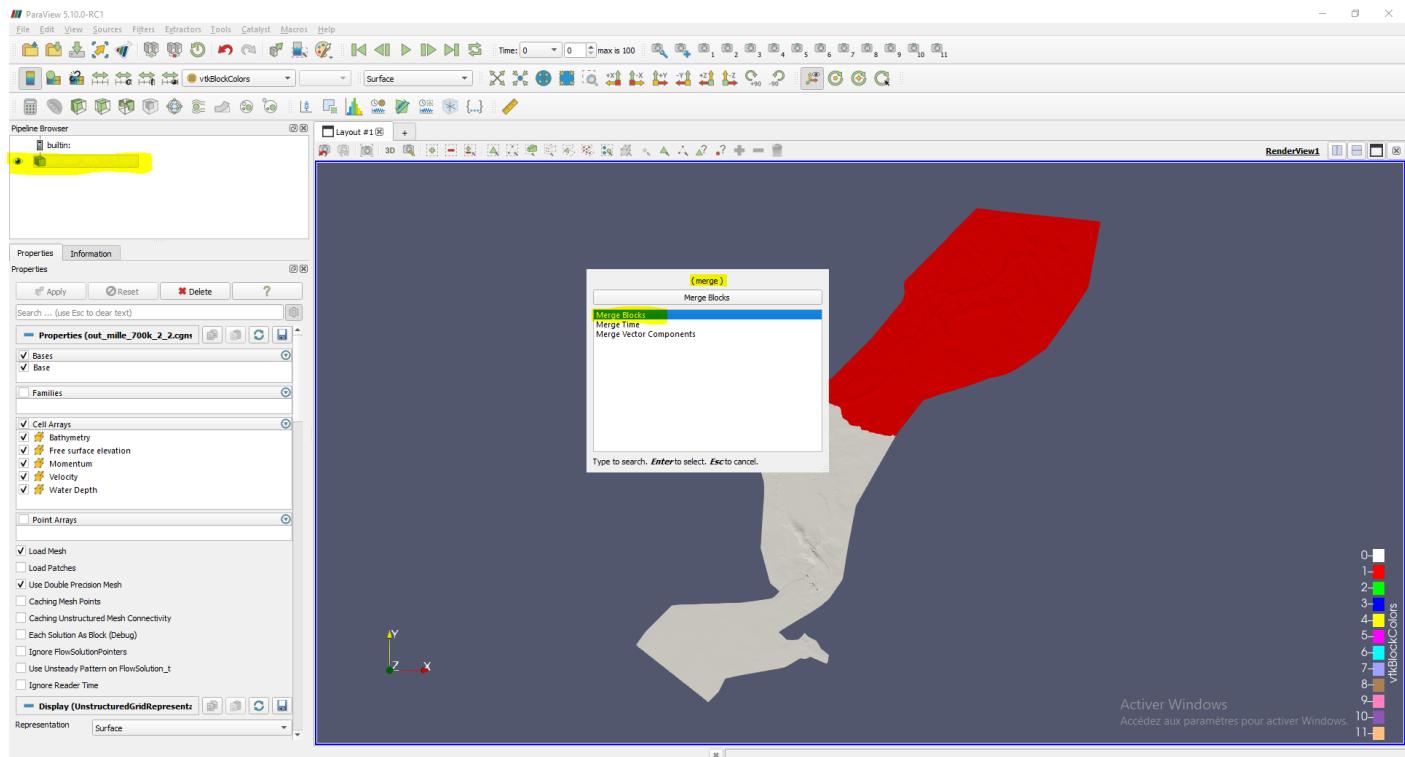


FIGURE 59 – Utilisation de l’outil *Merge Blocks* pour fusionner les sous-domaines. *Apply* pour appliquer les changements

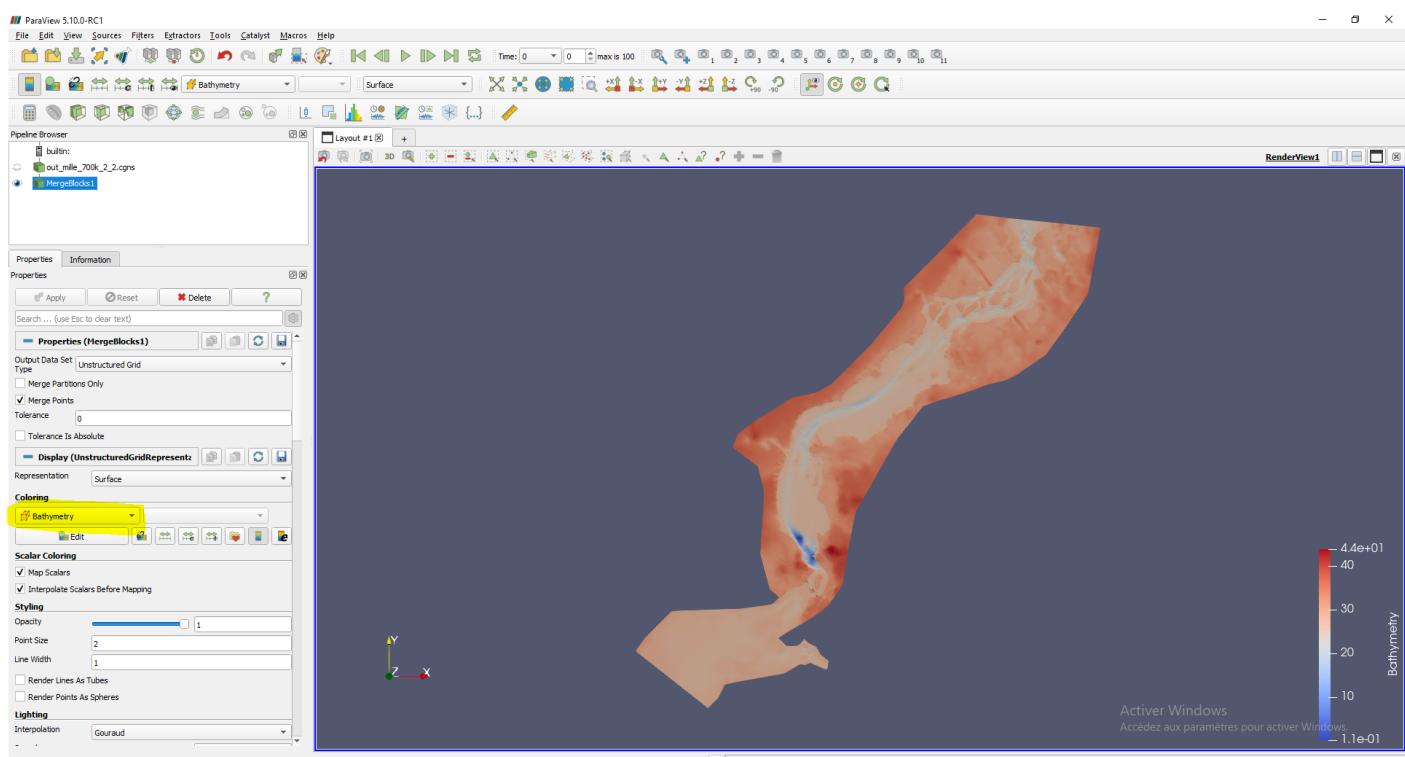


FIGURE 60 – Sélection de l’attribut de la solution à utiliser pour colorer le domaine

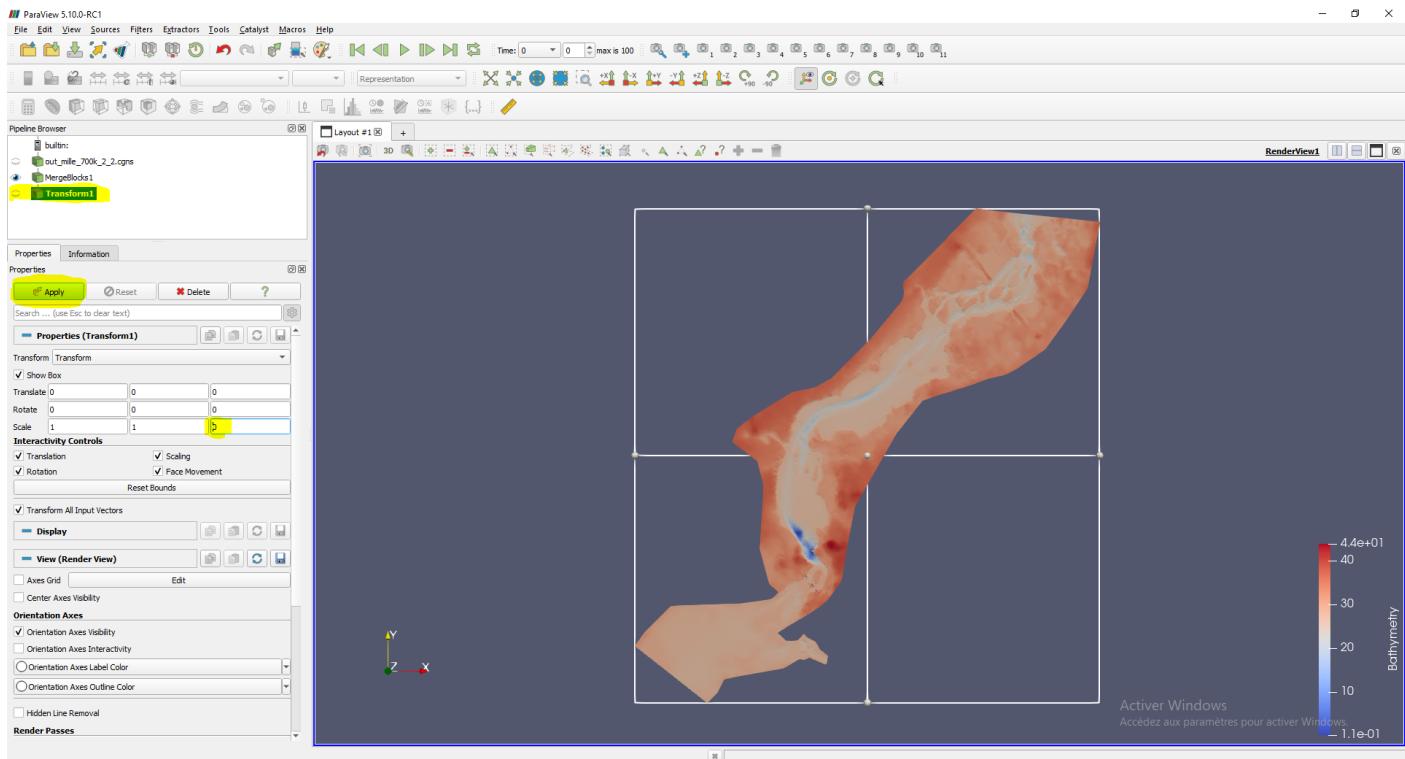


FIGURE 61 – Utilisation de *Transform* pour aplatis le maillage

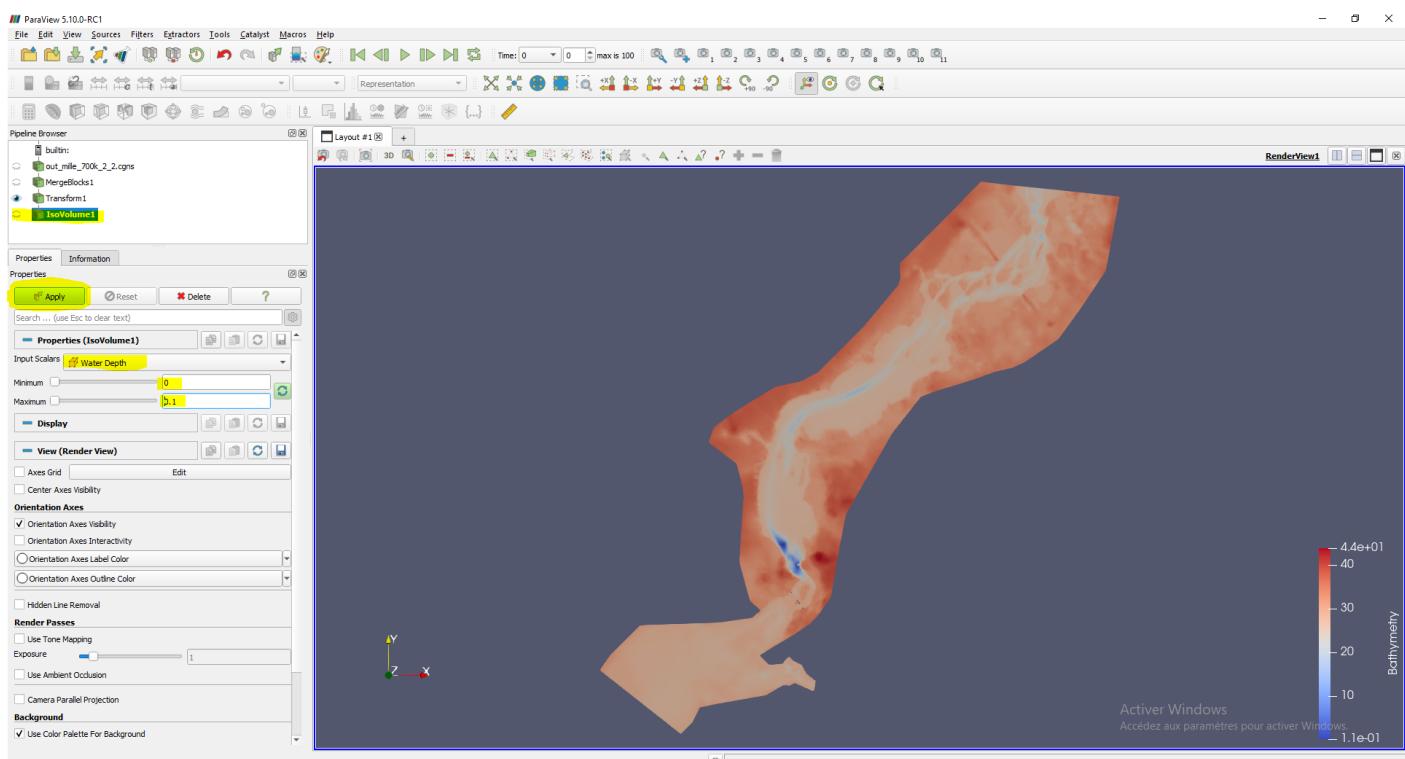


FIGURE 62 – Choix des valeurs à extraire de la solution avec l'outil *Iso volume*. Ici la zone où la *Water depth* (profondeur d'eau) entre 0 et 0.1m sera extraite (zone sèche).

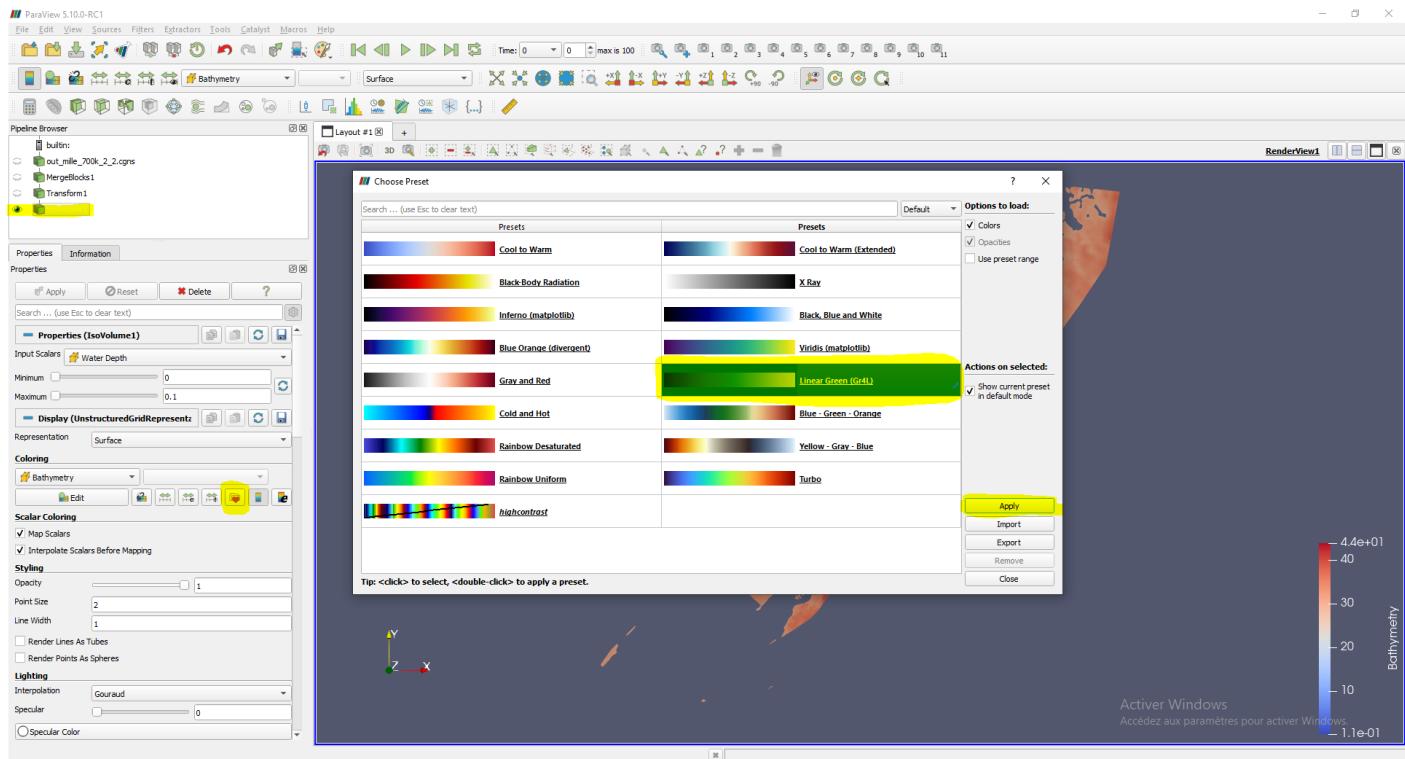


FIGURE 63 – Choix d'une palette de couleurs pour la zone sèche.

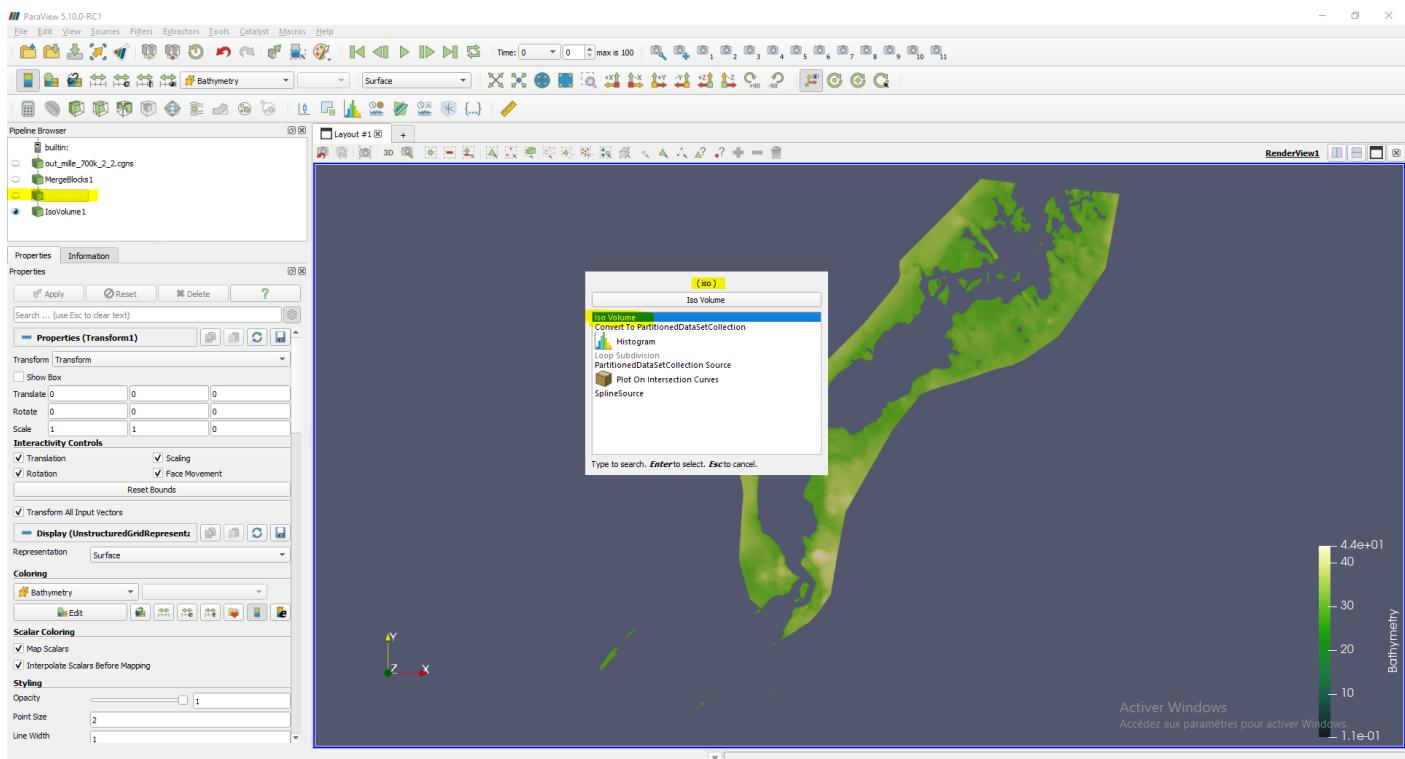


FIGURE 64 – Sélection de l'outil *iso volume* une seconde fois. Attention à bien re-sélectionner l'objet *Transform* dans le fenêtre de gauche et pas *Isovolum1*.

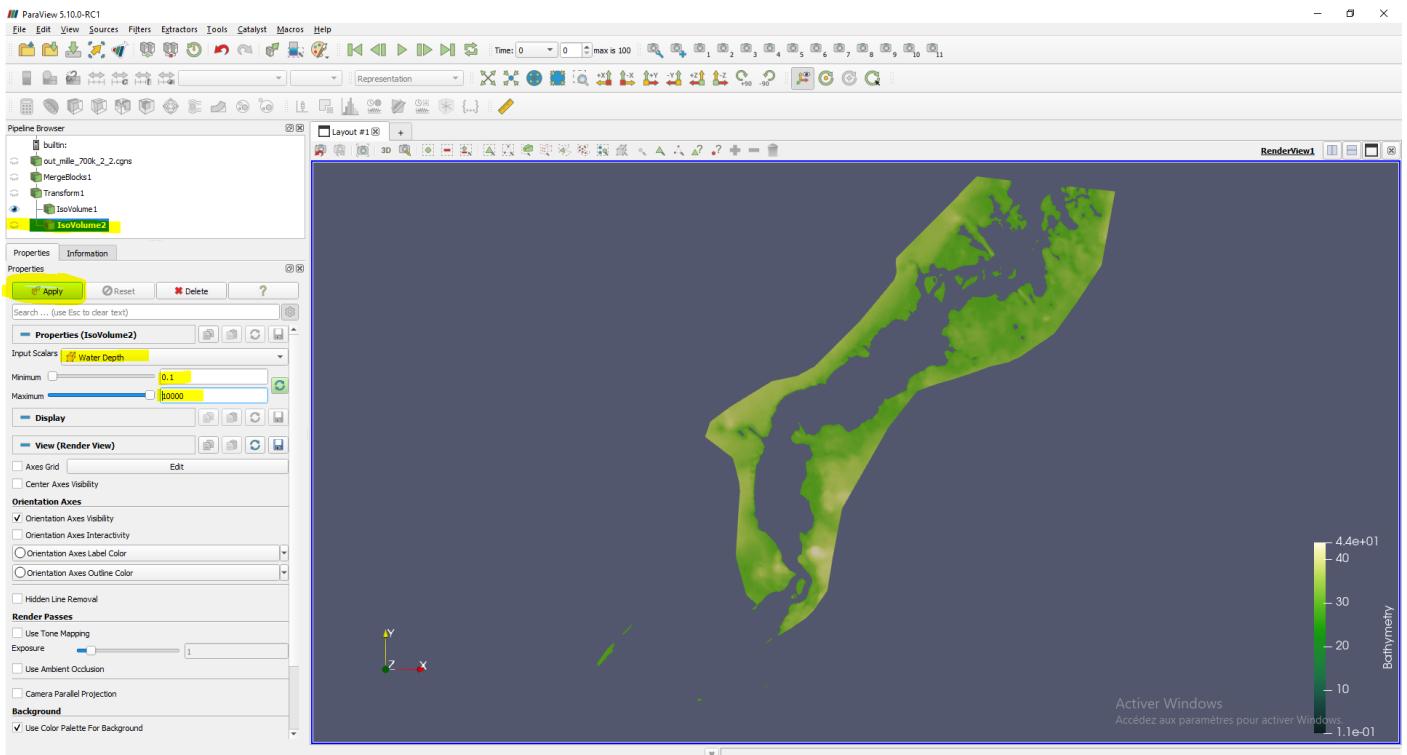


FIGURE 65 – Choix des valeurs de la solution à extraire. Ici la solution ou la *Water depth* est entre 0.1 et 10000m sera extraite (zone mouillée).

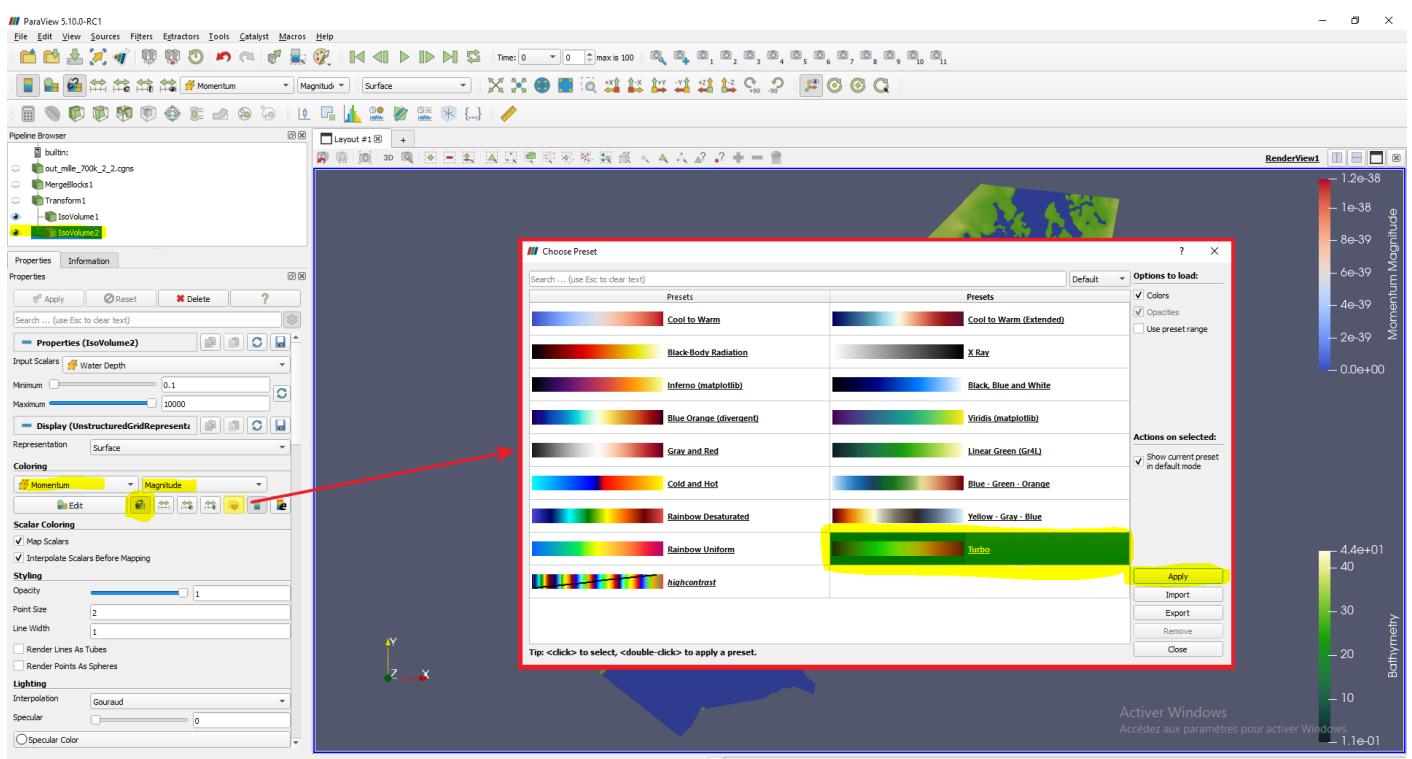


FIGURE 66 – Choix de l'attribut de la solution à utiliser pour la coloration du domaine et séparation des palettes de couleurs de la zone sèche et mouillée. Ensuite, choix de la palette de couleurs pour la zone mouillée (voir figures précédentes pour plus de détails).

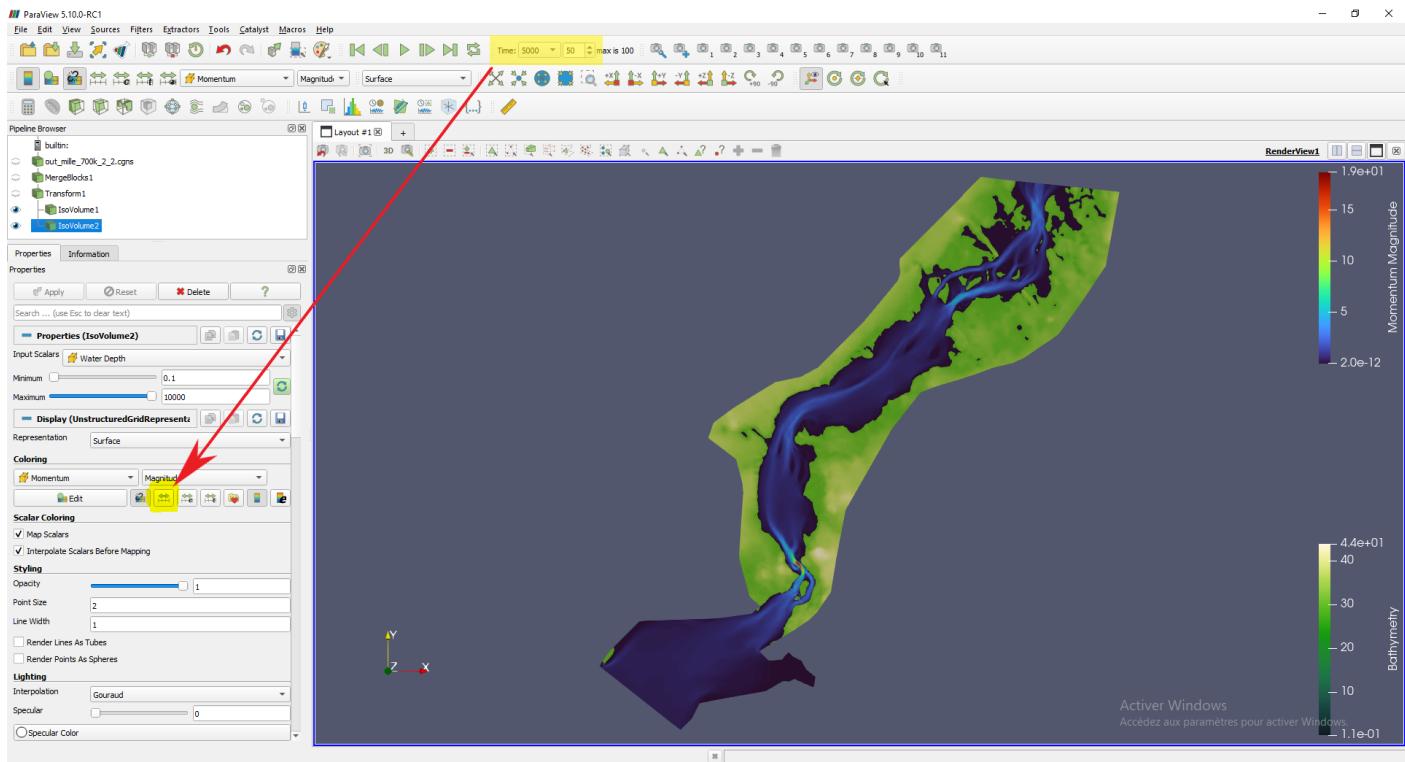


FIGURE 67 – Sélection du temps de la solution et réajustement automatique de l'échelle de couleur.

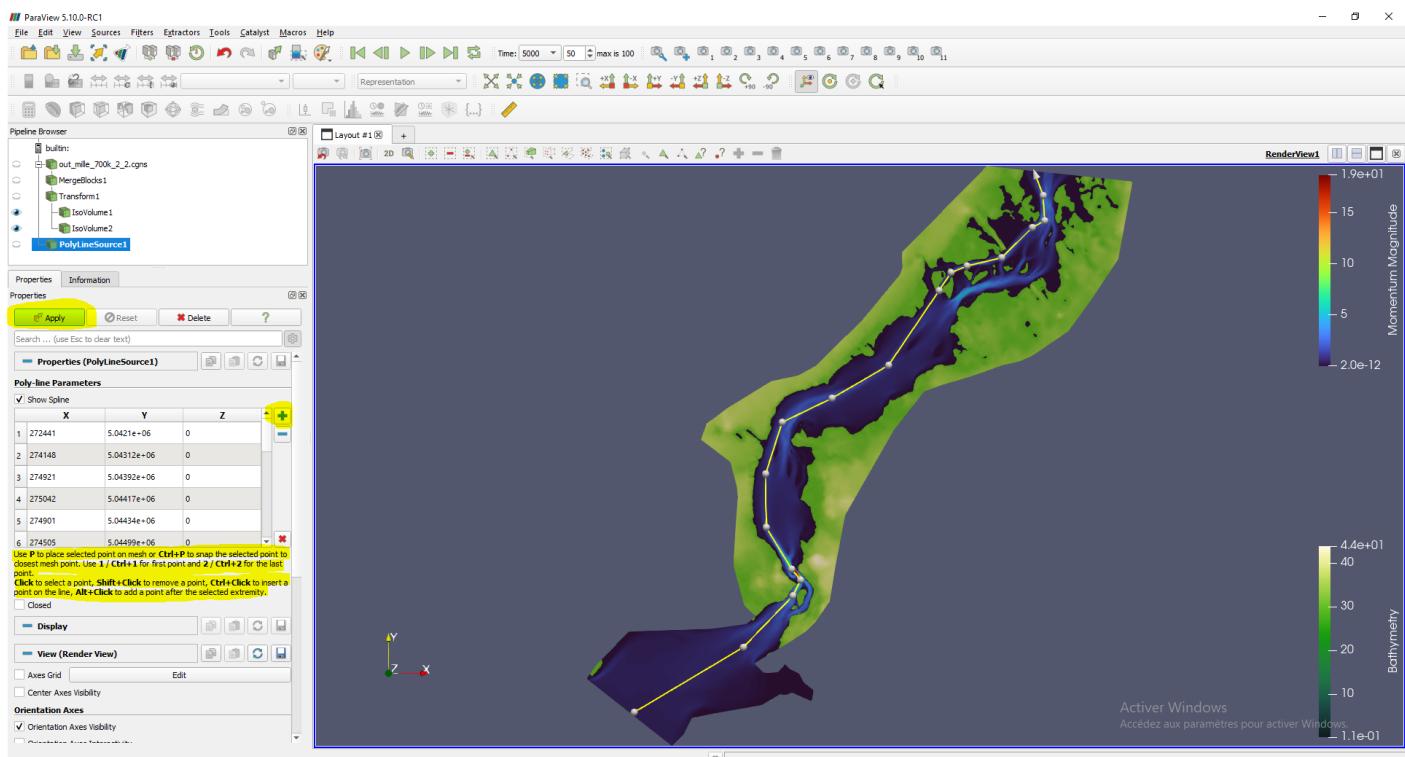


FIGURE 68 – Choix de l'outil *Polyline source* et ajout de différents points de la ligne.

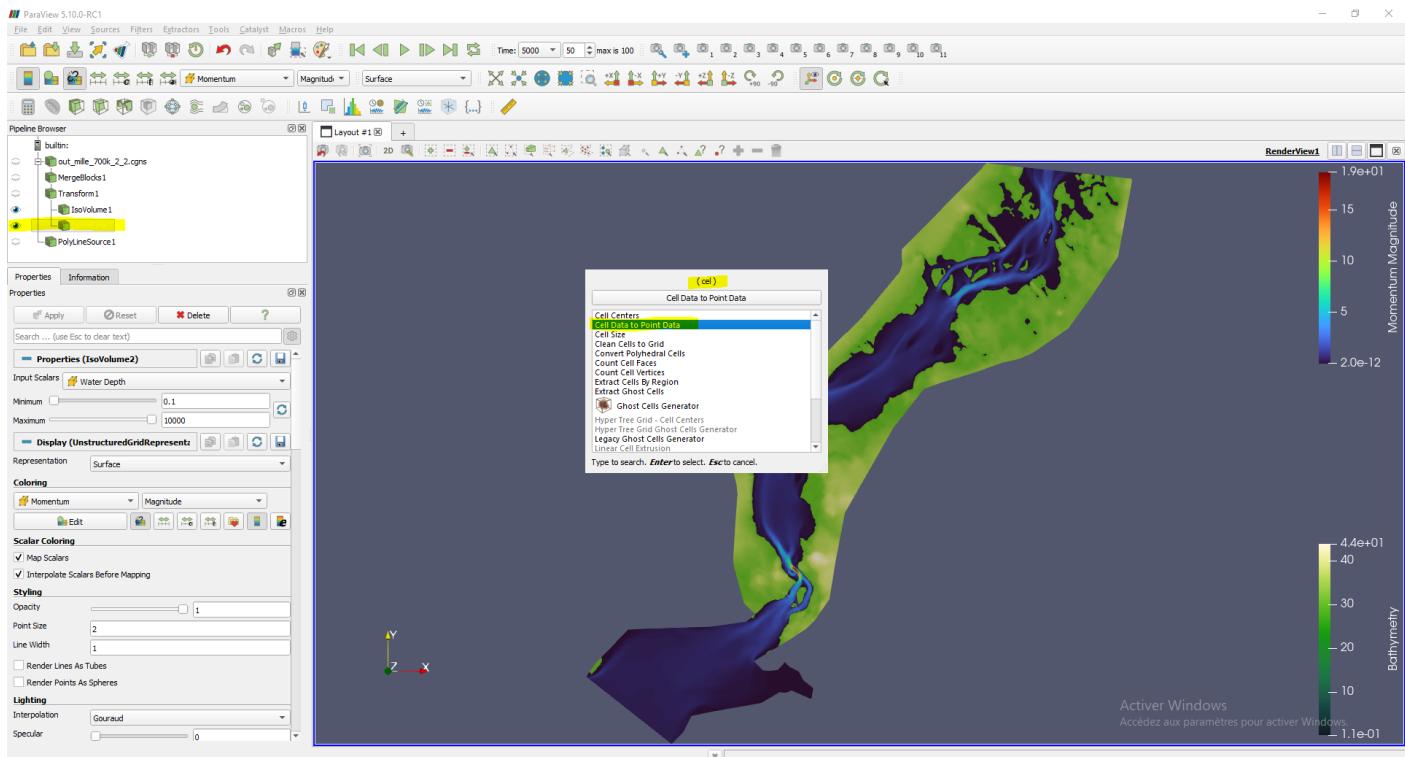


FIGURE 69 – Sélection de l’outil *Cell Data to Point Data* pour passer la solution depuis les cellules sur les noeuds. *Apply* pour appliquer les changements.

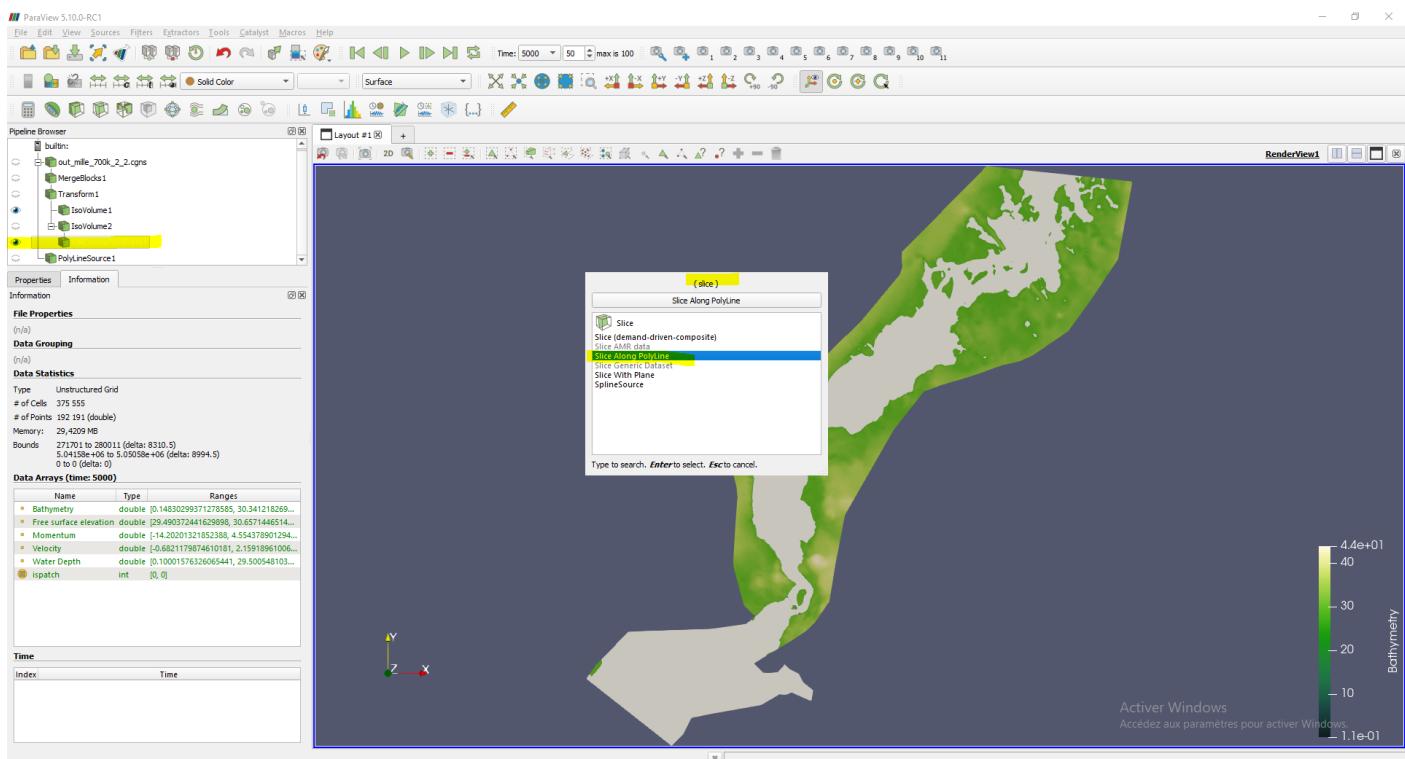


FIGURE 70 – Sélection de l’outil *Slice along polyline* pour projeter la solution sur la ligne bisée.

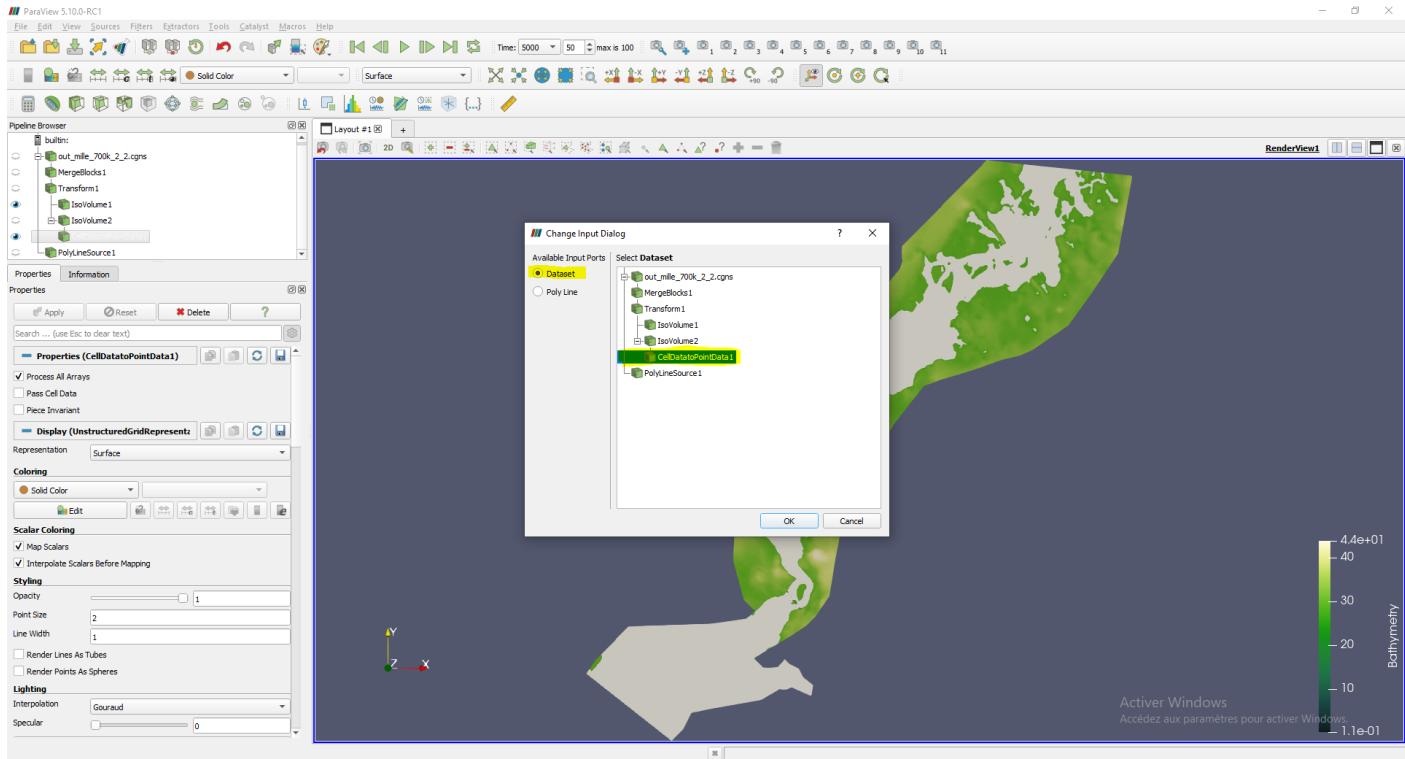


FIGURE 71 – Vérifier que le *Dataset* sélectionné est bien *CellDataToPointData1*.

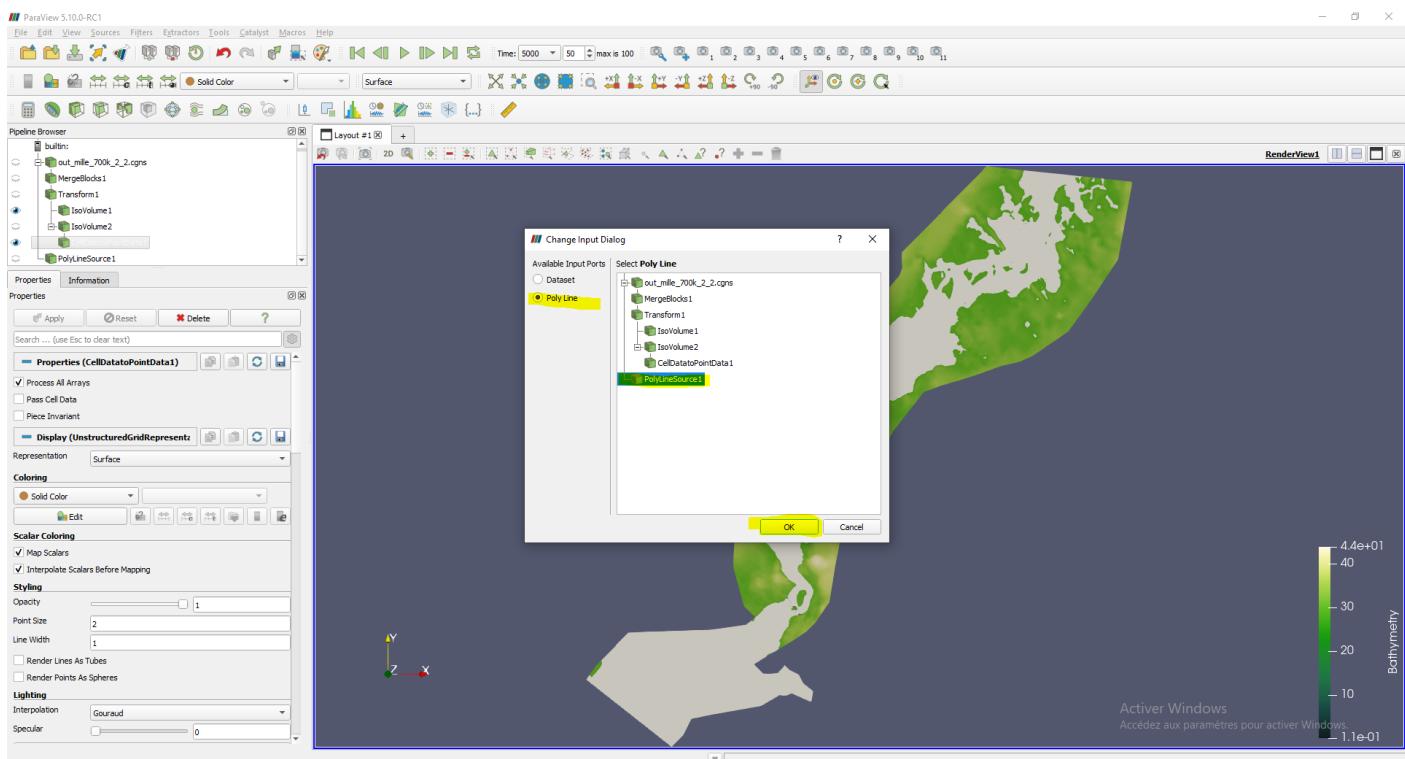


FIGURE 72 – Vérifier que la *Polyline* sélectionnée est bien *PolylineSource1*. Sélectionner ensuite *ok* puis *Apply* pour appliquer les changements.

Si l'on veut utiliser à nouveau cette ligne pour, par exemple, comparer deux méthodes de calcul, on peut cliquer sur l'icône de sauvegarde située au niveau de la catégorie "properties" au-dessus des coordonnées des points. Ainsi, lorsque l'on veut utiliser cette ligne à nouveau, l'outil *Polyline* aura sauvégarde les coordonnées et les utilisera comme valeurs par défaut.

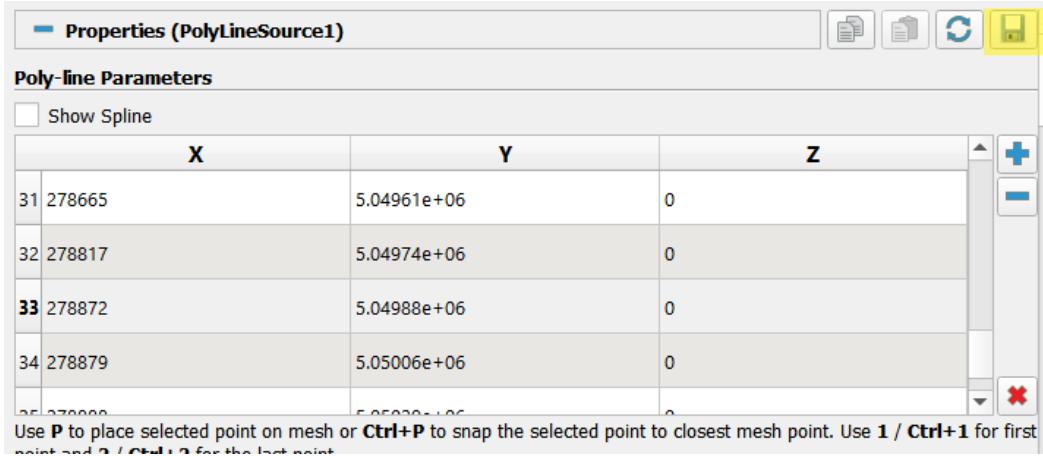


FIGURE 73 – Sauvegarde des propriétés de *Polyline*

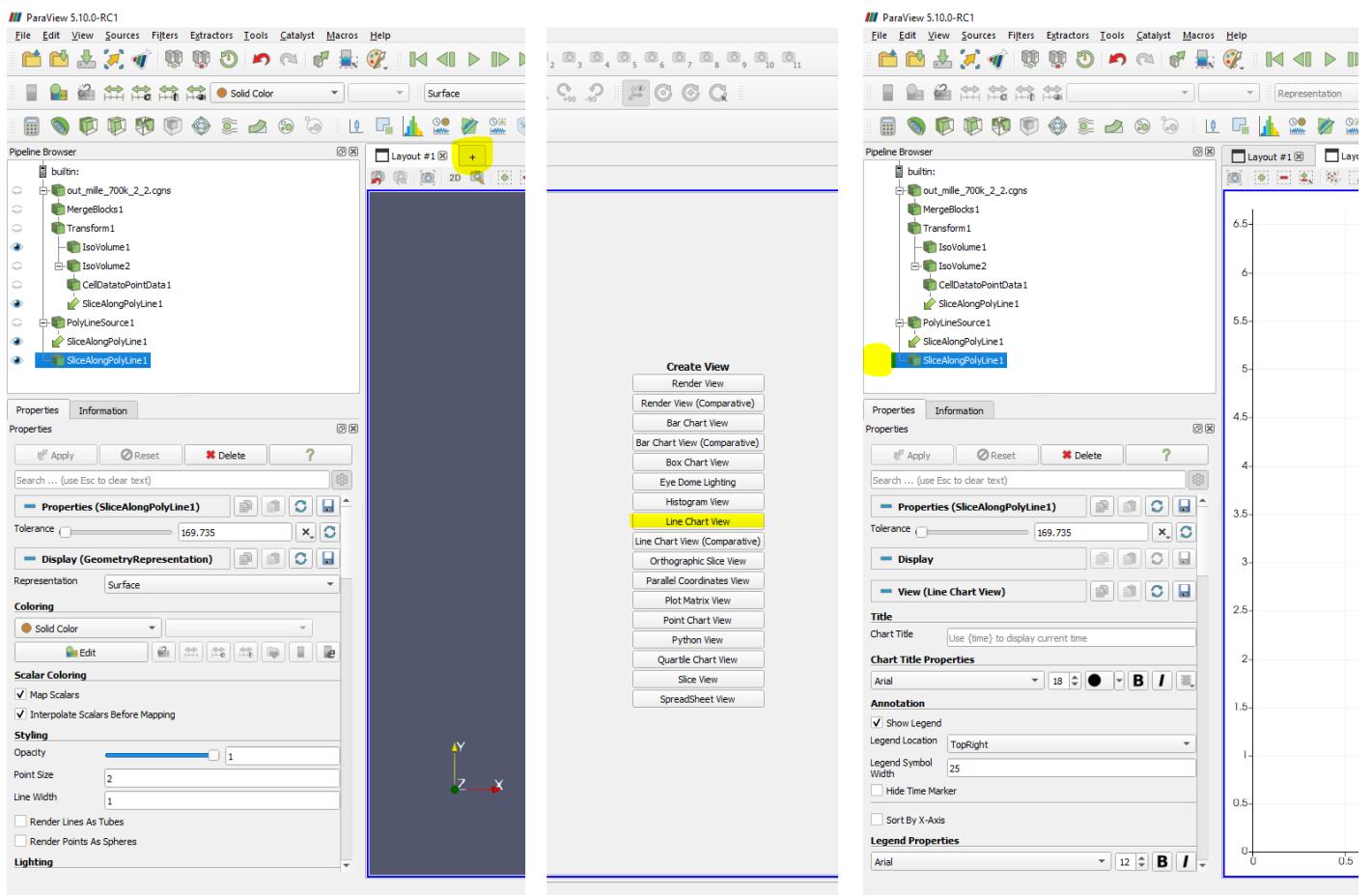


FIGURE 74 – Ouverture d'une nouvelle fenêtre de visualisation, Sélection du mode *Line Chart View* et sélection de l'objet à visualiser (clic sur l'œil bleu situé à gauche de l'objet voulu).

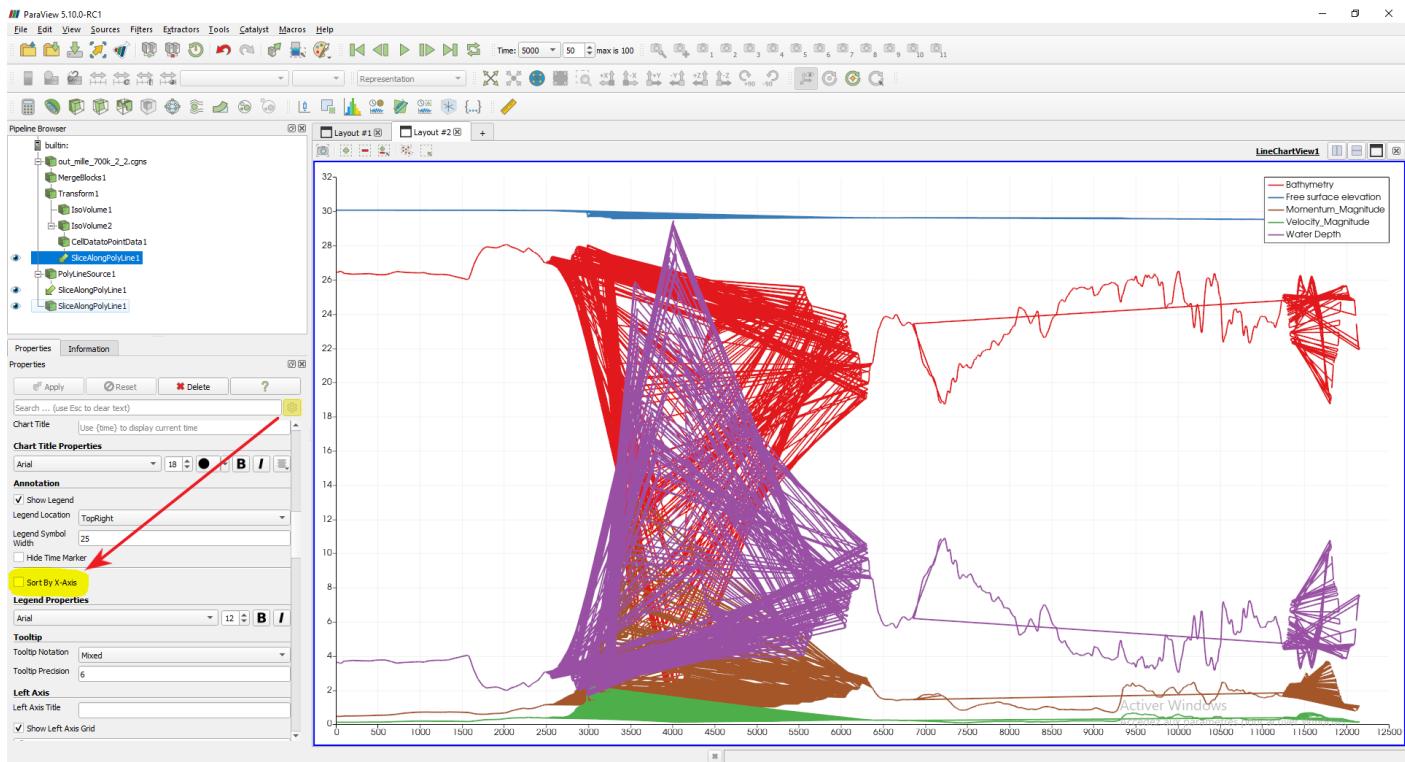


FIGURE 75 – Sélection du mode de paramètre avancé puis sélection de la fonction *Sort by X-Axis*.

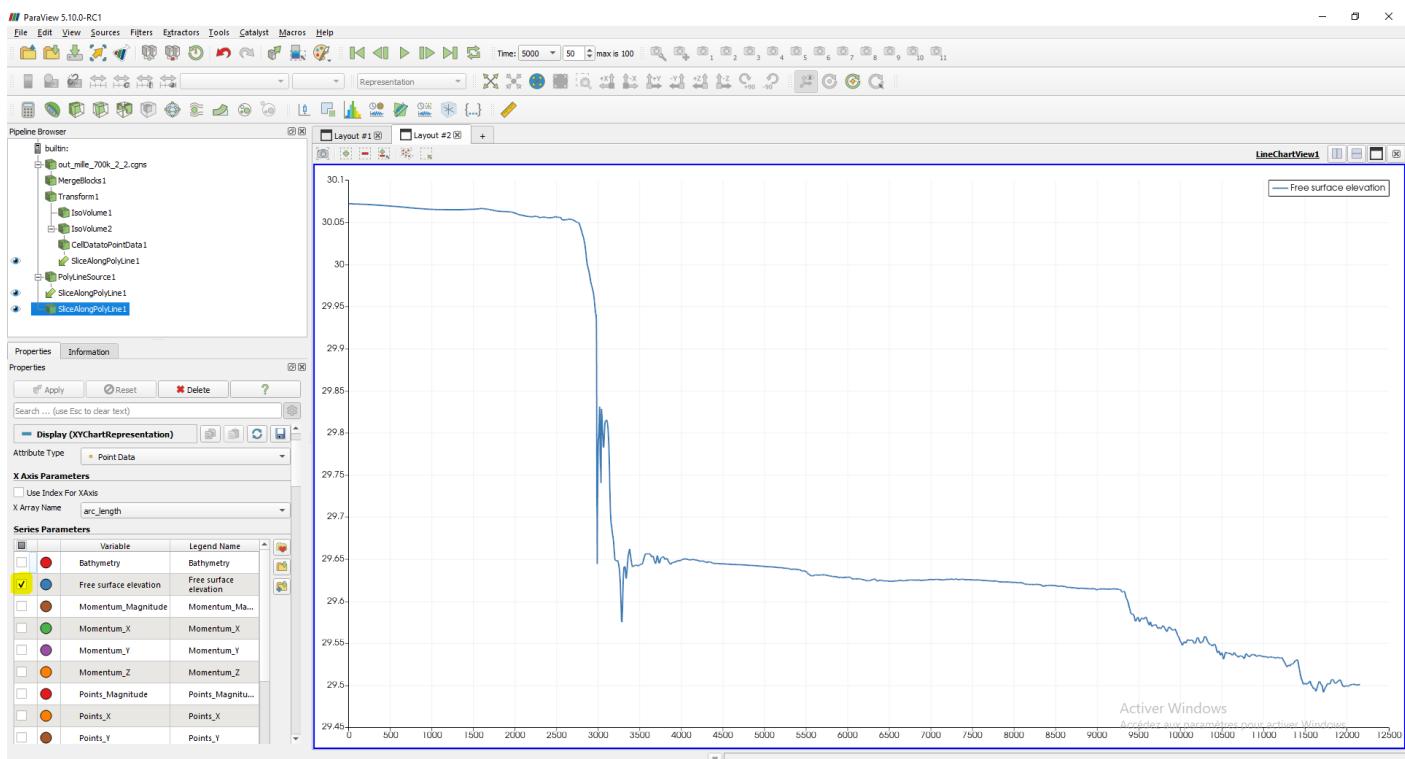


FIGURE 76 – Sélection des attributs de la solution à visualiser.

À présent, nous allons voir comment afficher les lignes d'inondations. La démarche est similaire à la précédente, mais quelques points diffèrent. Voici la démarche à respecter, le détail des outils *Transform* et *IsoVolume* ainsi que l'arborescence finale sont détaillés en figure 77.

- ouverture du fichier *out_mille_700k_2_2.cgns*
- utilisation de *MergeBlocks* pour fusionner les sous domaines - utilisation de *Transform* pour aplatisir le domaine, sélectionner *Bathymétrie* dans *Coloring* afin de colorer le sol.
- utilisation de *CellDataToPointData*

- utilisation de *IsoVolume* afin de colorer l'eau.
- utilisation de *Contour* sur le *CellDataToPointData* afin d'afficher l'interface entre l'eau et la rive.

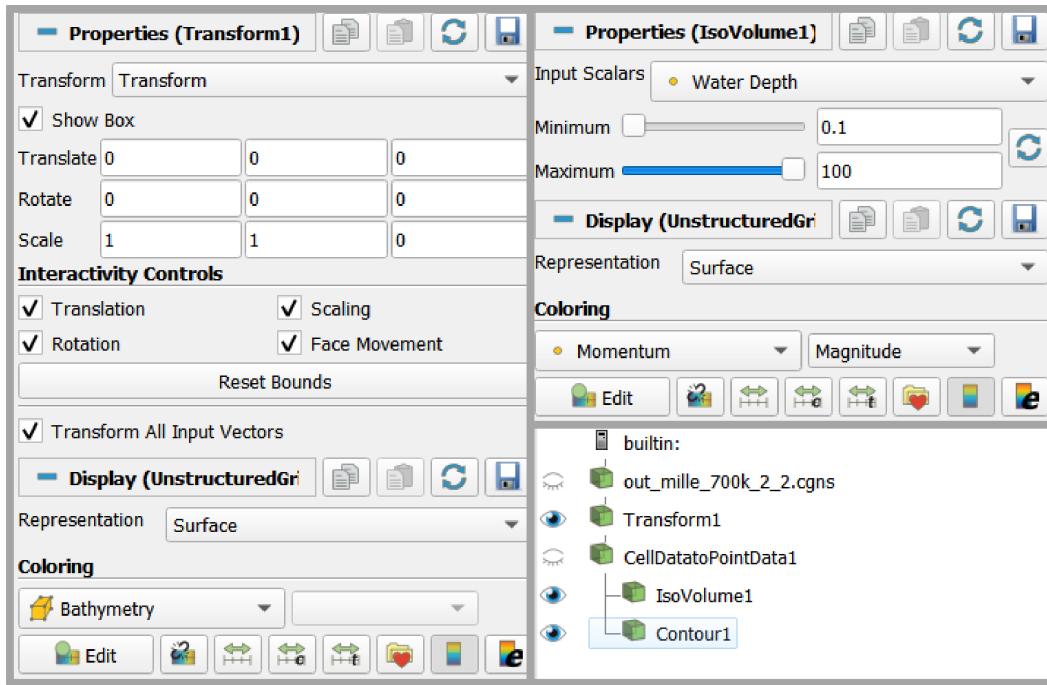


FIGURE 77 – Détail de *Transform*, *Isovolumne* et de l’arborescence.

Si on répète l’opération avec un ordre différent (ou un débit différent), on peut afficher les deux lignes d’inondation sur le même domaine. On obtient alors un résultat similaire à celui-ci (l’encadré en rouge est un zoom sur une zone représentative de la différence de hauteur d’eau) :

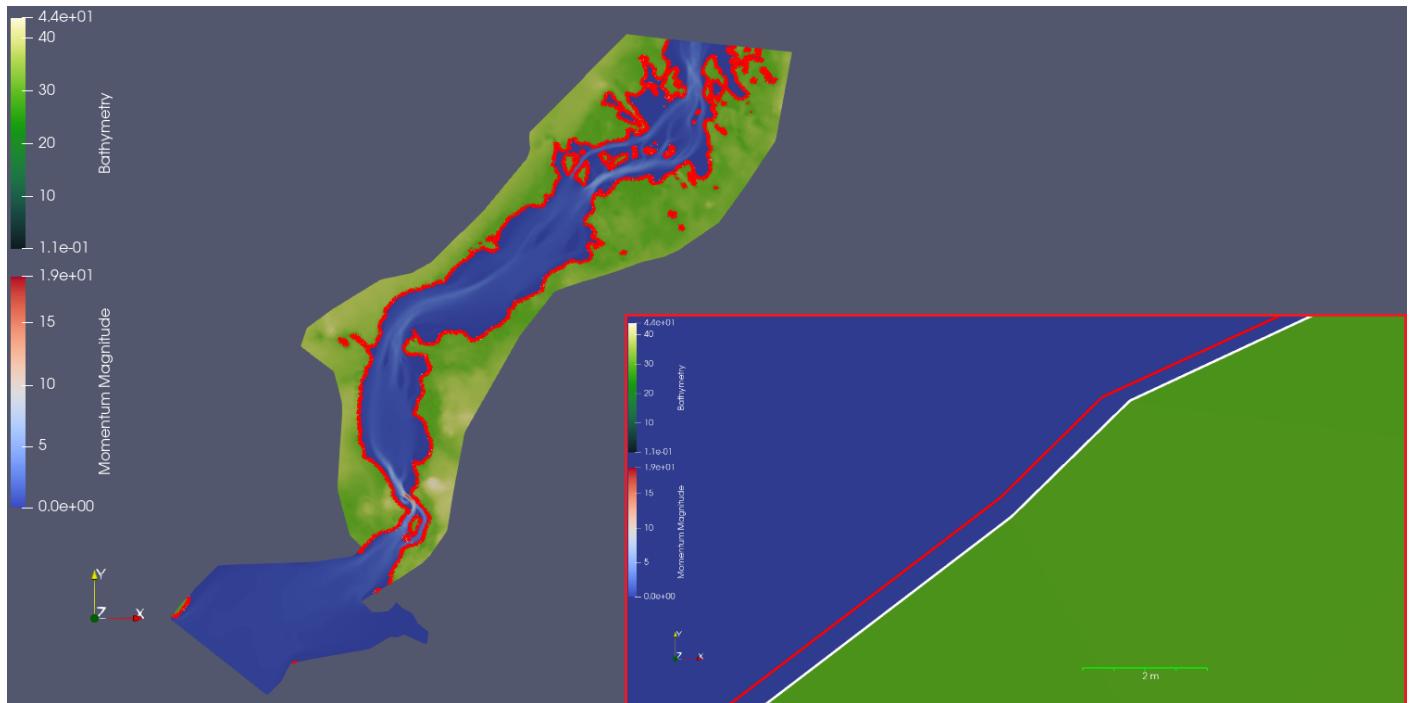


FIGURE 78 – Détail de *Isovolumne*, *Transform* et de l’arborescence.

5 Cas Tests

On présente dans cette section plusieurs cas tests qui servent à évaluer les capacités de *CuteFlow*. Chaque sous partie aura pour objectif la comparaison entre les solutions de différents ordres avec différentes conditions initiales. Les fichiers qui seront utilisés dans cette section se trouvent dans le dossier [CuteFlow/docs/](#).

5.1 Cas de bris de barrage fictifs

Dans un premier temps, nous allons utiliser le cas d'un bris de barrage fictif unidimensionnel. Les conditions initiales que nous allons utiliser pour les différents cas tests sont présentées dans le tableau 5.1. Les indices "L" et "R" font respectivement référence à la partie gauche et droite du domaine. Ces conditions initiales sont basées sur l'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 – Conditions initiales pour le cas d'un bris de barrage unidimensionnel

5.1.1 Solutions exactes et de référence

Les équations de Saint-Venant et leur résolution numérique utilisant une méthode des volumes finis explicite en temps sont présentés ci-dessous. Les notations utilisées sont présentées dans la figure 80. Cette section est basée sur la section 6 de l'article [4].

Les équations peuvent être écrites ainsi :

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

avec

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

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

où \bar{u} et \bar{v} sont vitesses moyennées selon la profondeur dans les directions x et y, h est la hauteur de la colonne d'eau comme définie dans la figure 80, et g est l'accélération gravitationnelle.

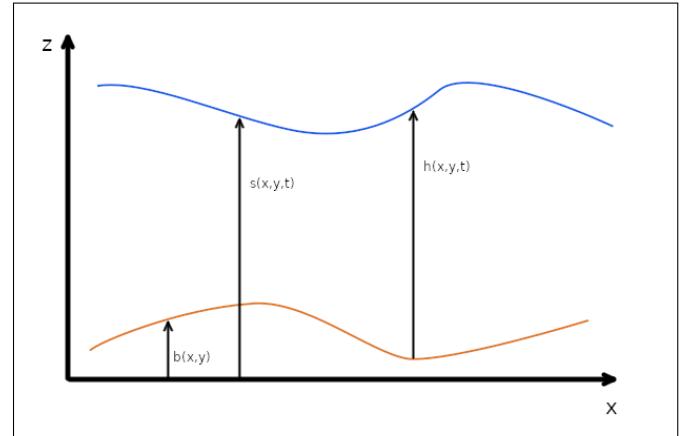


FIGURE 80 – Illustration des notations

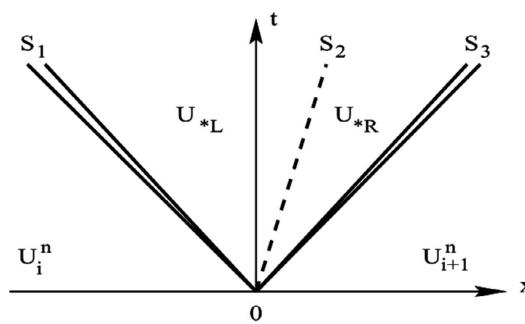


FIGURE 81 – Diagramme x-t pour la solution du problème de Riemann défini par les états gauche (U_i) et droit (U_j) ([2])

Lorsque c'est possible, la solution exacte est calculée via un code Python annexe, sinon, une solution de référence est générée en utilisant un fichier de maillage de 37 millions d'éléments.

5.1.2 Simulation sur mesh_6138 - Test 1

On simule un bris de barrage sur un domaine rectangulaire de 10 mètres de long avec une hauteur d'eau de 1m sur la gauche et un lit sec sur la droite.

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

On utilise ici un maillage contenant 6138 cellules triangulaires présenté en Figure 82. La Figure 5.1.2 présente l'évolution de la hauteur d'eau h et la Figure 5.1.2 l'évolution de la vitesse u . On profite du fait que nous disposons de la solution exacte pour ce test afin de la comparer aux autres solutions (ordre 1 et 2 sur maillage à 6138 éléments) et à la solution de référence (ordre 1 sur maillage à 37M d'éléments).

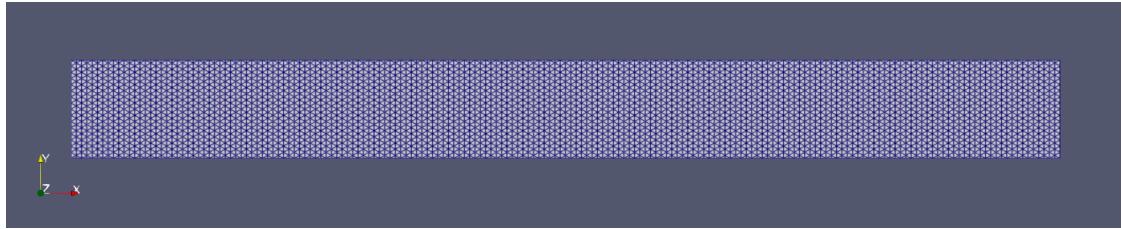


FIGURE 82 – Maillage de 6138 éléments

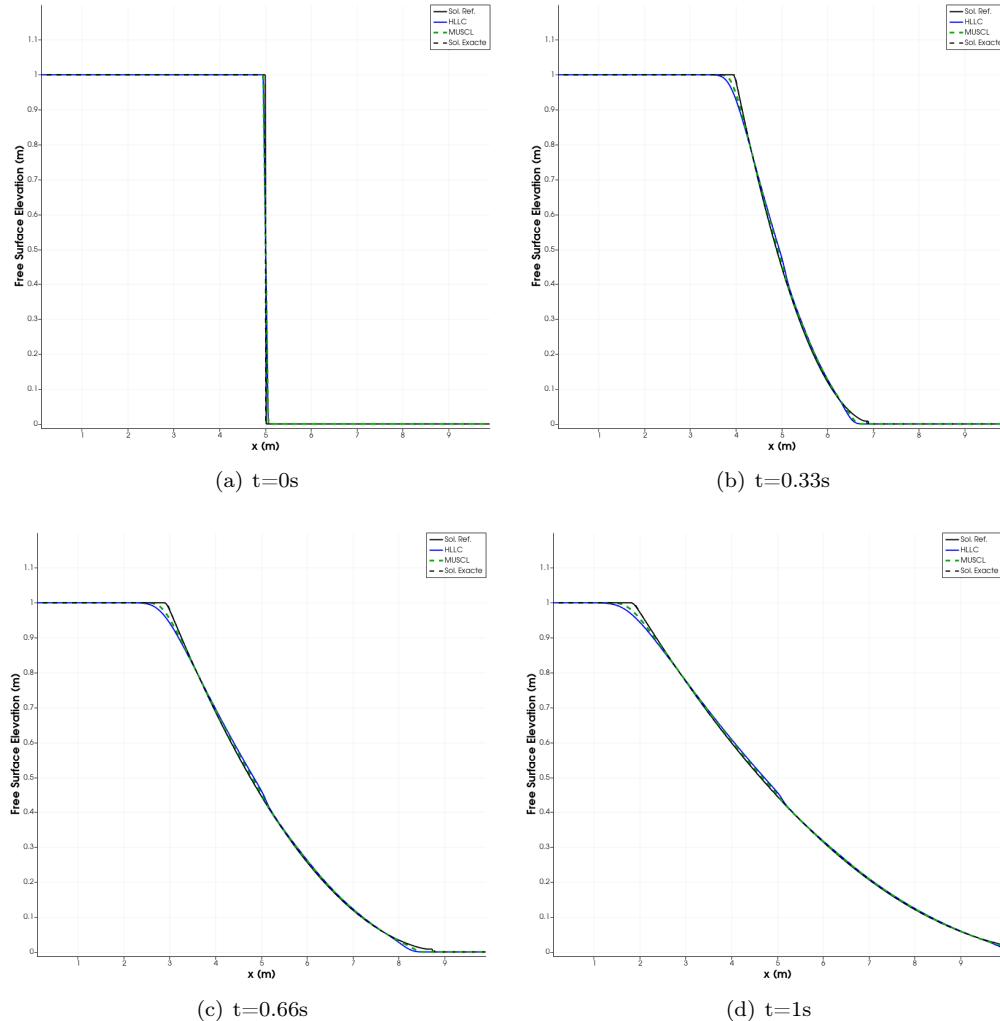


FIGURE 83 – Solutions projetées sur l'axe y pour le test 1, bris de barrage unidimensionnel sur un maillage à 6138 cellules, solution de référence sur un maillage à 37 millions de cellules à $t = 0, 0.33, 0.66$ et 1 secondes (Free Surface Elevation) et solution exacte.

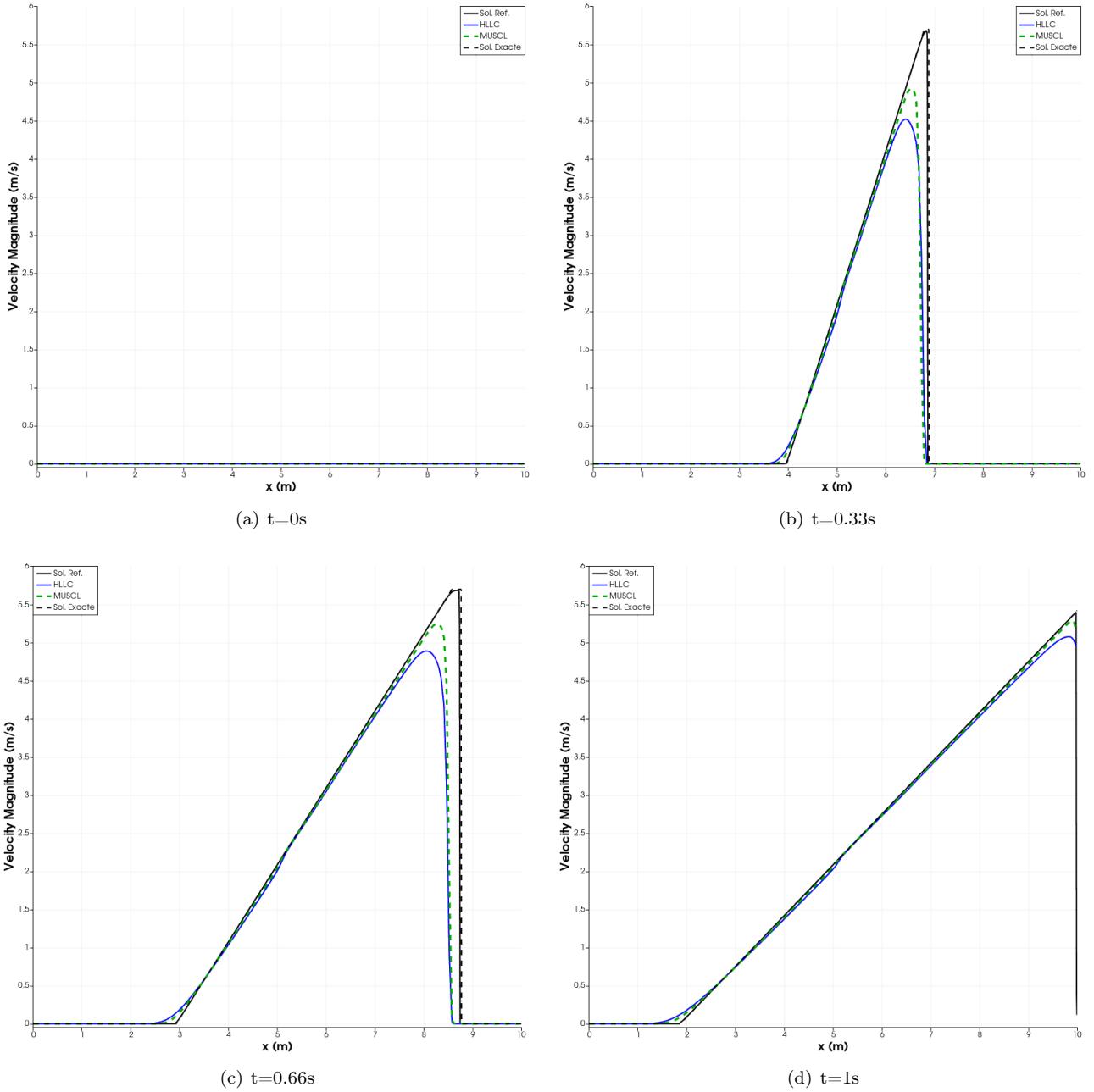


FIGURE 84 – Solutions projetées sur l’axe y pour le test 1, bris de barrage unidimensionnel sur un maillage à 6138 cellules, solution de référence sur un maillage à 37 millions de cellules à $t = 0, 0.33, 0.66$ et 1 secondes (Velocity magnitude) et solution exacte.

5.1.3 Simulation sur mesh_6138 - Test 2

Pour rappel, les conditions initiales de cette simulation sont les suivantes :

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

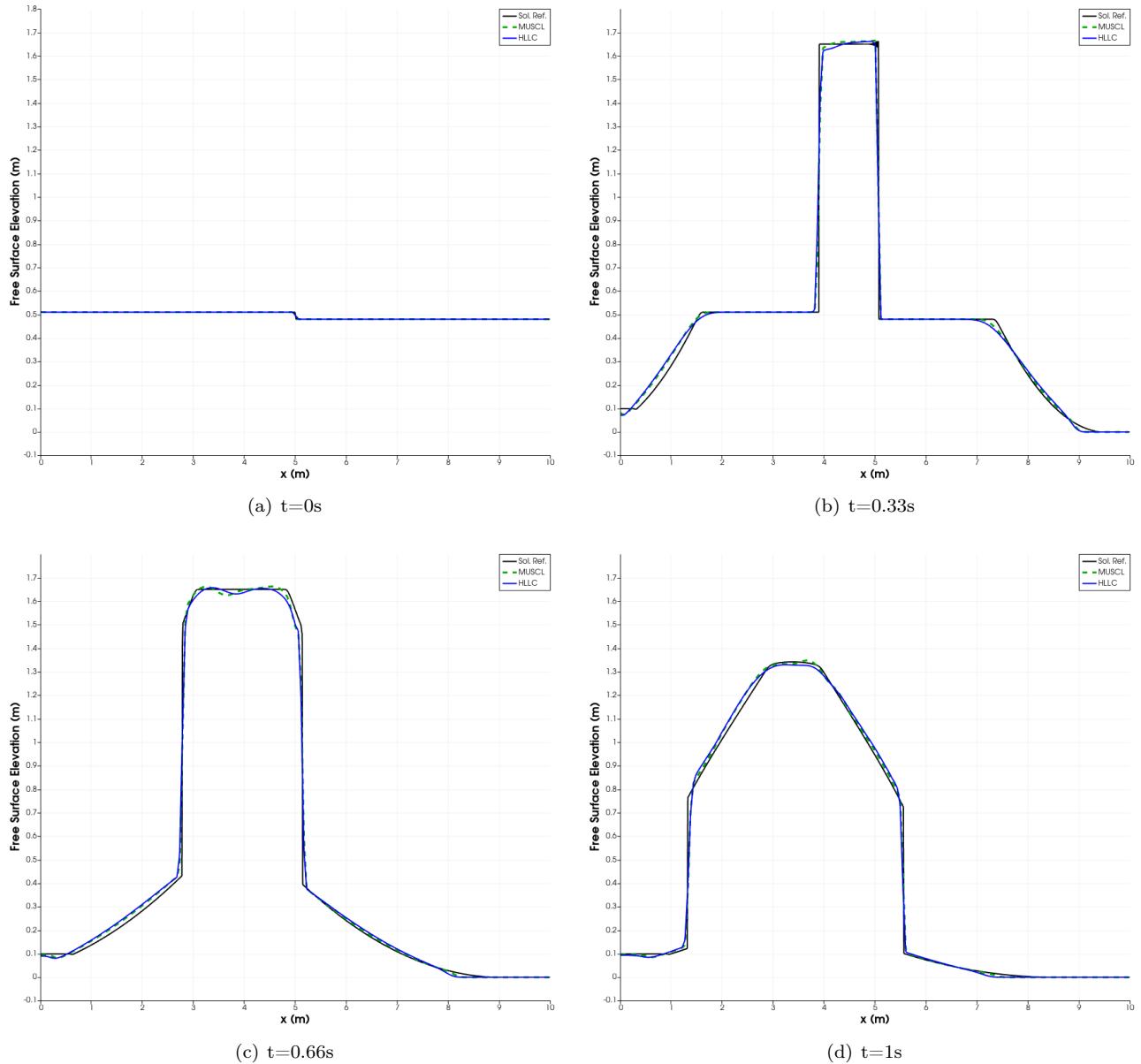


FIGURE 85 – Solutions projetées sur l'axe y pour le test 3, bris de barrage unidimensionnel sur un maillage à 6138 cellules et solution de référence sur un maillage à 37 millions de cellules à $t = 0, 0.33, 0.66$ et 1 secondes (Free Surface Elevation)

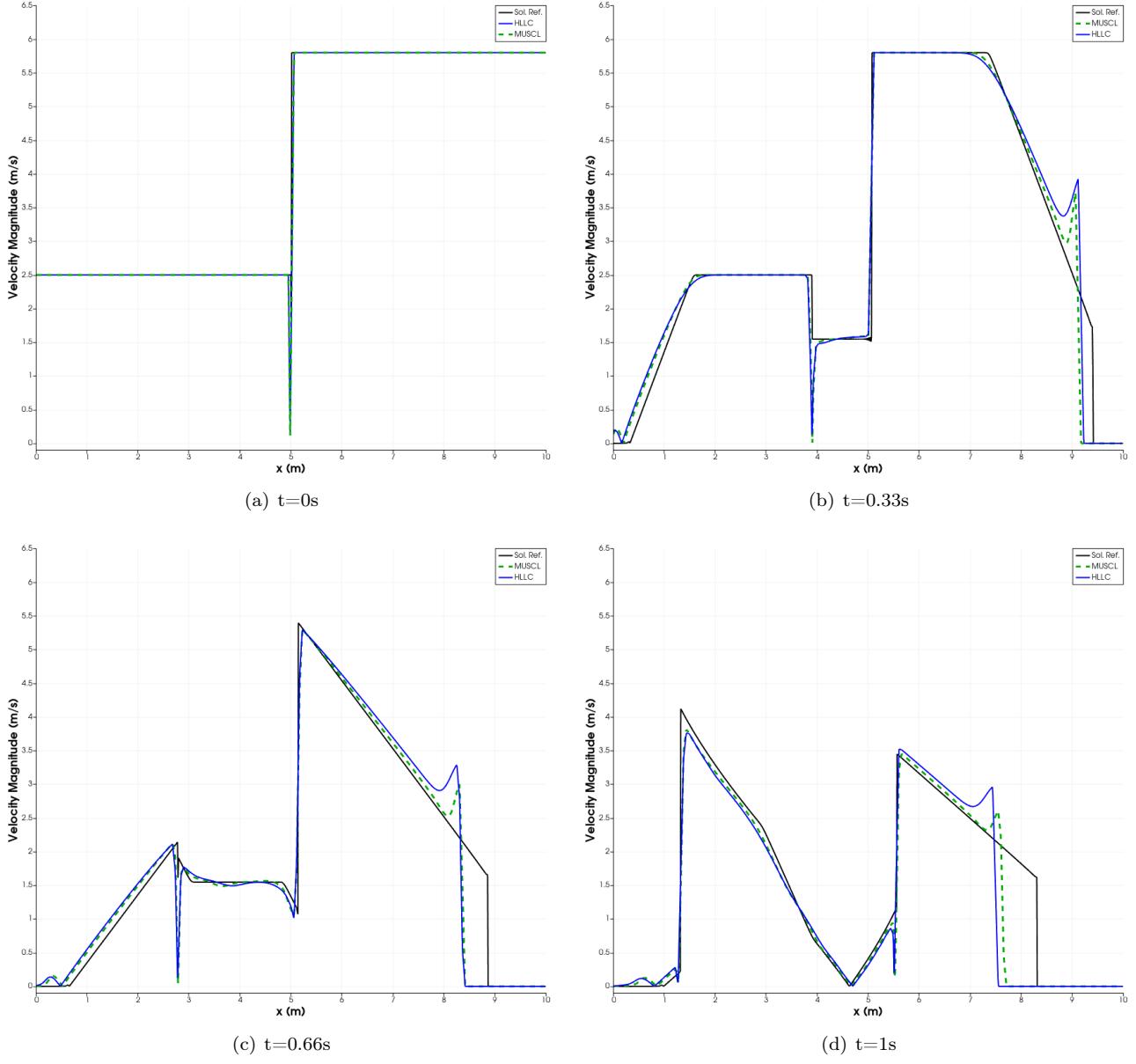


FIGURE 86 – Solutions projetées sur l’axe y pour le test 2, bris de barrage unidimensionnel sur un maillage à 6138 cellules et solution de référence sur un maillage à 37 millions de cellules à $t = 0, 0.33, 0.66$ et 1 secondes (Vitesse selon X)

5.1.4 Simulation sur mesh_6138 - Test 3

Pour rappel, les conditions initiales de cette simulation sont les suivantes :

$$\begin{cases} h = \begin{cases} 1 & \text{si } x < 0.5m \\ 1 & \text{si } x > 0.5m \end{cases} [m], \\ \bar{u} = \begin{cases} -3 & \text{si } x < 0.5m \\ 3 & \text{si } x > 0.5m \end{cases} [m/s]. \end{cases} \quad (2)$$

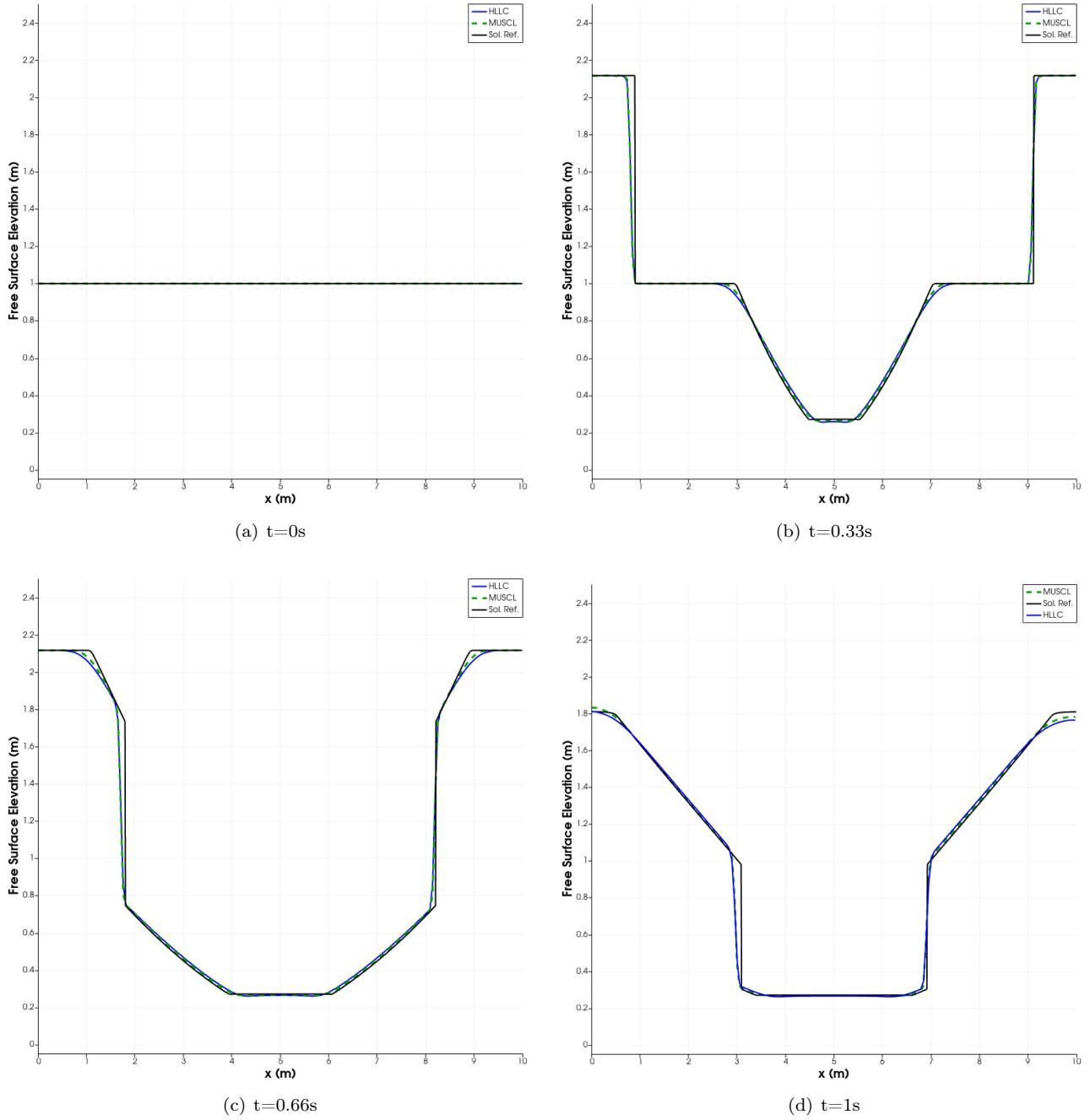


FIGURE 87 – Solutions projetées sur l'axe y pour le test 3, bris de barrage unidimensionnel sur un maillage à 6138 cellules et solution de référence sur un maillage à 37 millions de cellules à $t = 0, 0.33, 0.66$ et 1 secondes (Free Surface Elevation)

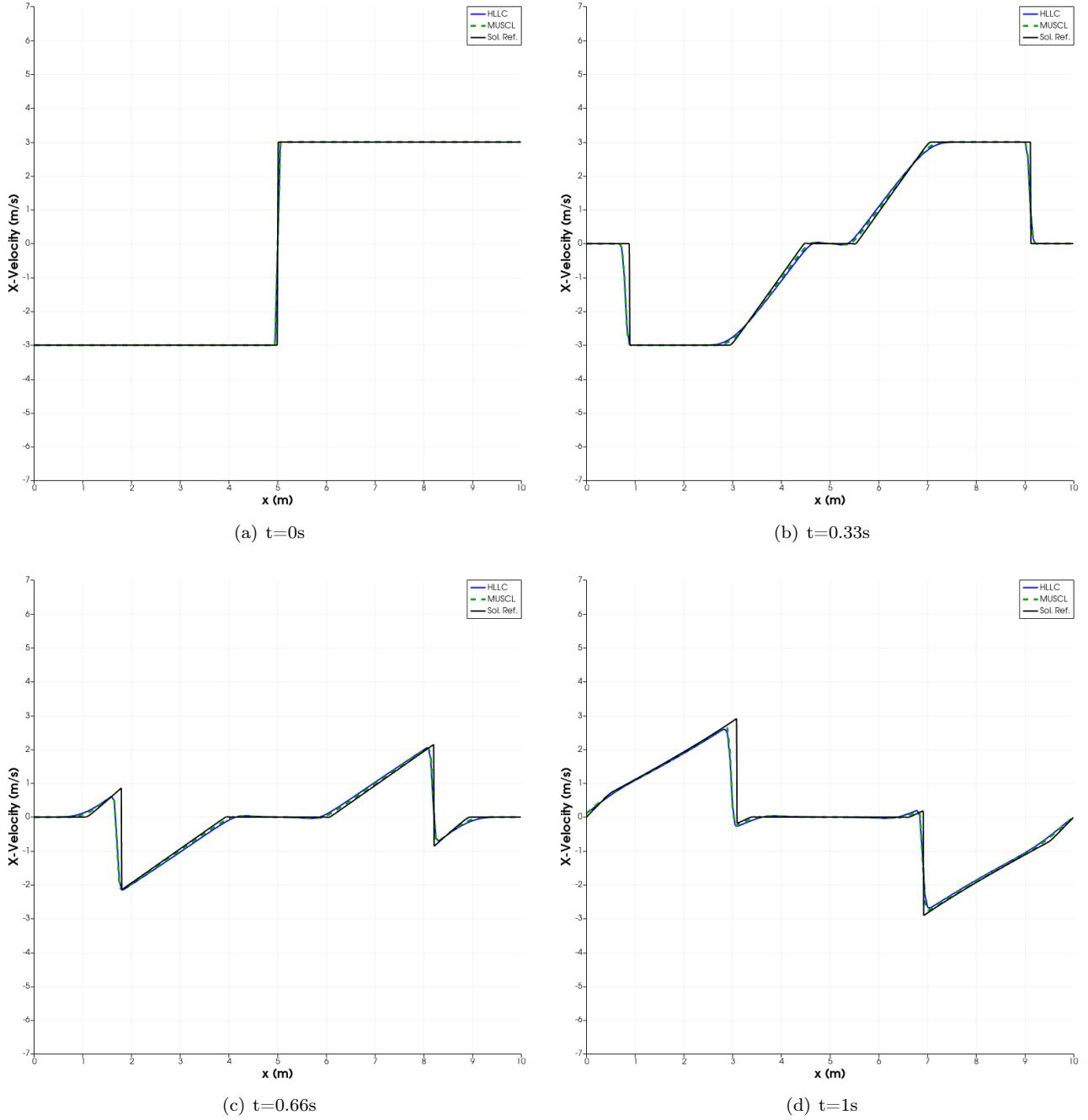


FIGURE 88 – Solutions projetées sur l'axe y pour le test 3, bris de barrage unidimensionnel sur un maillage à 6138 cellules et solution de référence sur un maillage à 37 millions de cellules à $t = 0, 0.33, 0.66$ et 1 secondes (Vitesse selon X)

5.2 Cas de bris de barrage fictif avec bump

On simule un bris de barrage sur un domaine rectangulaire de 10 mètres de long avec une hauteur d'eau de 1m sur la gauche et 0.1m sur la droite. Cette fois ci le domaine n'est pas plat mais comporte un "bump" sur la droite. La solution de référence est calculée sur un maillage de 37 millions de cellules.

$$\begin{cases} h = \begin{cases} 1 & \text{si } x < 4m \\ 0.1 & \text{si } x > 4m \end{cases} [m], \\ \bar{u} = 0 [m/s]. \end{cases} \quad (3)$$

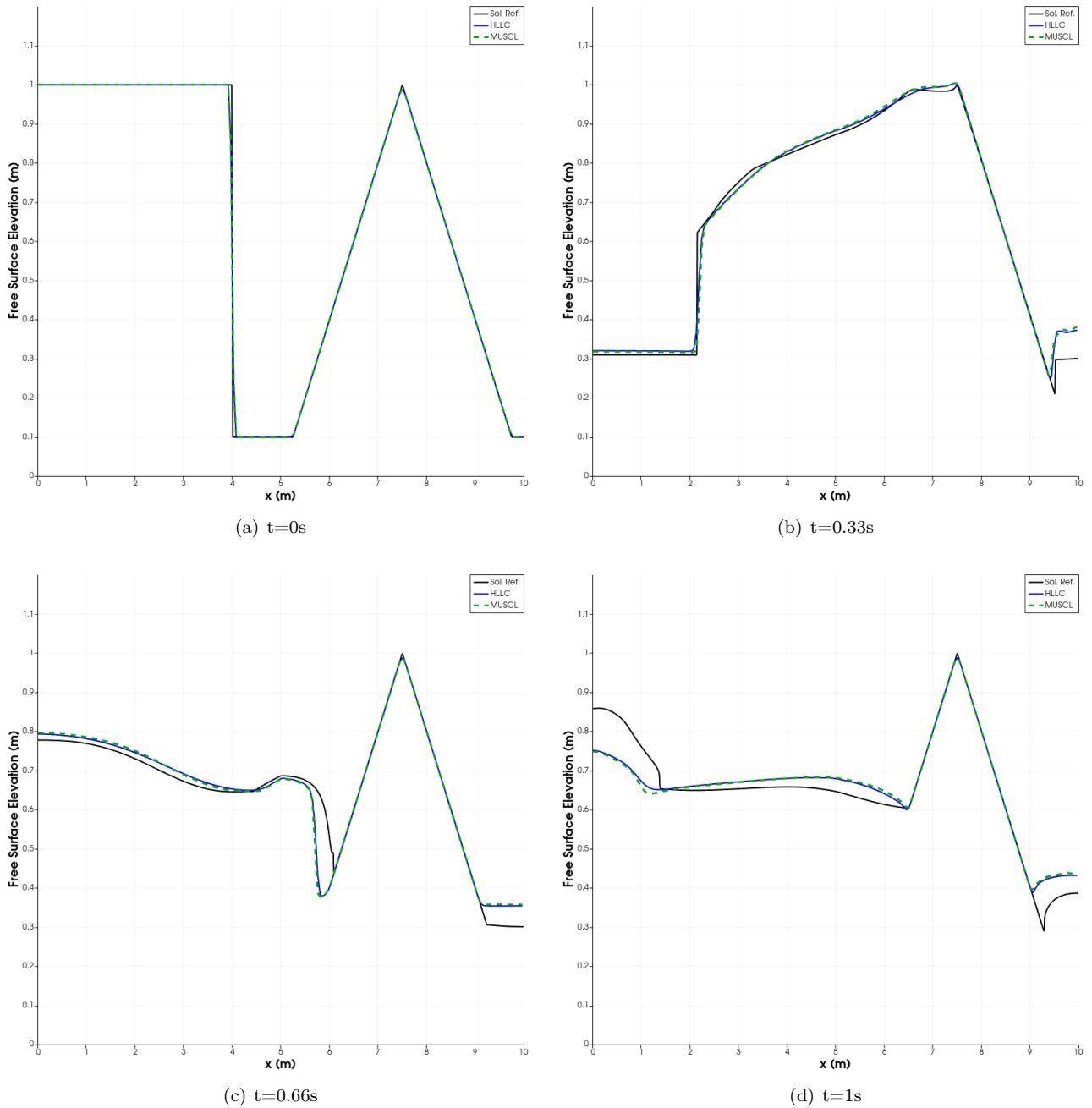


FIGURE 89 – Solutions projetées sur l'axe y pour le bris de barrage unidimensionnel sur un maillage à 6138 cellules et solution de référence sur un maillage à 37M de cellules à $t = 0, 0.33, 0.66$ et 1 secondes (FSE)

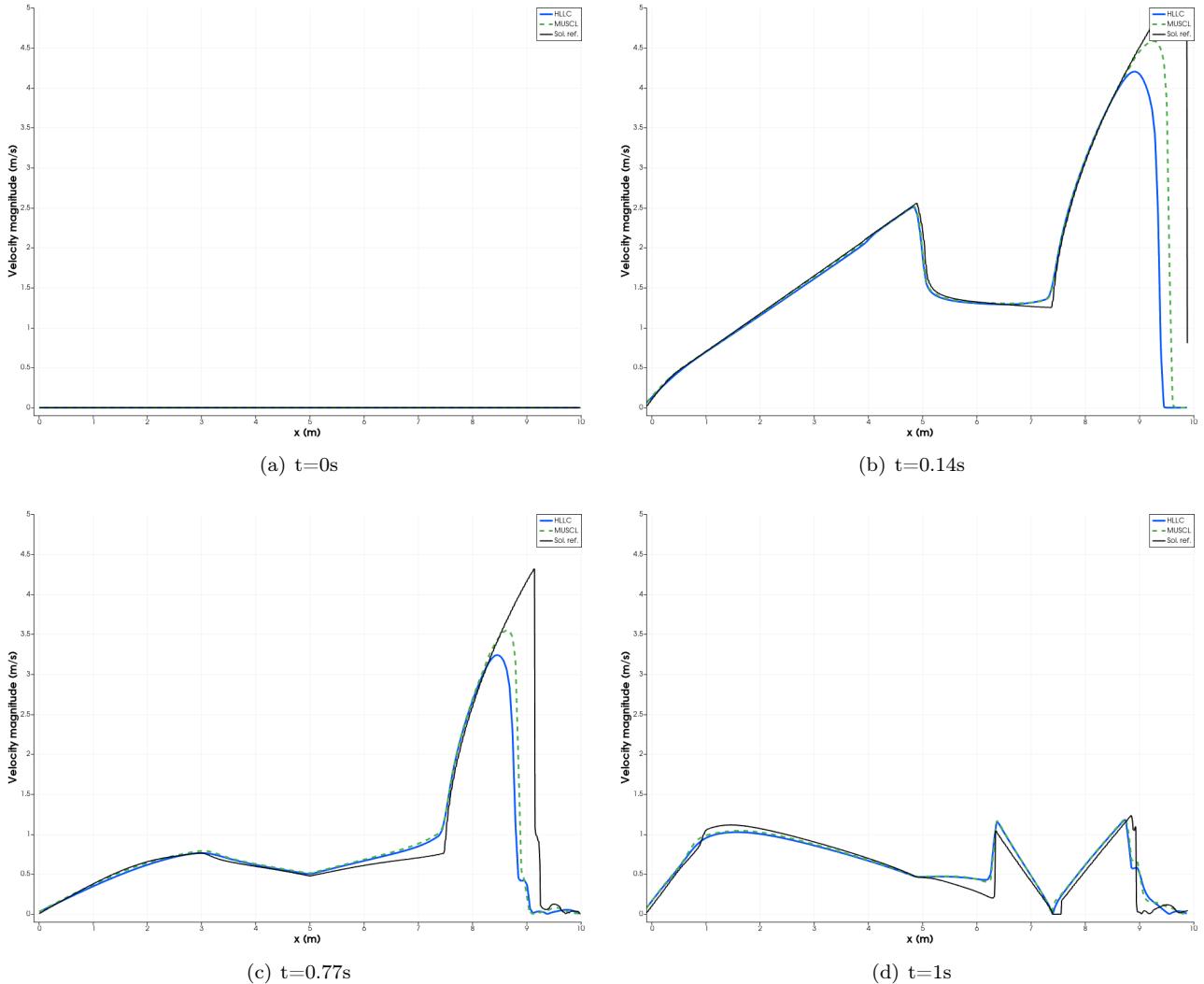


FIGURE 90 – Solutions projetées sur l’axe y pour le bris de barrage unidimensionnel sur un maillage à 6138 cellules et solution de référence sur un maillage à 95459 cellules à $t = 0, 0.14, 0.77$ et 1 secondes (Velocity Magnitude)

5.3 Cas du bris de barrage circulaire

Dans cette section on présente les résultats pour un bris de barrage circulaire. Le domaine est défini comme une surface carrée de 40 mètres de côté. Le fond est plat et la partie intérieure du cercle d'eau est d'une hauteur plus élevée que la partie extérieure à l'instant initial $t=0$ de tel que :

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

Les domaines "in" et "out" sont séparés par le cercle d'équation $r(x, y) = \sqrt{(x - 20)^2 + (y - 20)^2} = 2.5$, les coordonnées x et y correspondant aux cotés du domaine carré avec $0 < x < 40$ et $0 < y < 40$.

La figure suivante présente l'évolution de la hauteur d'eau sur un schéma d'ordre 2 avec un maillage de 90 000 éléments.

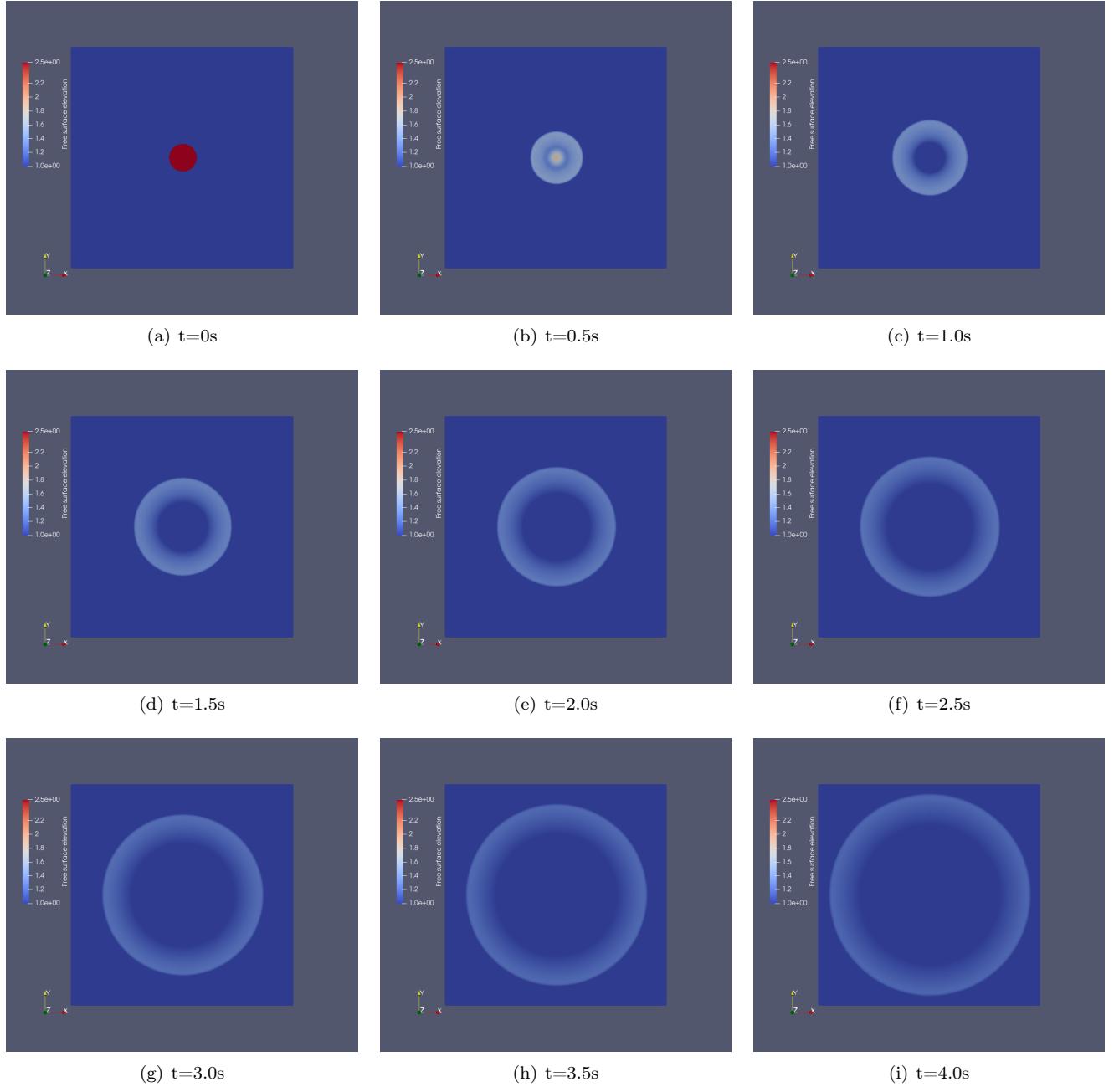


FIGURE 91 – Évolution de la hauteur d'eau (FSE)

On peut ensuite comparer les solutions entre différents ordres, ici sur un maillage de 90 000 éléments, la solution est tracée sur une droite parallèle aux cotés traversant le domaine d'un bord à l'autre en passant par son centre.

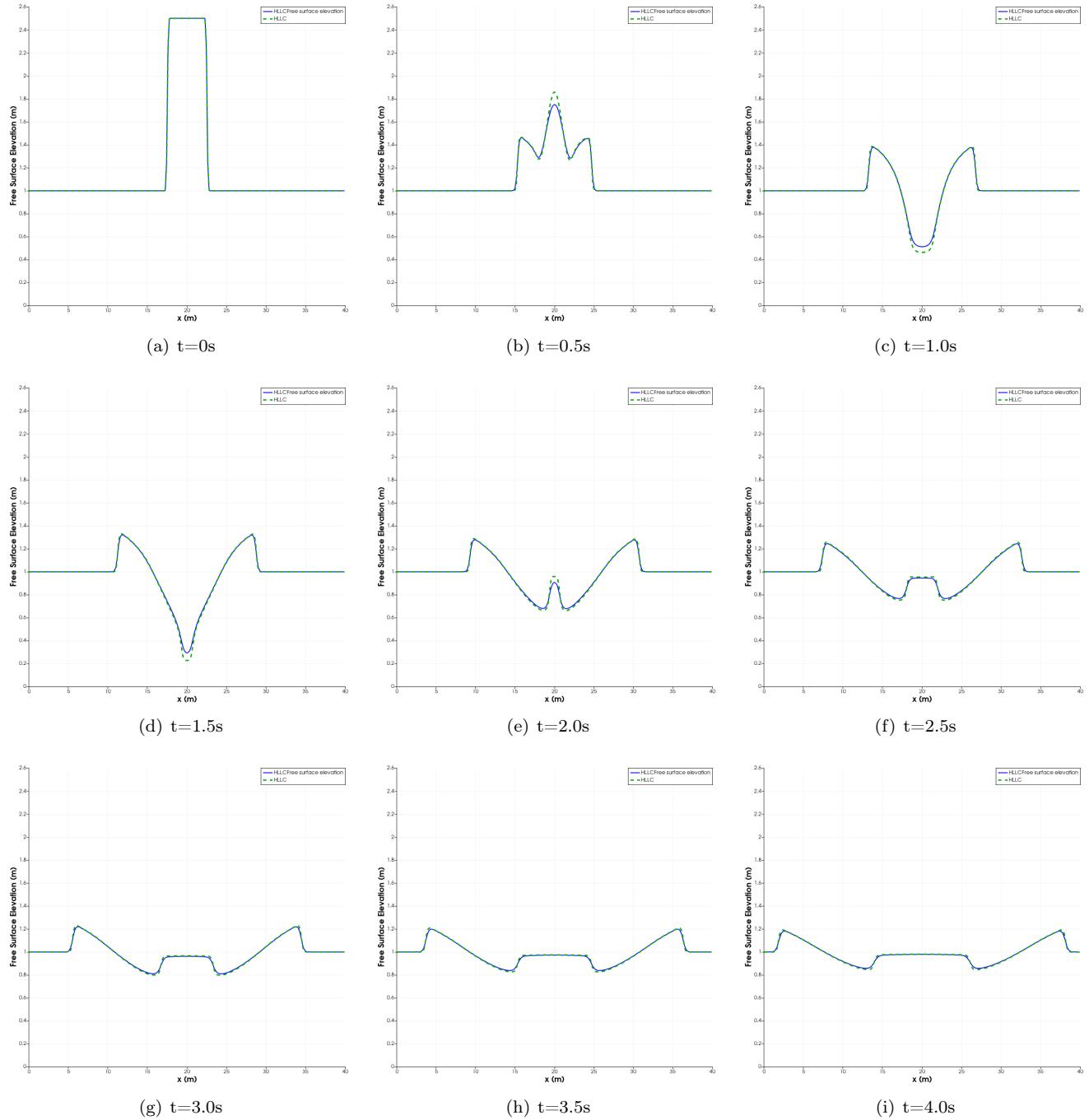


FIGURE 92 – Évolution de la hauteur d'eau h

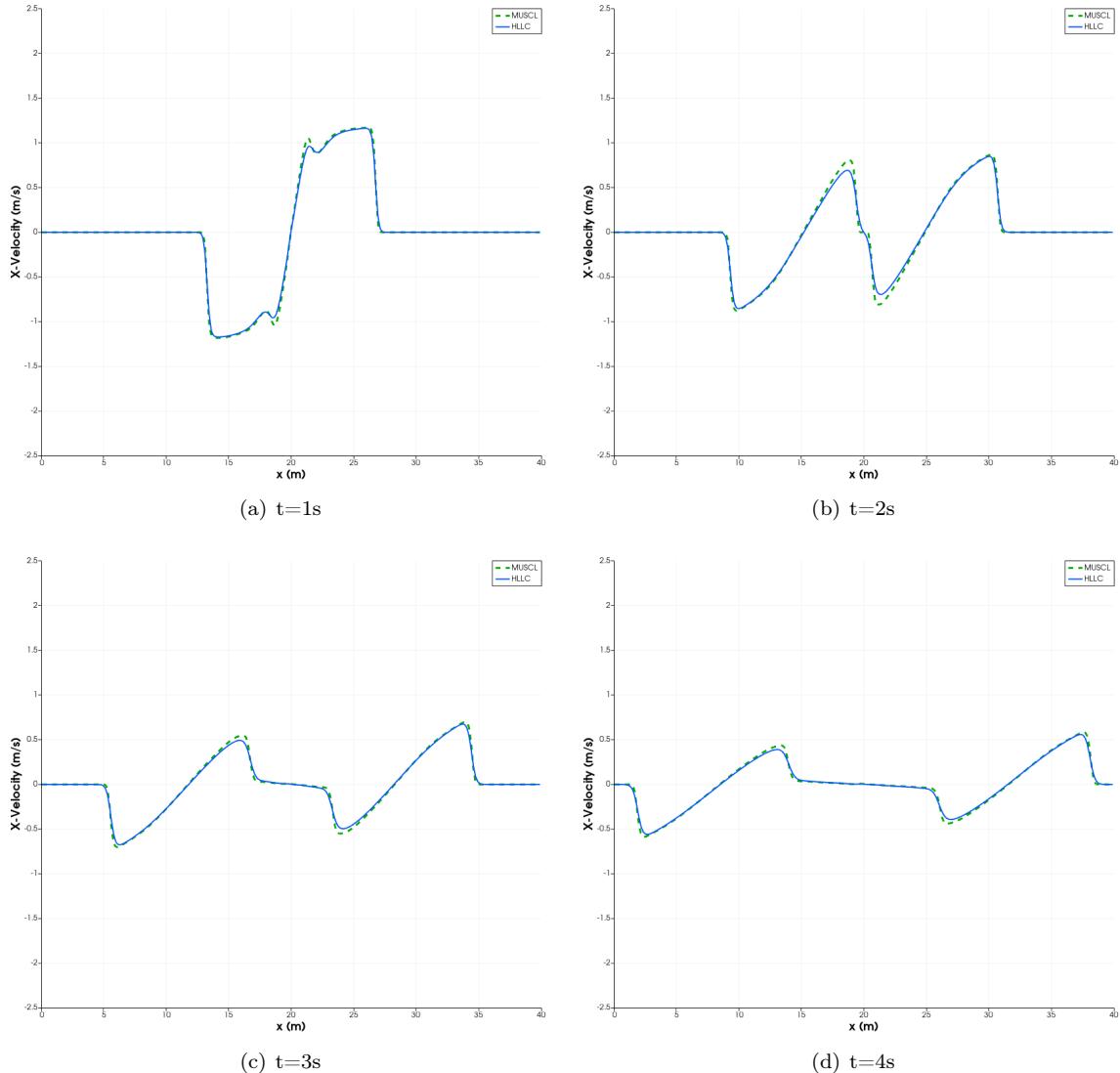


FIGURE 93 – Comparaison entre les ordres 1 et 2 à $t = 1, 2, 3$ et 4 secondes sur un maillage à 90k éléments (Vitesse selon X)

Enfin on compare les résultats obtenus avec différents maillages qui sont respectivement de 9546, 93031 et 947939 éléments.

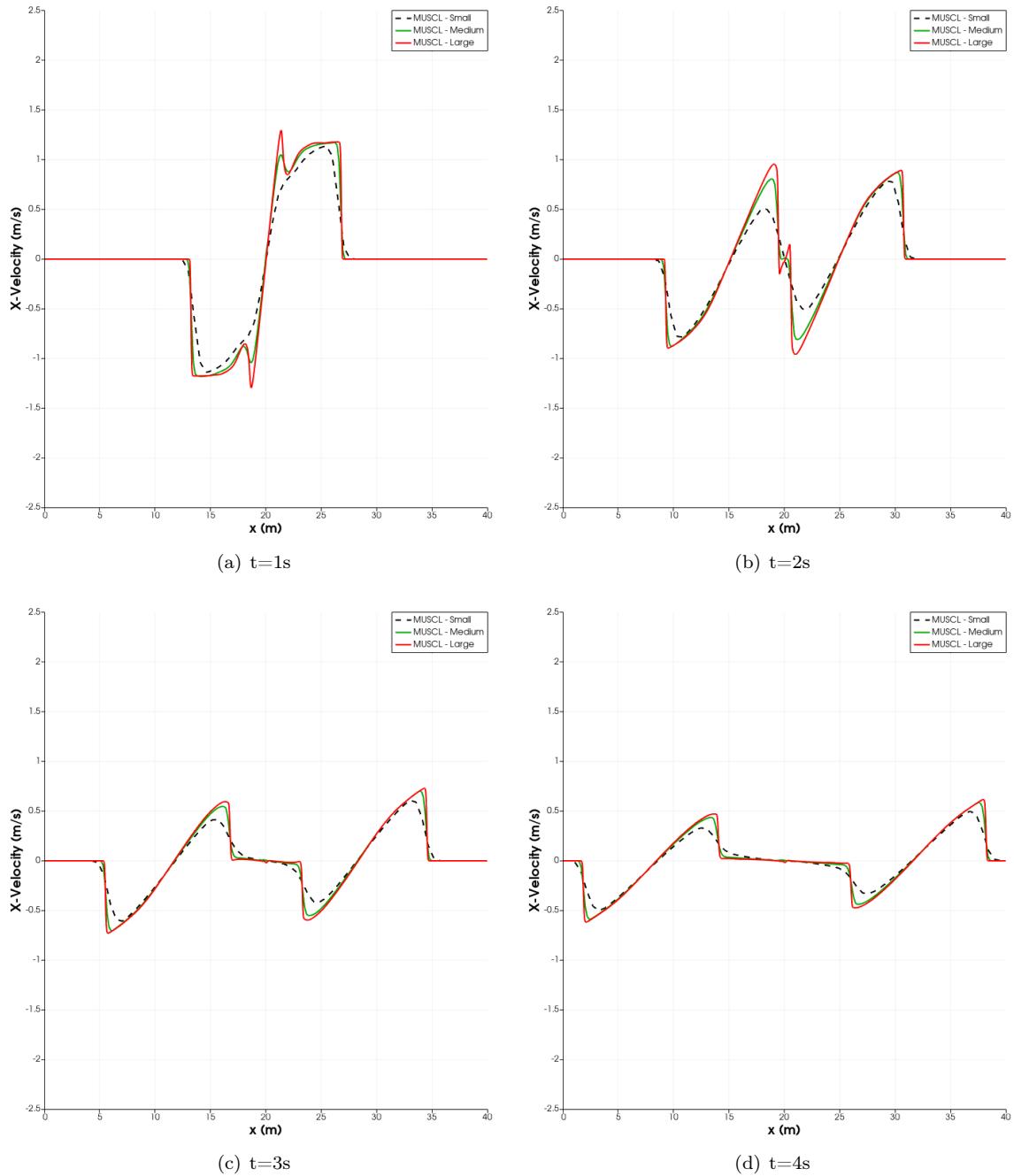


FIGURE 94 – Comparaison entre les différents maillages à $t = 0, 1, 2$ et 4 secondes (Vitesse selon X)

5.4 Cas de la rivière des Milles Îles

Dans cette section on compare les solutions d'ordre 1 et 2 sur le cas de la rivière des Milles Îles. On utilise un maillage modélisant la section de la rivière avec 700 000 éléments.

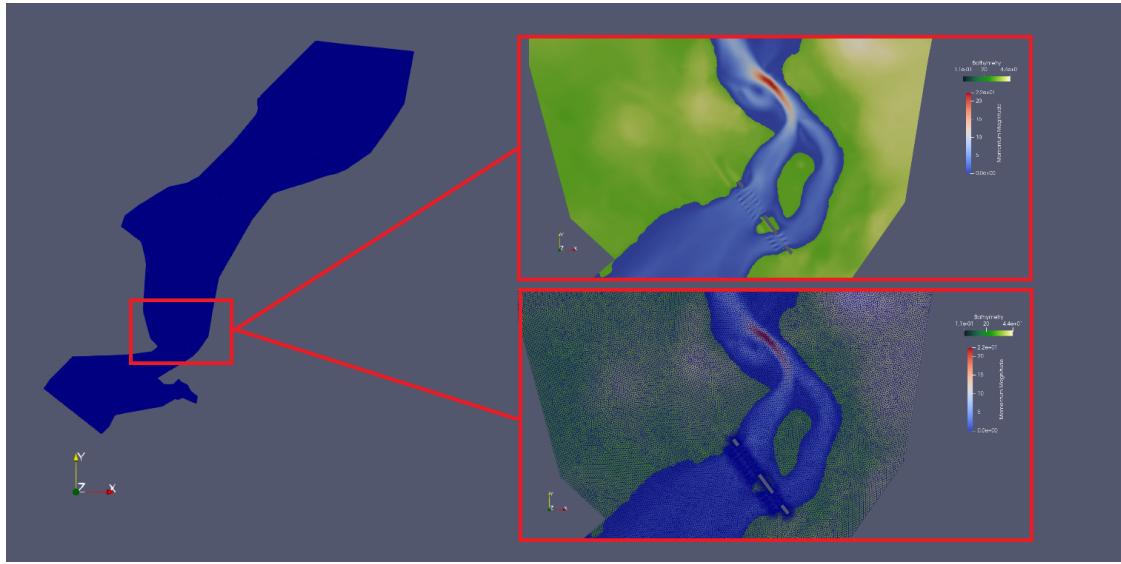


FIGURE 95 – Maillage de 700k éléments les Milles Îles

La solution présentée en Figure 96 est projetée le long d'une ligne suivant le cours de la rivière. Les lignes d'innondations calculées par les deux méthodes sont comparées sur la Figure 97.

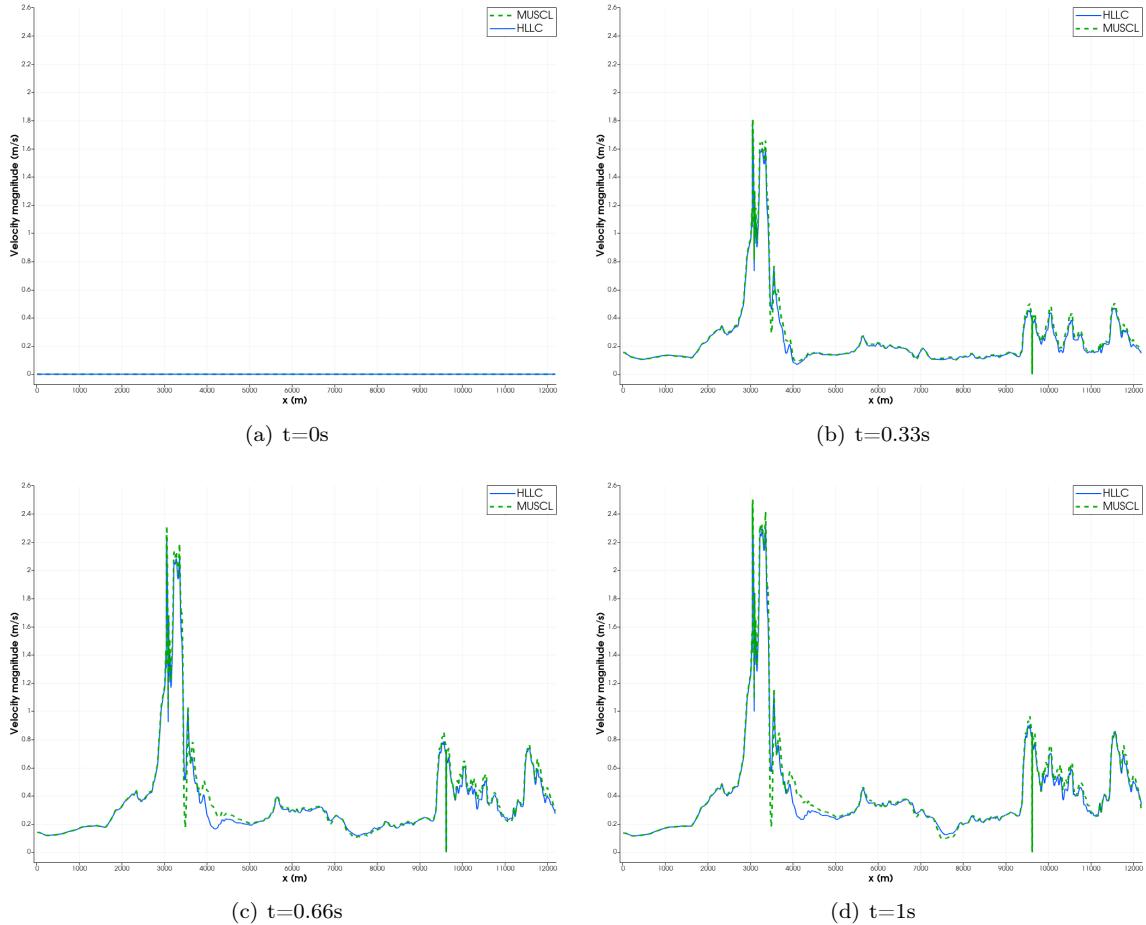


FIGURE 96 – Solutions projetées le long d'une ligne pour la rivière des Milles Îles sur un maillage à 700k cellules à $t = 0, 3300, 6600$ et 10000 secondes

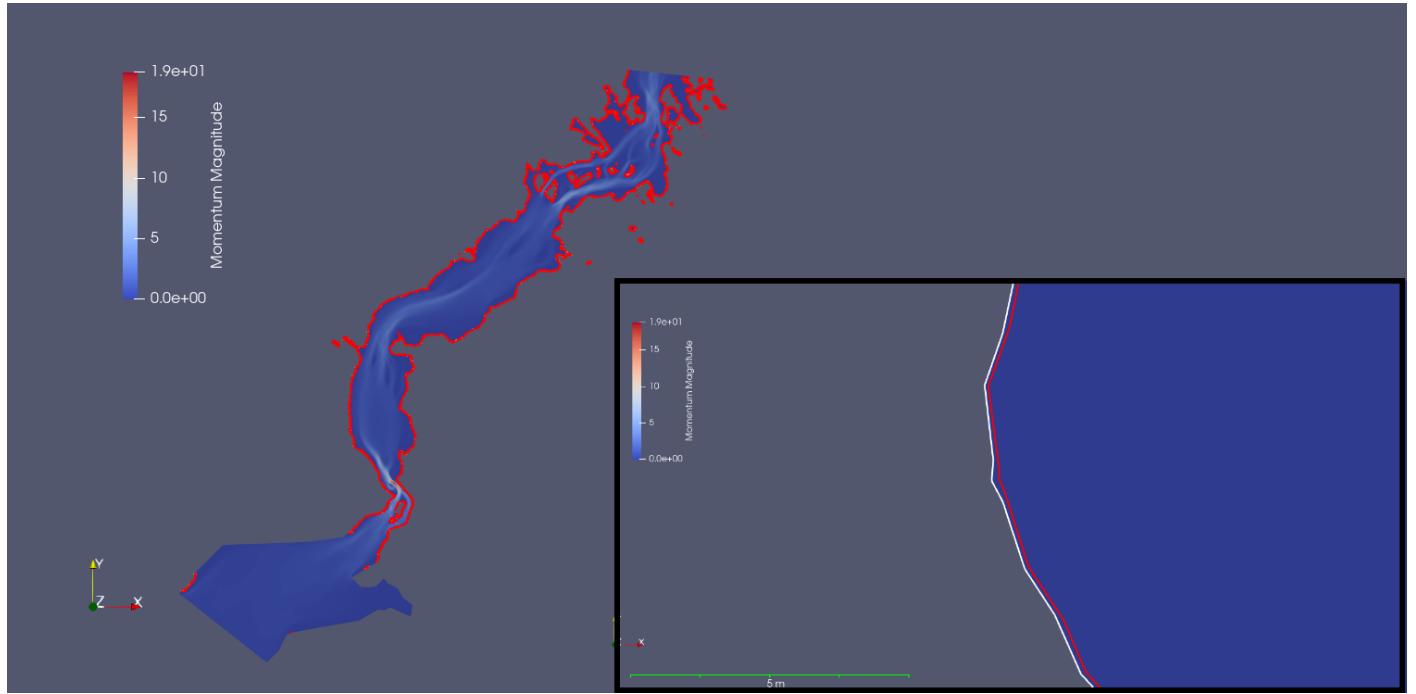


FIGURE 97 – Écart de hauteur d'eau entre les méthodes HLLC (en blanc) et MUSCL (en rouge) à 10000 secondes sur une interface mouillé/sec représentatif en aval de la rivière des milles Îles

On souhaite maintenant réaliser une comparaison entre les solutions pour un changement du débit d'entrée (sans changer le niveau de sortie). La Figure 99 présente les différences de ligne d'inondation pour trois débits dans la rivière des milles Îles : $600 \text{ m}^3.\text{s}^{-1}$, $1000 \text{ m}^3.\text{s}^{-1}$ et $1400 \text{ m}^3.\text{s}^{-1}$. Les lignes d'inondations dues à ces débits sont respectivement de couleur rouge, blanche et bleu. Les solutions sont calculées avec la méthode MUSCL et la zone d'étude est présentée dans le Figure 98.

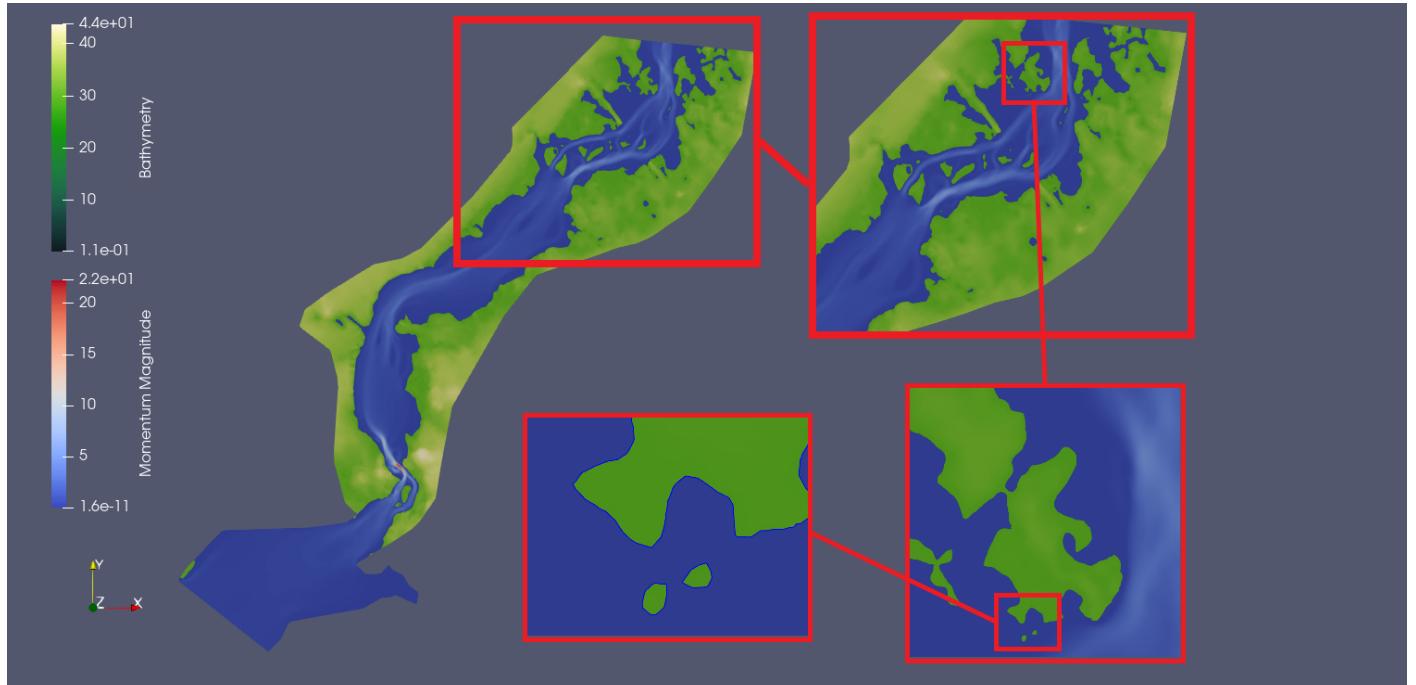


FIGURE 98 – Localisation de la zone pour l'analyse

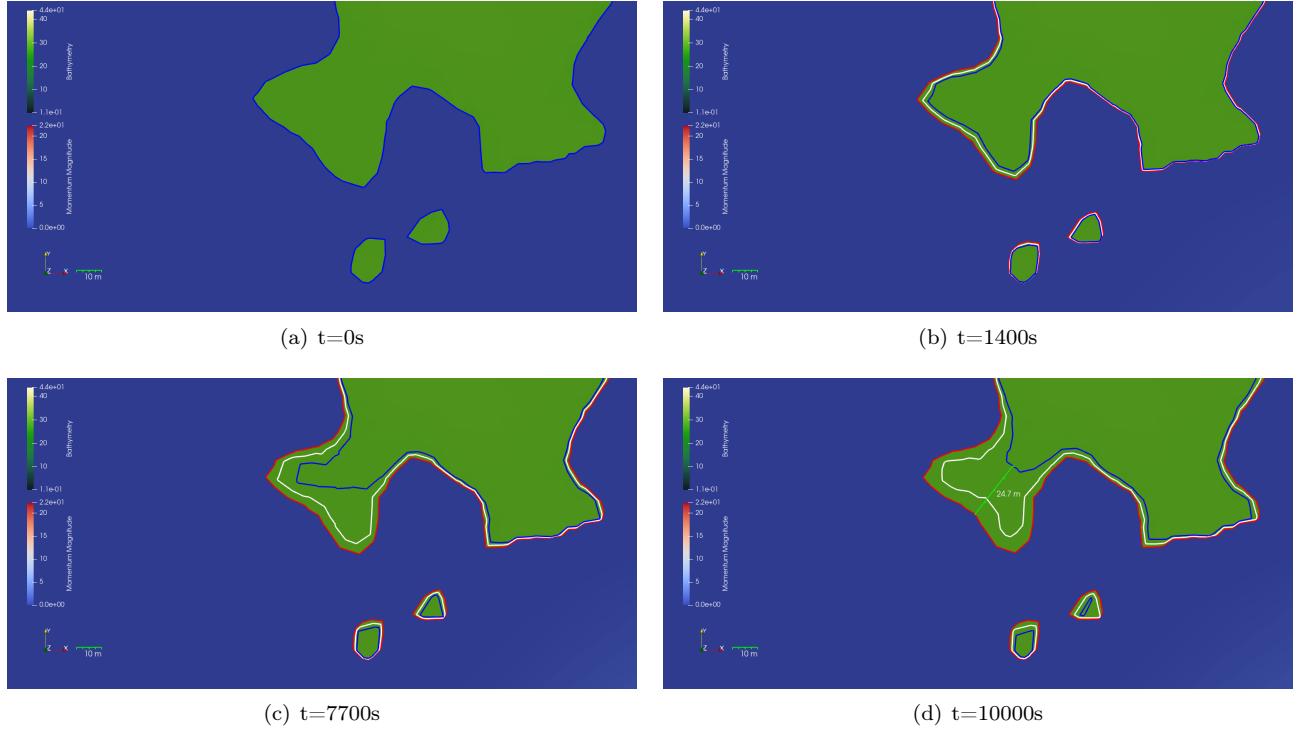


FIGURE 99 – Écart de hauteur d'eau entre les trois différents débits à $t = 0, 3300, 6600$ et 10000 secondes sur une zone aval de la rivière des milles Îles

5.5 Cas de l'archipel de Montréal

Dans cette section, on étudie le domaine l'archipel de Montréal. Les données utilisées dans cette section peuvent être trouvées dans le dossier [CuteFlow/docs/Archipel_de_Montreal](#). Ce domaine compte 7 entrées et les conditions initiales consistent à donner un débit initial à chacune de ces entrées ainsi qu'une hauteur d'eau de sortie.

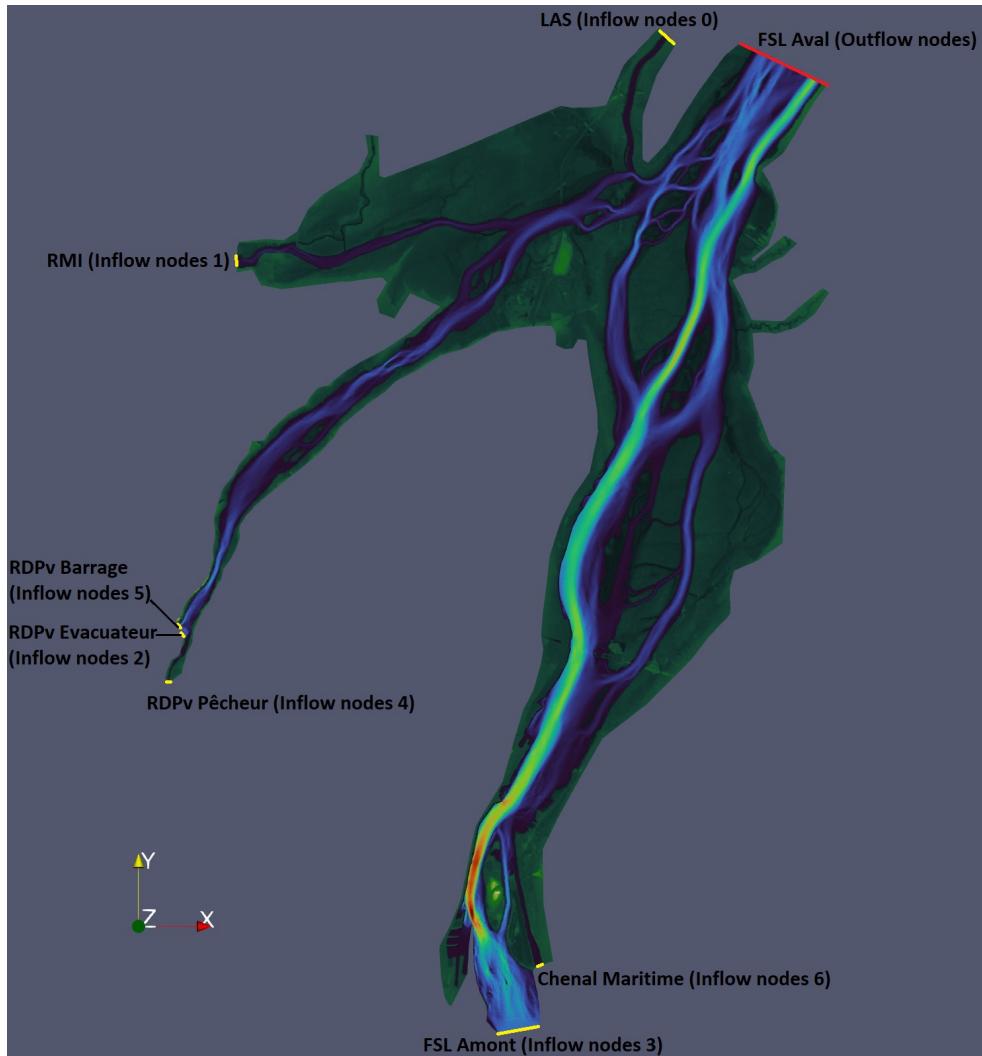


FIGURE 100 – Dénomination des entrées du domaine de l'archipel de Montréal

Dans un premier temps, on comparera les résultats obtenus sur un maillage avec un nombre de Manning constant à des mesures réelles effectuées dans le fleuve (voir section 5.5.1). Ensuite, les résultats de ces comparaisons seront utilisés pour générer un maillage avec des nombres de Manning variables dans le but de générer une solution qui s'approche au mieux de la réalité (voir section 5.5.2).

5.5.1 Manning constant

5.5.1.1 Simulation archi_0160

Le premier cas pour lequel nous avons des mesures réelles, archi_0160, utilise une hauteur d'eau de sortie de 5,93 m avec les débits d'entrées du Tableau 101.

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

FIGURE 101 – Débits d'entrées pour le cas archi_0160

Les solutions d'ordre 1 et 2 projetées sur une ligne d'eau dans la Rivière-Des-Prairies (voir Figure 102) sont présentées sur la Figure 103.

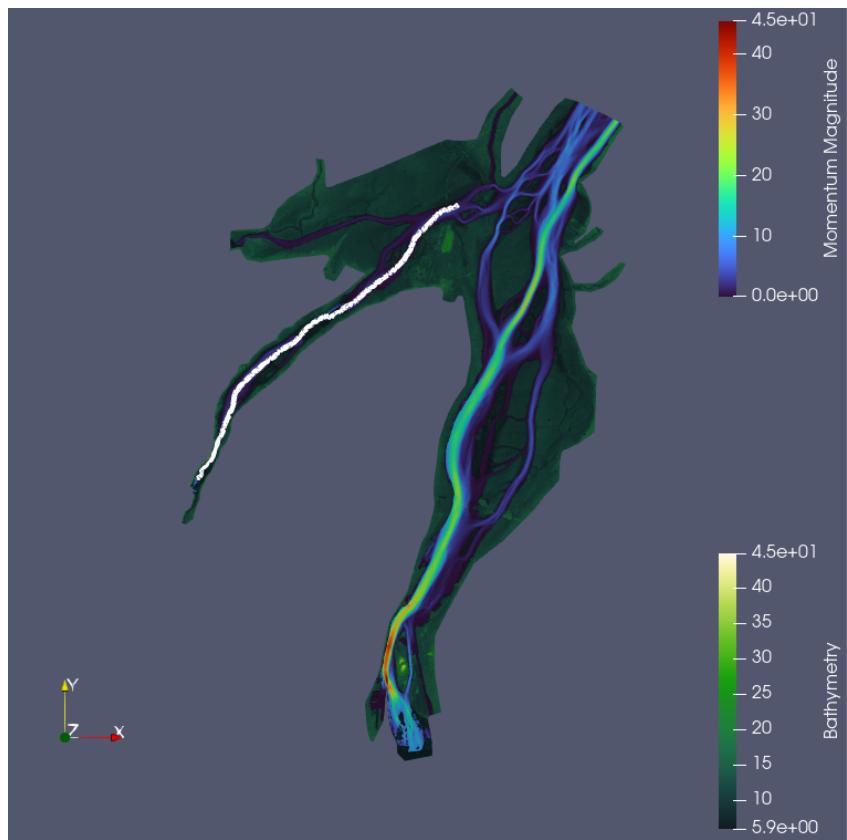


FIGURE 102 – archi_0160 - Ligne pour affichage de la solution - Rivière-Des-Prairies

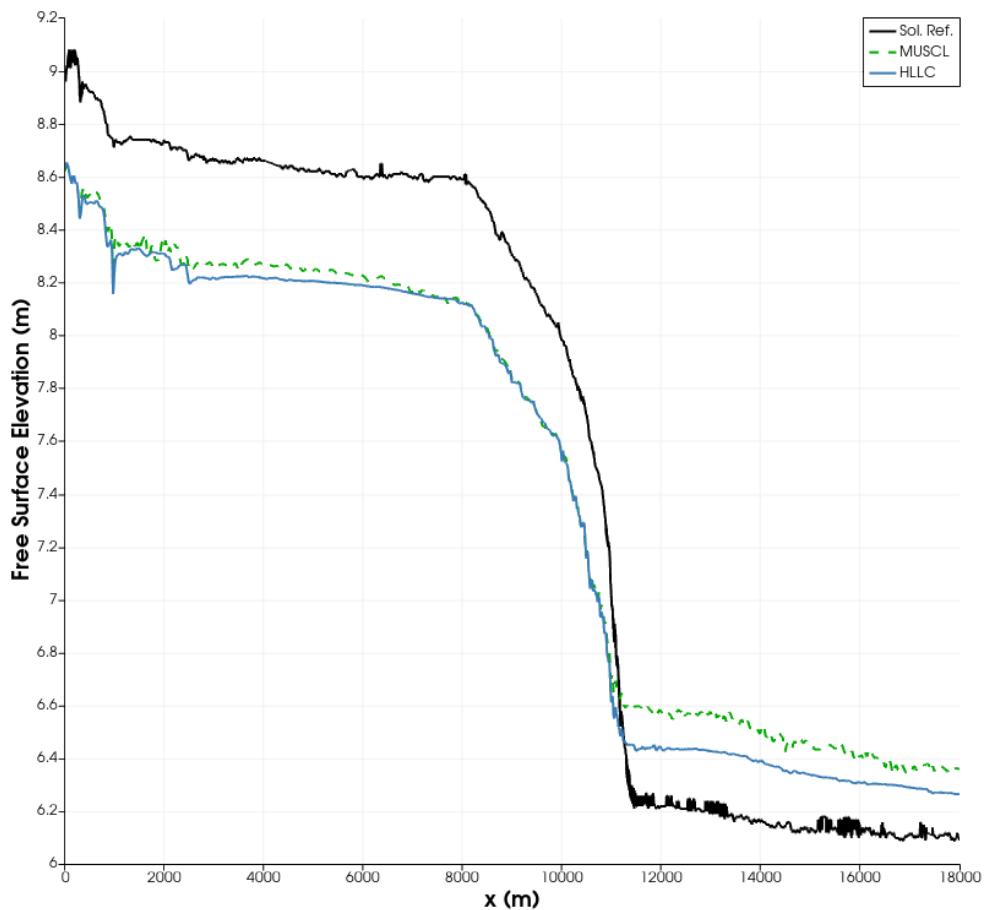


FIGURE 103 – archi_0160 - Écart entre les ordres 1 et 2 sur le maillage à Manning constant et les relevés à t = 80 000 secondes sur une ligne dans la Rivière-Des-Prairies (102) (Free Surface Elevation)

5.5.1.2 Simulation archi_0164

Pour le cas archi_0164, une hauteur d'eau de sortie de 7.05 m est utilisée avec les débits d'entrées présentés dans le Tableau 104.

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

FIGURE 104 – Débits d'entrées pour le cas archi_0164

Les solutions d'ordre 1 et 2 projetées sur une ligne d'eau dans la Rivière-Des-Prairies (voir Figure 102) sont présentées sur la Figure 105.

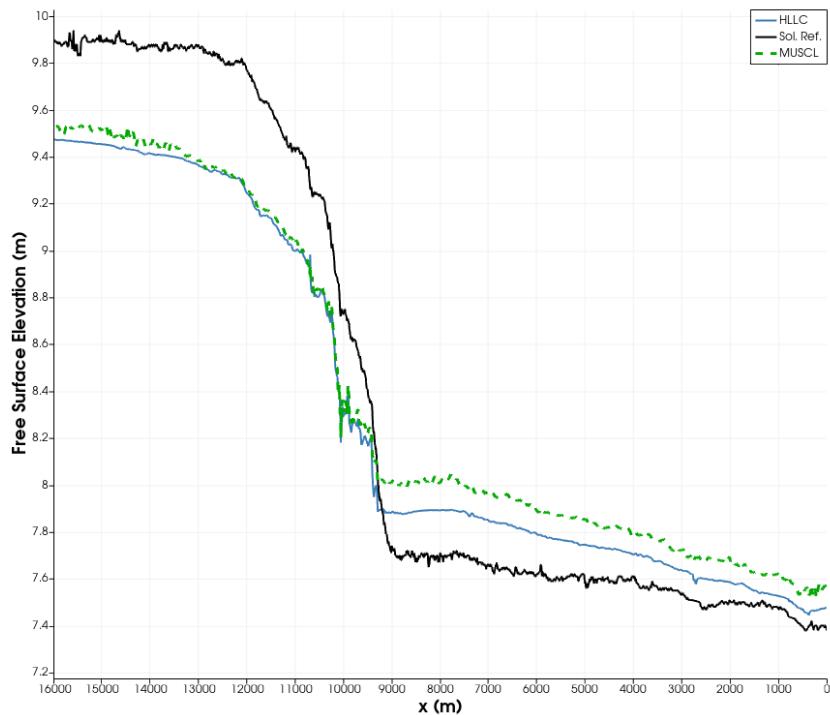


FIGURE 105 – archi_0164 - Écart entre les ordres 1 et 2 sur le maillage à Manning constant et les relevés à $t = 40\ 000$ secondes sur une ligne traversant la Rivière-Des-Prairies (102) (Free Surface Elevation)

5.5.2 Manning variable

Pour tenter d'améliorer la fiabilité des résultats, on souhaite modifier le nombre de Manning représentant les frottements au fond de la rivière. Dans la section précédente, les deux simulations utilisaient un nombre de Manning constant dans tout le domaine. On va maintenant utiliser un nombre de Manning différent pour chaque maille du domaine. De cette manière, on va pouvoir améliorer les résultats en augmentant le nombre de Manning dans les zones où la hauteur d'eau est trop faible et en le diminuant dans les zones où elle est trop haute.

La section 2.4 présente la marche à suivre afin de changer les nombres de Manning du maillage. On utilise ici 25 points dans le maillage (voir Figure 106(a)) auxquels on va attribuer un nombre de Manning bien particulier. Après plusieurs tests, on arrive à une carte des nombres de Manning présentée sur la Figure 106(b).

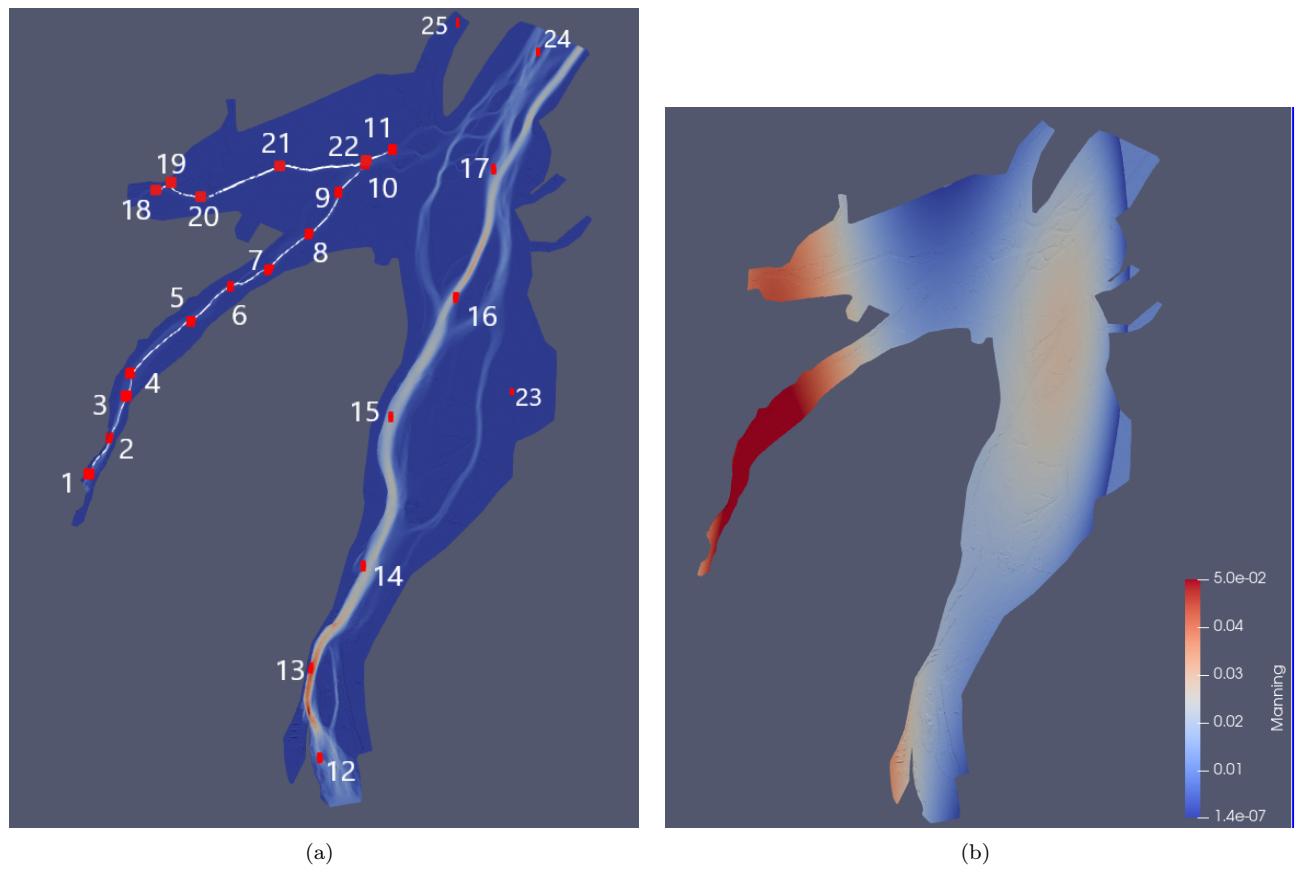


FIGURE 106 – Points utilisés dans le script (a), Nombres de Manning sur le maillage (b)

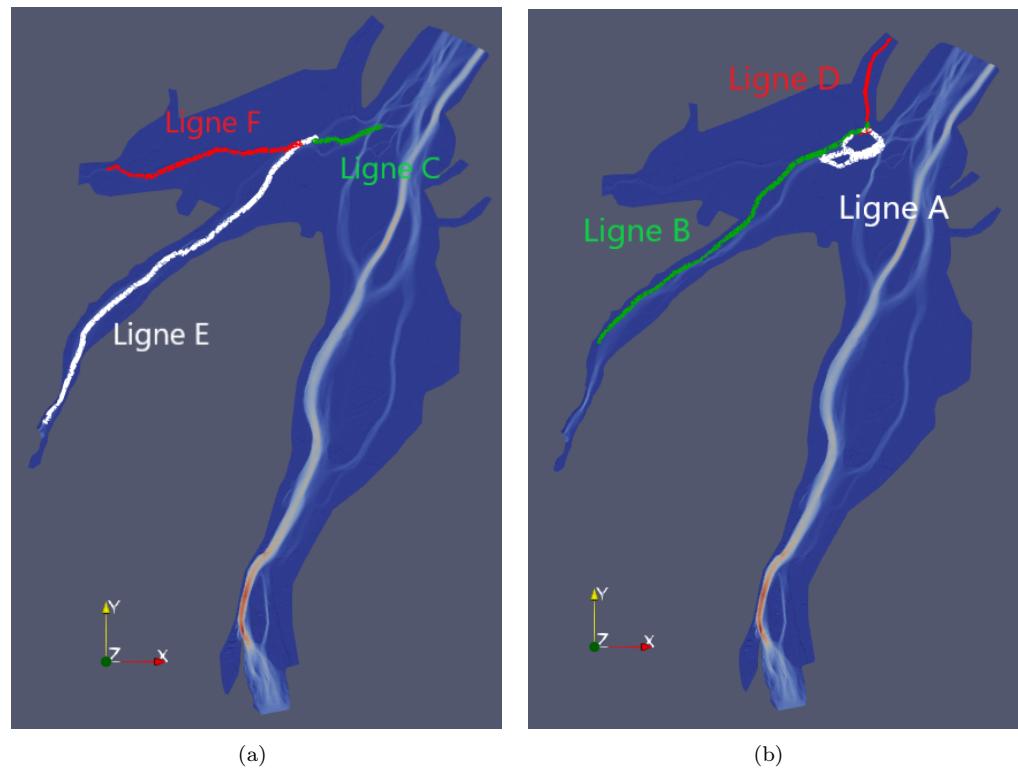


FIGURE 107 – Lignes pour le calcul des résultats

5.5.2.1 Cas archi_0160

Les calculs sont effectués sur 3 lignes dans la rivière présentées dans la Figure 5.5.2. Les résultats obtenus sont présentés sur la Figure 108.

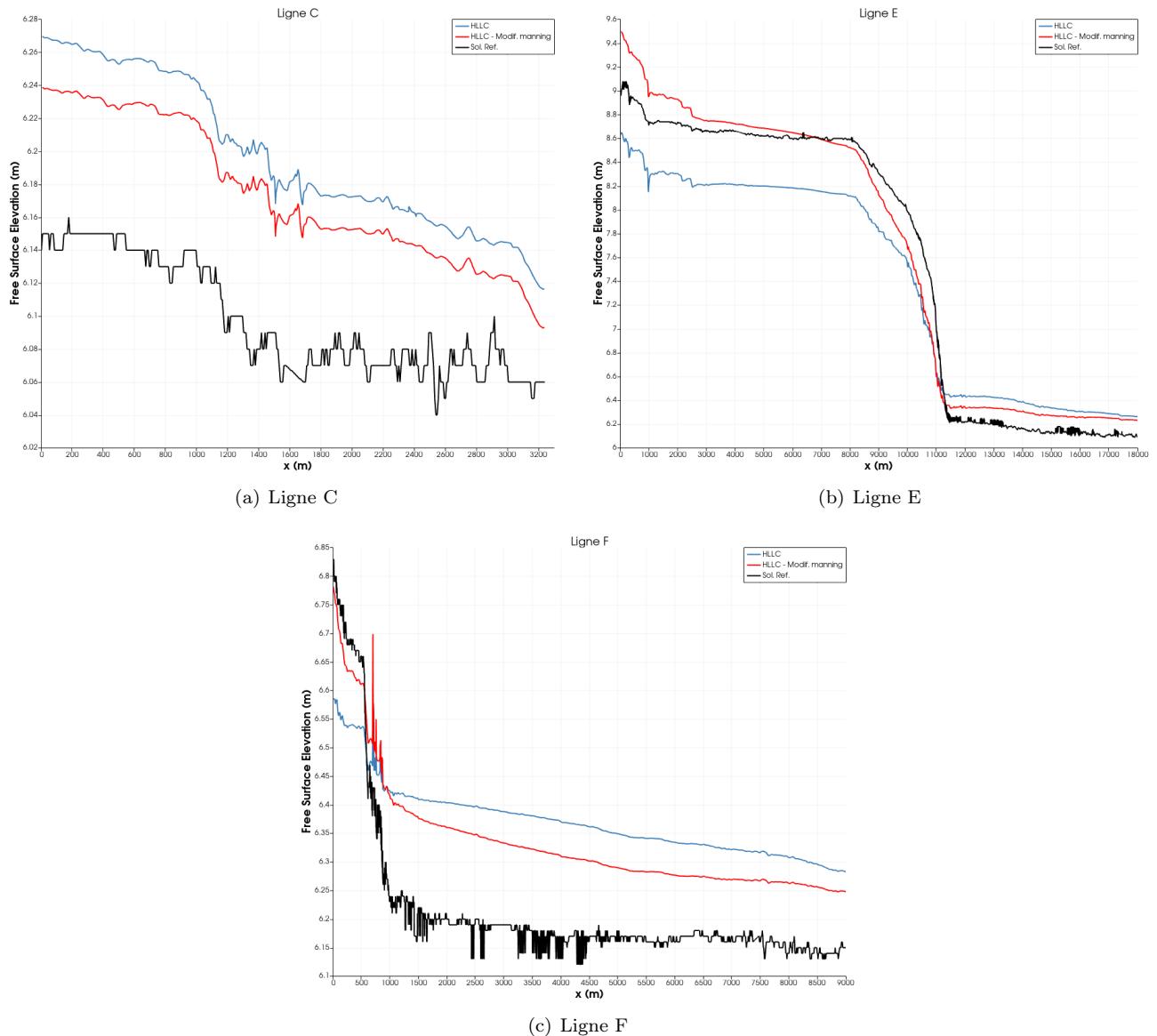


FIGURE 108 – archi_0160 - Écart entre les résultats de la méthode HLLC (figure 103), les résultats calculés sur le maillage avec les nombres de Manning modifiés et les relevés à $t = 40\ 000$ secondes sur 3 lignes traversant l'archipel de Montréal (Free Surface Elevation)

5.5.2.2 Cas archi_0164

Les calculs sont effectués sur 3 lignes dans la rivière présentées dans la Figure 5.5.2. Les résultats obtenus sont présentés sur la Figure 109.

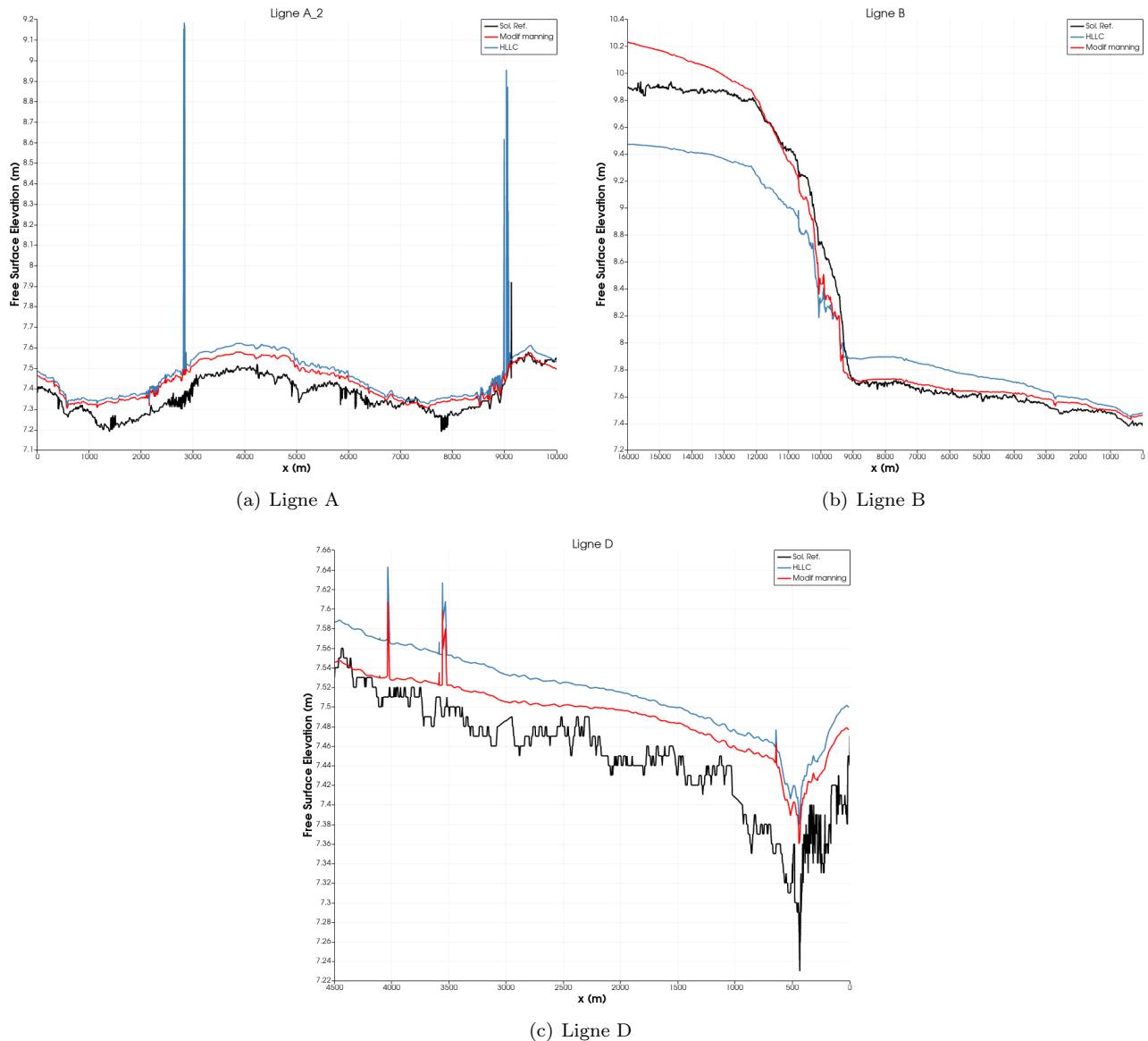


FIGURE 109 – archi_0164 - Écart entre les résultats de la méthode HLLC (figure 103), les résultats calculés sur le maillage avec les nombres de Manning modifiés et les relevés à $t = 40\ 000$ secondes sur 3 lignes traversant l'archipel de Montréal (Free Surface Elevation)

Ces simulations nous permettent de constater des améliorations sur les lignes étudiées. Les maillages peuvent être raffinés encore plus pour des résultats dont la précision sera accrue. L'intérêt d'effectuer ce travail est de générer un maillage dont les propriétés s'approchent le plus possible de celles du véritable fleuve.

6 Calibration du nombre de Manning avec CuteFlow

Dans les écoulements à surface libre, la rugosité du lit d'écoulement exerce une influence significative sur le comportement de l'eau, variant d'un point à un autre dans le domaine d'écoulement et dépendant de plusieurs paramètres. Pour les études des écoulements naturels, le choix des coefficients de rugosité appropriés est d'une importance cruciale pour garantir la fiabilité des résultats.

Dans cette partie, nous présentons une méthodologie efficace pour l'identification du coefficient de Manning, basée sur les données de terrain et un échantillon de résultats numériques. Cette méthode se déroule en deux étapes et combine les simulations numériques (effectuées avec CuteFlow) aux méthodes non déterministes assistées par un modèle de substitution afin de déterminer les valeurs optimales des coefficients de Manning.

La démarche adoptée consiste à générer une solution de référence pour un écoulement dans un domaine divisé en un nombre m de sous-domaines, chacun avec des coefficients de rugosité de Manning n attribués. Par la suite, le processus de calibration démarre à partir d'une solution initiale arbitraire, générée par le modèle d'ensemble réduit, à partir d'une distribution arbitraire des paramètres de Manning. Les paramètres de Manning utilisés pour effectuer les simulations numériques sont obtenues par échantillonnage à l'aide de l'algorithme de Sobol. Cette démarche de calibration peut être illustrée par la figure ci-dessous :

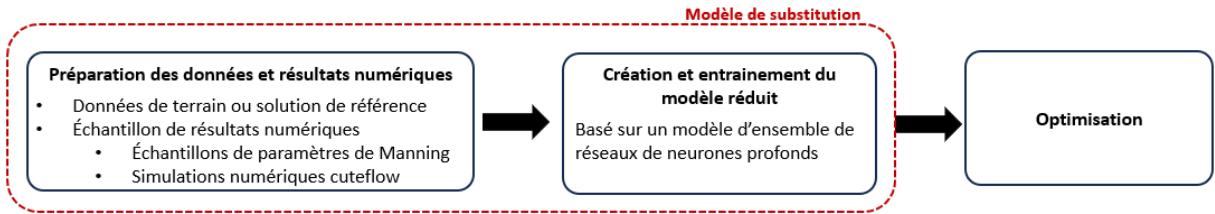


FIGURE 110 – Organigramme du processus de calibration des coefficients de Manning (*basé sur un modèle de substitution*)

6.1 Préparation des données et résultats numériques

Cette étape consiste essentiellement à générer une solution de référence (ou mesures de terrains) ainsi qu'un échantillon de résultats numériques. La solution de référence servira de base pour le calcul de la fonction d'erreur dans le processus d'optimisation tandis que les résultats numériques seront extraits pour entraîner les réseaux de neurones.

On reprend ici le fichier mesh_6072.cgns comme domaine d'application. Nous supposons que le maillage a déjà été créé et fourni et que la solution de référence a déjà été calculée. Dans notre exemple, la solution de référence a été calculée avec le nouveau maillage généré à la section 2.4.1. Afin d'être plus efficient dans la suite, c'est-à-dire la préparation de l'échantillon de résultats numériques, les calculs dans cuteflow seront lancés de manière simultanée. Nous présentons dans la suite les étapes à suivre pour y parvenir.

6.1.1 Zonage du maillage et affectation des nombres de Manning

6.1.1.1 Extraction des identifiants de cellules/mailles

La première étape consiste créer le fichier excel de Zonage du domaine contenant les noms des zones et les numéros de cellules y appartenant (voir section 2.4.1).

Une fois cette étape terminée, nous pouvons importer le fichier Excel dans notre espace de travail sur les serveurs. Nous allons réaliser cet exemple dans le dossier CuteFlow/Calibration_manning/Exemple2.

```

#Depuis le dossier Cuteflow
cd Calibration_manning

# Creer le dossier Exemple2
mkdir Exemple2
cd Exemple2

# Telecharger le fichier Excel de zonage depuis votre ordinateur local vers le dossier courant sur le serveur

# En etant dans le dossier Exemple2, copier le code python de Zonage
cp ../code/Multi_Zonage.py .
  
```

6.1.1.2 Échantillonnage et zonage

La seconde étape consiste à concevoir un plan d'expérience pour les différentes valeurs des coefficients de Manning qui seront utilisées pour les différents cas de simulation. Il existe pour ce faire une multitude de méthodes ou algorithmes permettant de générer une liste d'échantillons (m-uplets) en fonction du nombres de dimensions ou variables à tester. Nous utilisons dans notre cas, l'algorithme de Sobol pour l'échantillonnage des valeurs de Manning. La procédure d'utilisation de ce code est détaillée dans le fichier **Readme.txt** qui l'accompagne.

```

# Creer le dossier base_files dans le dossier Exemple2
cd Exemple2
mkdir base_files
  
```

```

# Copier le fichier de maillage dans le dossier base_files
cp ../../meshes/rect_mesh_convergence/mesh_6072.cgns .

# Copier l'executable de Cuteflow
cp ../../bin/cuteflow .

# Copier le code d'echantillonnage
cp ../../code/Sobol_sequence_Sampling/Sobol_seq.py .

```

Rendu à cette étape, le résultat de la commande `tree` lancée dans le dossier Calibration_manning/Exemple2 devrait être conforme à la figure ci-dessous :

```

[igor81@gra-login1 Exemple2]$ tree
.
└── base_files
    ├── cuteflow
    │   └── mesh_6072.cgns
    ├── Sobol_seq.py
    └── Zonage.py
        └── Zones_mesh6072.xlsx

1 directory, 5 files

```

FIGURE 111 – Arborescence du dossier Exemple2

On crée/active l'environnement virtuel dans lequel on va lancer le code python (voir section 2.4). Une fois l'environnement activé, on utilise le code comme ceci :

```

# depuis le dossier base_files
python3 Sobol_seq.py

```

Le fichier Manning.txt contient un ensemble de **N (m-uplets)** valeurs de Manning que l'on affectera à chacune des zones en fonction des cas. Dans le code Sobol_seq.py, on spécifie le nombre N (**N=2**m échantillonnages**) d'échantillons, le coefficient de variation **cv**, le score **z** correspondant au niveau de confiance associé à **cv**, et les valeurs centrales (valeurs de référence) pour chaque dimension. Les intervalles de recherche des coefficients de Manning sont calculés dans le code, en prenant ces valeurs de références comme solutions cibles, et en considérant une distribution uniforme avec un coefficient de variation **cv**.

Dans l'exemple suivant,

```

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

# Configuration de l'echantillonnage QMC : Parametres d'entree
n_variables = len(space[0])
m_echantillons = 4 # Le nombre d'echantillons est n_variables = 2**m_echantillons.

```

on a défini **N=16** échantillons, **cv = 5%**, **z=1.96** et 4 dimensions pour les 4 zones définies dont les 3 premières représentent les 03 zones sélectionnées et importées dans le fichier de zonage Zones_mesh6072.cgns.

Une fois le fichier Manning.txt créé, on exécute le code de zonage comme suit :

```

cd ..

# Copier le fichier Manning.txt dans le dossier courant
cp base_files/Manning.txt .

# Creation de dossiers et des fichiers Excels contenant les identifiants de cellules et les valeurs de Manning respectives
python3 Multi_Zonage.py Zones_mesh6072.xlsx

```

Le code Multi_Zonage.py lit en entrée le fichier des valeurs de Manning et crée pour chaque m-uplet (chaque ligne du fichier Manning.txt) un dossier Case_*[1- 16]. Il génère dans chacun de ces dossiers, un nouveau fichier de zonage contenant en plus, les valeurs de Manning pour chaque cellule.

On peut vérifier la bonne structure du dossier Calibration_manning/Exemple2 en exécutant la commande `tree` dans ce dossier. La sortie devrait être semblable à la figure ci-après :

```
(ENV) [igor81@gra-login1 Exemple2]$ tree
.
├── base_files
│   ├── cuteflow
│   ├── Manning.txt
│   └── mesh_6072.cgns
└── Sobol_seq.py
.
├── Case_1
│   └── zonage_Case_1.xlsx
├── Case_10
│   └── zonage_Case_10.xlsx
├── Case_11
│   └── zonage_Case_11.xlsx
├── Case_12
│   └── zonage_Case_12.xlsx
├── Case_13
│   └── zonage_Case_13.xlsx
├── Case_14
│   └── zonage_Case_14.xlsx
├── Case_15
│   └── zonage_Case_15.xlsx
├── Case_16
│   └── zonage_Case_16.xlsx
├── Case_2
│   └── zonage_Case_2.xlsx
├── Case_3
│   └── zonage_Case_3.xlsx
├── Case_4
│   └── zonage_Case_4.xlsx
├── Case_5
│   └── zonage_Case_5.xlsx
├── Case_6
│   └── zonage_Case_6.xlsx
├── Case_7
│   └── zonage_Case_7.xlsx
├── Case_8
│   └── zonage_Case_8.xlsx
├── Case_9
│   └── zonage_Case_9.xlsx
└── Zones_mesh6072.xlsx
17 directories, 23 files
```

FIGURE 112 – Sortie de la commande `tree` après exécution du script `Multi_Zonage.py`

6.1.2 Ajout des nombres de Manning aux fichiers de maillage (mesh.cgns)

6.1.2.1 Création des fichiers d'entrée préliminaires

Afin d'ajouter un nombre de Manning à chacune des mailles du fichier de maillage, nous avons besoin du fichier d'entrée contenant les IDs de cellules et les nombres de Manning correspondant pour chacune d'elles. Pour créer ces fichiers, nous utilisons le script `Multi_gen_entree_gen_Manning.py` qui va générer un fichier « `Entree_gen_manning_k.txt` », $k=\{1,2,\dots,16\}$ pour chacun des 16 cas considérés.

Depuis le dossier **Calibration_manning/Exemple2**, on s'assure d'être dans l'environnement virtuel puis on exécute les commandes suivantes :

```
# Copier le script Multi_gen_entree_gen_Manning.py dans le repertoire de travail
cp ..//code/Multi_gen_entree_gen_manning.py .

# Executer le code
python3 Multi_gen_entree_gen_manning.py
```

Une fois cette commande exécutée, une simple vérification avec la commande `tree` ou `ls` permet de confirmer que chacun des 16 dossiers contient un fichier texte « `Entree_gen_manning_k` », $k=\{1,2,\dots,16\}$

6.1.2.2 Lancement du script `Multi.sh`

Le script `Multi.sh` lit deux fichiers en input et crée dans chaque dossier de simulation un fichier de données (**donnees.f**) et un code python pour l'assignation des valeurs de manning à l'ensemble du maillage (**gen_manning.py**).

Dans le premier fichier (**Manning.txt**) passé en argument, il lit pour m-uplet (correspondant à un cas de simulation), la dernière valeur du coefficient de Manning et la stocke dans variable `average_manning`. Elle correspond au coefficient de Manning de la zone non sélectionnée à l'étape de zonage. Dans le deuxième fichier (**initialisation.dat**) passé en argument, le script `Multi.sh` lit les valeurs de débit, et de hauteur d'eau (amon et aval) puis, génère dans chaque dossier le fichier de données avec ces paramètres.

```
GNU nano 4.6
8 2.000 0.500
```

FIGURE 113 – Contenu du fichier `initialisation.dat`

Dans le fichier "initialisation.dat", la première valeur correspond au débit, la seconde à **h amont** et la dernière à **h aval**. Le script `Multi.sh` appelle également deux autres scripts bash (`gen_donnees.sh` et `gen_manning.sh`) afin de générer les fichiers souhaités en sortie. Il est important de bien les copier dans le répertoire courant, comme ceci :

```
# Depuis le dossier Calibration_manning/Exemple2
# Copier les fichiers gen_donnees.sh et gen_manning.sh
cp .../scripts/gen_donnees.sh gen_manning.sh .

# Copier le script Multi.sh
cp .../scripts/Multi.sh .
```

Une fois cette étape réalisée, on peut lancer le script avec la commande :

```
# Lancement du script Multi.sh
./Multi.sh Manning.txt initialisation.dat
```

A cette étape, le contenu de chaque dossier doit être similaire à la figure ci-dessous :

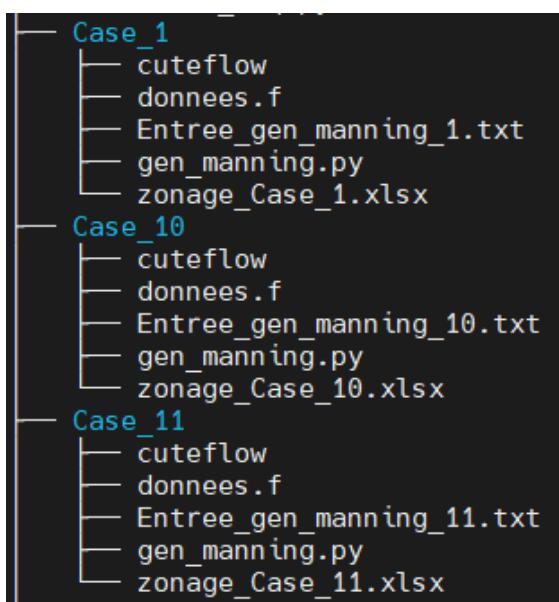


FIGURE 114 – Contenu de chaque dossier après exécution du script `Multi.sh`

6.1.2.3 Crédit des fichiers de maillage `manning_mesh*.cgns` et `manning_mesh.vtk`

Le script `Multi_manning.sh` permet de se déplacer dans chaque dossier et lancer le code `gen_manning.py` afin d'affectation des valeurs de Manning aux cellules du maillage.

Ce script se lance comme suit :

```
# Copier le script Multi_manning.sh dans le dossier courant
cp .../scripts/Multi_manning.sh .

# Se rassurer d'être dans l'environnement virtuel et lancer le script
./Multi_manning.sh fichier_maillage.cgns
```

Avant de lancer le script Multi_manning.sh, il faut se rassurer que les variables y soient bien définies (nom du fichier de maillage, nombre de cas) comme ci-dessous :

```
#!/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 du script Multi_manning.sh (nombre de cas = 16)

6.1.3 Lancement des calculs : job array

Toutes les étapes précédentes ont servi à créer la structure des dossiers nécessaire pour lancer toutes les 16 simulations en simultané. Il faut à présent lancer les calculs sur le cluster de Compute Canada. Pour ce faire, on utilise également un job array comme dans la section 3.4.4. Le fichier que nous utiliserons pour cette étape est le fichier **calib_array.sh** contenu dans le dossier Calibration_manning/scripts.

```
#!/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 – Fichier calib_array.sh (nombre de simulations =16)

Pour soumettre le job à l'ordonnanceur :

```
# Copier le script dans le dossier courant
cp .../scripts/calib_array.sh .

# Lancer le job_array
sbatch calib_array.sh
```

Les 10 simulations seront lancées sur le cluster de calcul comme on peut le voir sur la figure ci-dessous :

(ENV) [igor81@gra-login3 Exemple2]\$ sq	JOBID	USER	ACCOUNT	NAME	ST	TIME_LEFT	NODES	CPU	TRES_PER_N	MIN_MEM	MODELST	(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 – Lancement du job_array

6.2 Post-traitement des fichiers solutions *out_*.cgns* et extraction des données

À ce stade, l'utilisateur peut soit faire une visualisation à distance avec le logiciel Paraview installé sur les grappes de calcul (voir section 4.1) ou encore télécharger les fichiers localement et y accéder à partir du logiciel installé sur son ordinateur.

6.2.1 Visualisation des fichiers solutions générés par CuteFlow

Le processus de visualisation des fichiers solutions générés par CuteFlow est très simple. Il se résume aux grandes étapes suivantes (voir section 4.2) :

Ouverture du fichier solution **out_*.cgns** > choix du lecteur de données > application de l'outil MergeBblocks (*si nécessaire*) > sélection et application de l'outil Cell Data to point Data pour éviter le crénelage (*facultatif*).

6.2.2 Extraction des données

Une fois le fichier solution ouvert dans Paraview, il est possible d'extraire et exporter les solutions temporelles sur les cellules d'intérêt du maillage. Nous présentons dans cette partie un exemple simple d'extraction des solutions du fichier *out_manning_mesh_6072_Case_5.cgns*. Les solutions sont extraites le long d'une droite horizontale divisant le domaine en deux.

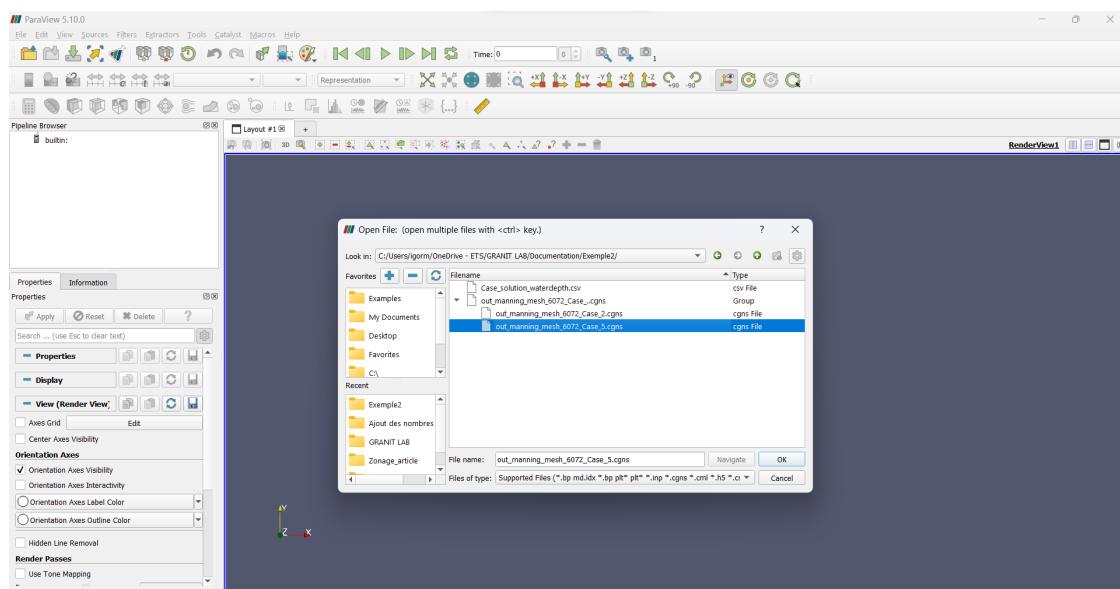


FIGURE 118 – Ouverture du fichier *out_manning_mesh_6072_case_5.cgns* et choix du lecteur

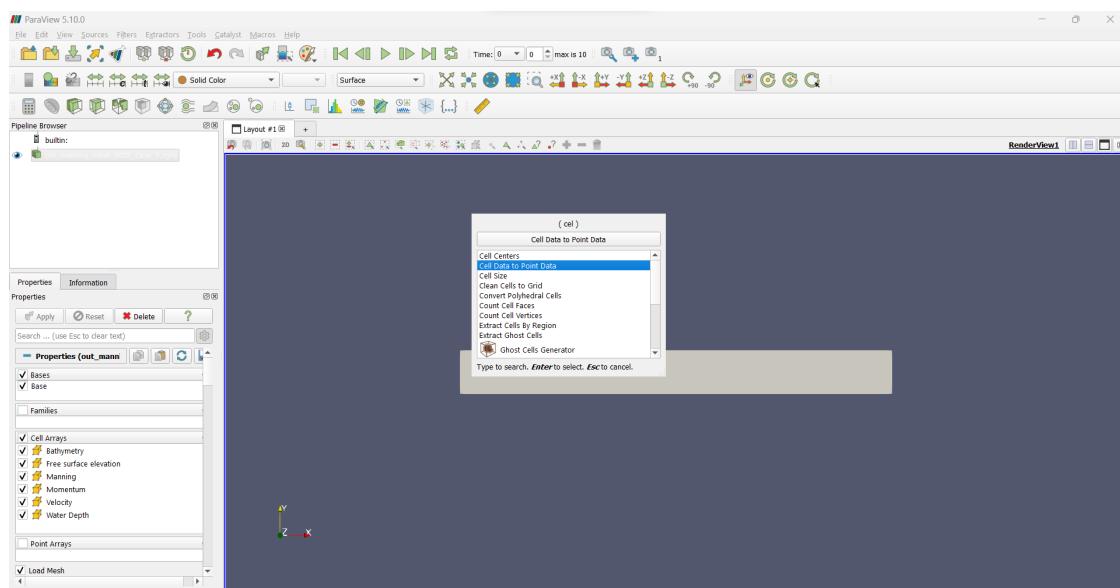


FIGURE 119 – Sélection de l'outil *Cell Data to Point Data*

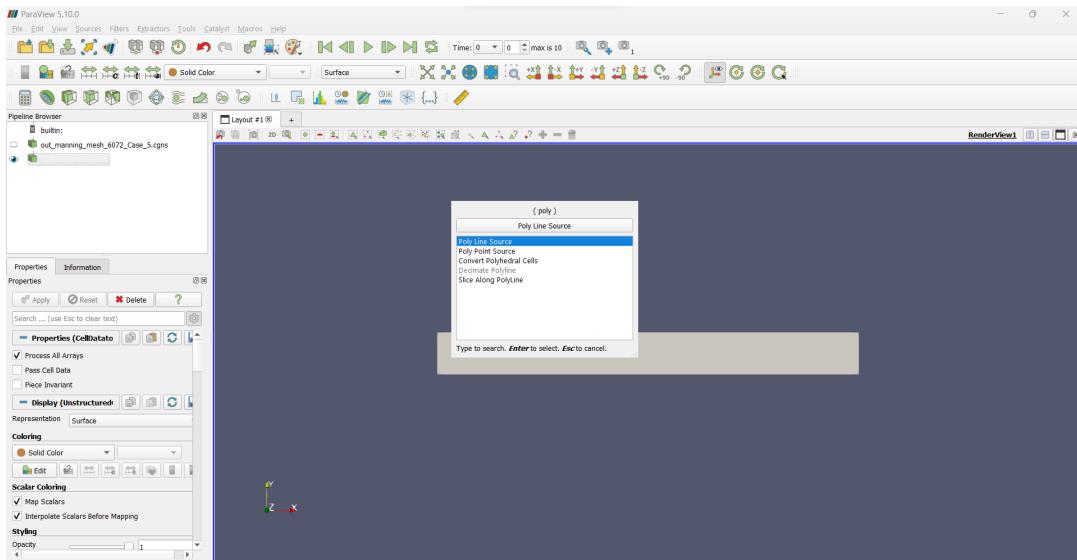


FIGURE 120 – Sélection de l’outil *Poly Line Source* pour tracer la ligne de visualisation des résultats.

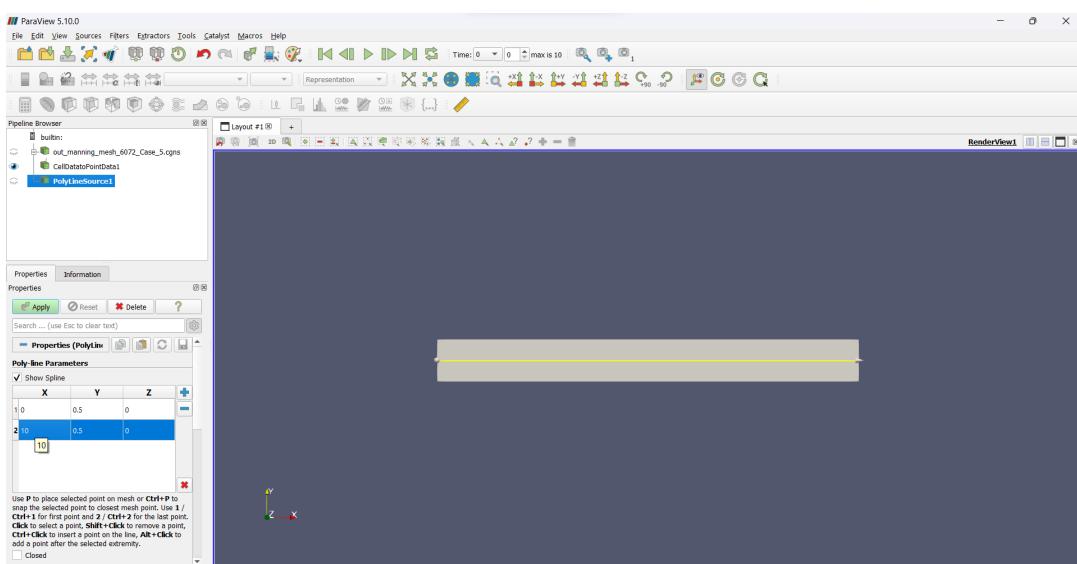


FIGURE 121 – Choix des coordonnées de départ et d’arrivée de la ligne, puis *Apply*

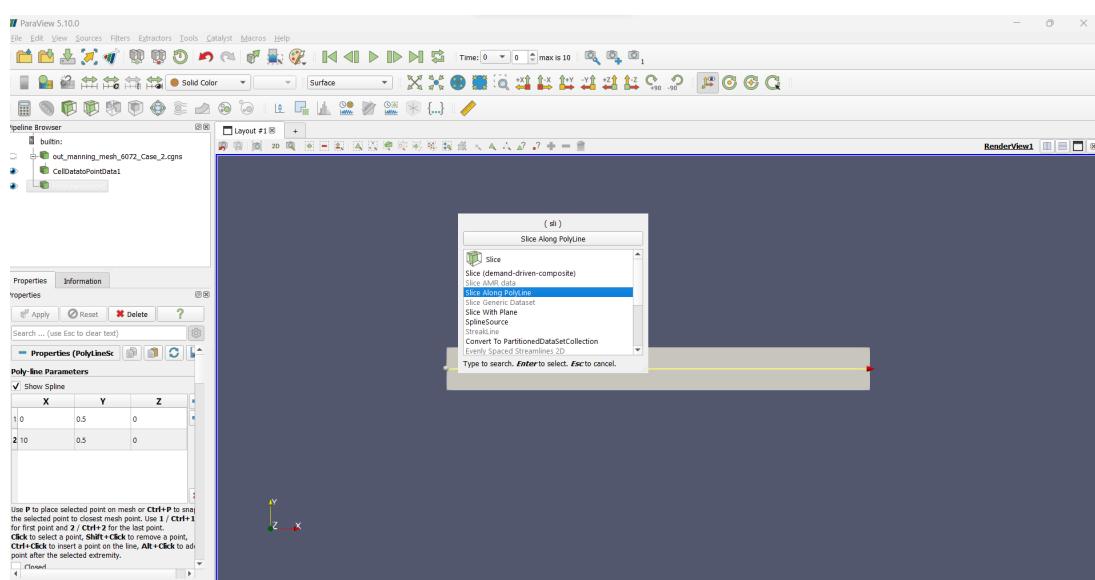
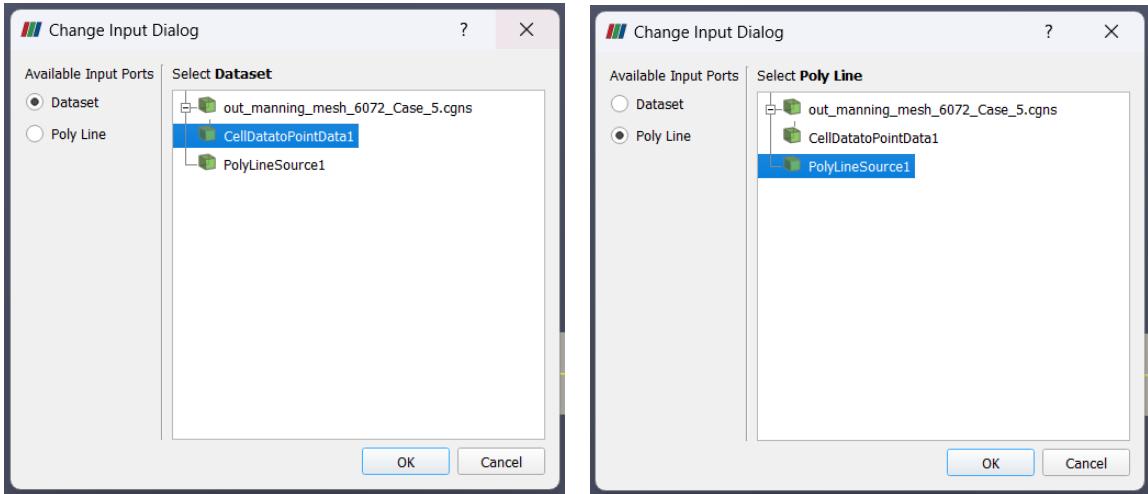


FIGURE 122 – Sélection de l’outil *Slice along polyline* pour projeter la solution sur la ligne.



(a) Vérifier que *Dataset* est bien *CellDataToPointData1*. (b) Vérifier que *Poly Line* est bien *PolyLineSource1*.

FIGURE 123 – Vérification des paramètres de projection de la solution sur la ligne tracée

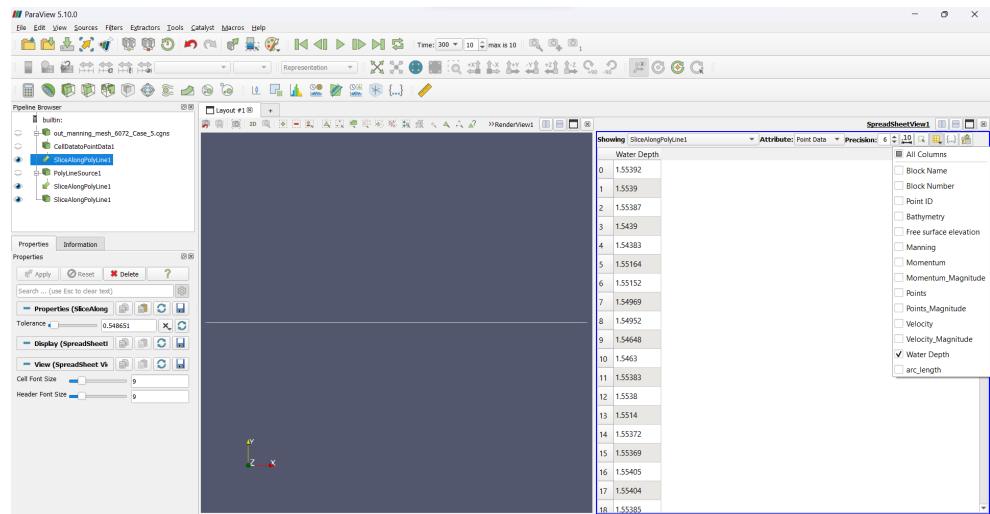


FIGURE 124 – Sélection des attributs à exporter

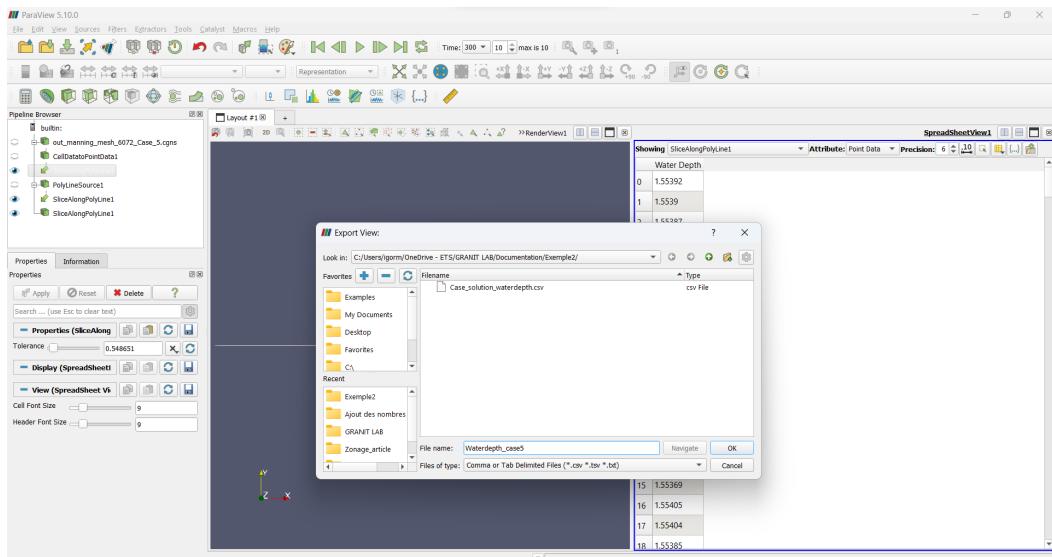


FIGURE 125 – Sauvegarde des solutions dans le fichier *Waterdepth_case5*

Le fichier **Waterdepth_case5** sera crée dans le répertoire spécifié et peut être ouvert avec un éditeur de texte tel que Notepad++ ou un tableur comme Excel, pour une utilisation ultérieure.

Reprendre les mêmes étapes pour les 15 autres fichiers solutions et rassembler les résultats dans un seul fichier comme ci-dessous (**Outputs_223.xlsx** dans notre cas par exemple). Ce fichier servira dans la suite à l'entraînement des réseaux de neurones constituant le modèle d'ensemble.

	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 du fichier final contenant l'ensemble des solutions pour tous les cas de simulation

6.3 Création et entraînement du modèle d'ensemble

Le modèle d'ensemble peut être construit à partir des mesures de différentes variables hydrauliques (hauteur d'eau, quantité de mouvement, vitesse d'écoulement) et même en combinant celles-ci. Il est construit à partir d'un ensemble de réseaux de neurones entraînés et moyennés. Pour l'heure, cette étape est réalisée en local, sur l'ordinateur personnel de l'utilisateur. Elle peut être réalisée avec le logiciel Matlab (en utilisant les bibliothèques intégrés) ou alors en Python, à la discrédition de l'utilisateur.

Dans notre exemple, le modèle d'ensemble est construit à partir des mesures de la hauteur d'eau. Pour ce faire, un ensemble de 10 réseaux de neurones à propagation avant, chacun composé de quatre couches a été formé en initialisant les poids de manière aléatoire. Le code Matlab utilisé pour construire les réseaux de neurones est fourni ci-dessous :

```
%%%%%%%%%%%%%%
% Nom du fichier : Train_dataset.m
%%%%%%%%%%%%%

% Nettoyer la memoire
clc; clear all; close all;

% Lire les donnees
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

% Normaliser les coefficients de Manning
Max_x1 = max(x1,[],all);
Min_x1 = min(x1,[],all);
save('max_manning_data.mat','Max_x1');
save('min_manning_data.mat','Min_x1');

% Pour le calcul de la fonction objective
Max_y1 = max(y1,[],'all');
Min_y1 = min(y1,[],'all');
```

```

save('max_y_data.mat','Max_y1');
save('min_y_data.mat','Min_y1');

% Normalisation de la matrice des valeurs de Manning
for i = size(x1,2)
    x(:,i) = (x1(:,i)-Min_x1)/(Max_x1-Min_x1);
end
% Creation du modèle d'ensemble
M = 10 ; % Le nombre de réseaux de neurones dans le modèle
for m = 1:M
    % Créer le réseau de neurones
    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-07;
    net1 = train(net1, x, y1);

    % Sauvegarder le réseau de neurones
    reseau_hauteur_xnorm = net1;
    save (sprintf('reseau_hauteur_xnorm_%d.mat', m), 'net1')
    pause
end

```

Le fichier des échantillons de Manning (*Inputs_16.xlsx*) lu par ce code se présente comme suit :

	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 du fichier *Inputs_16.xlsx* des valeurs de Manning

Les sorties individuelles de chaque réseau de neurones seront ensuite moyennées pour obtenir la solution de sortie du modèle d'ensemble.

6.4 Optimisation des paramètres

L'optimisation des paramètres de Manning nécessite de soigneusement définir la fonction objective qui sera minimisée en vue de déterminer les solutions optimales. Elle consiste à comparer la solution de sortie du modèle d'ensemble (résultat prédit) à la solution de référence (données de terrain) en optimisant la fonction objective ou fonction de perte. La fonction de perte utilisée dans cette démarche est l'erreur quadratique moyenne (en anglais MSE) définie dans le code ci-après :

```

%%%%%
% Nom du fichier : Objective_function.m
%%%%%

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

% Note
% x_new doit etre normalise avant d'etre appele par PSO ou AG
% M correspond au nombre de reseaux de neurones dans le modele d'ensemble
% Npts correspond au nombre de points de mesures

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

% Lecture des valeurs observees (Case_solution)
y_measure = xlsread('Case_solution.xlsx', 'A2:A224'); % Hauteur d'eau

% Bouche sur les reseaux de neurones
nets = cell(1, 10);
for m = 1:M
    % Charger les reseaux de neurones
    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});

    % Prediction des sorties grace au RNA
    ypredict = ANN{m}(x_new');

    % Enregistrement dans la matrice Ym du modele d'ensemble
    Ym(:,m) = ypredict ;
end

% Calcul de la valeur moyenne de Ypredict
ypredict_ens = mean(Ym,2);

% Calcul de la variance du modele d'ensemble
Sigma_ens = 0;
for m = 1:M
    % Calcul de la variance du reseau de neurone courant
    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 a Sigma_ens
    Sigma_ens = Sigma_ens + Sigma_m ;
end

% Variance du modele (facultatif)
Sigma_ens =Sigma_ens/M ;

% OBJ : Calcul de l'erreur quadratique (MSE)
f= (norm(ypredict_ens(:,1) -y_measure(:,1)))^2;
f = f/Npts;
end

```

Une fois le modèle d'ensemble construit et la fonction objective définie, la calibration peut être réalisée par l'algorithme génétique ou par optimisation par essaim de particules (PSO). Les codes Matlab utilisés sont fournis ci-après :

```

%%%%%
% Nom du fichier : GA.m
% Programme principal d'optimisation par algorithme genetique
%%%%%

clc;
close all;
clear all;
tic;

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

% Bornes sup et inf des intervalles de recherche pour les valeurs de Manning
lb = [0.0198 0.0235 0.0216 0.0153];
ub = [0.0242 0.0285 0.0264 0.0187];

% Normalisation des bornes
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)

% Valeurs des mannings calibres par optimisation
x_abs = x*(Max_x1-Min_x1)+Min_x1;

% Valeurs de Manning exactes/souhaitees
Manning_exacte = [0.022, 0.026, 0.024, 0.017];

% Table des resultats
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);

% Qualite des resultats et affichage (Exemple)
[f,ypredict_ens,y_measure,Sigma_ens] = fun(x);
disp('La variance du modele est:');
disp(Sigma_ens);

```

```

%%%%%%%%%%%%%
% Nom du fichier : PSO.m
% Programme principal d'optimisation par essaim de particules
%%%%%%%%%%%%%

% 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');

% Bornes sup et inf
lb = [0.0198 0.0235 0.0216 0.0153];
ub = [0.0242 0.0285 0.0264 0.0187];

% Normalisation des bornes
lb_norm = (lb-Min_x1)/(Max_x1-Min_x1);
ub_norm = (ub-Min_x1)/(Max_x1-Min_x1);
nvars = 5;

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

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

% Valeurs des mannings calibres par optimisation
x_abs = x*(Max_x1-Min_x1)+Min_x1;

% Valeurs de Manning exactes/souhaitées
Manning_exacte = [0.022, 0.026, 0.024, 0.017];

% Table des résultats
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);

% Qualité des résultats et affichage (Exemple)
[f,ypredict_ens,y_measure,Sigma_ens] = fun(x);
disp('La variance du modèle est:');
disp(Sigma_ens);

```

7 Calibration des coefficients de Manning : cas tests

On présente dans cette section des exemples permettant d'évaluer la performance et la robustesse de la méthodologie et des algorithmes développés précédemment pour la calibration efficace des coefficients de Manning, spécifiquement dans le contexte des écoulements subcritiques. Pour chaque test, les coefficients de Manning ont été dérivés à partir des valeurs de références. L'échantillonnage a été effectué à l'aide de l'algorithme de Sobol, avec les intervalles de recherche définis en prenant ces valeurs de références comme solutions cibles, et en considérant une distribution uniforme avec un coefficient de variation **cv** fixé à 5 %. Les fichiers qui seront utilisés dans cette section se trouvent dans le dossier **CuteFlow/Calibration_manning/docs**

7.1 Écoulement dans un canal divergent

Dans un premier temps, nous allons évaluer l'efficacité de la procédure sur un domaine simple à géométrie variable. Le canal choisi présente une longueur totale de 16 mètres, et se compose d'une section étroite de 6 mètres de long sur 1 mètre de large, ainsi que d'une section plus large de 10 mètres de long sur 2 mètres de large. La subdivision du maillage est représentée par la figure ci-dessous :

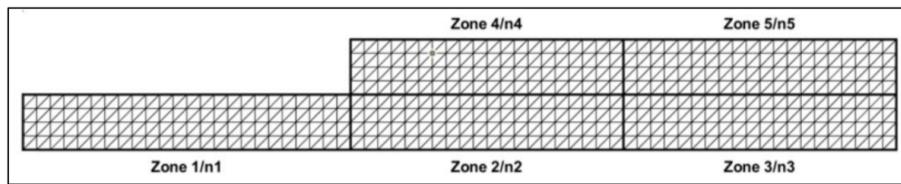


FIGURE 128 – Domaine du canal divergent et décomposition en 5 sous-domaines

7.1.1 Initialisation et paramétrage

Une analyse préliminaire a permis d'identifier des zones d'intérêts, propices à la localisation des points de mesure. Dans la pratique, l'hydraulicien doit avoir une bonne maîtrise du cours d'eau simulé pour bien choisir la position des points de mesure.

Les résultats sont recueillis et comparés en 108 points, répartis stratégiquement le long d'une ligne horizontale et trois lignes verticales respectives illustrées dans la figure 129.

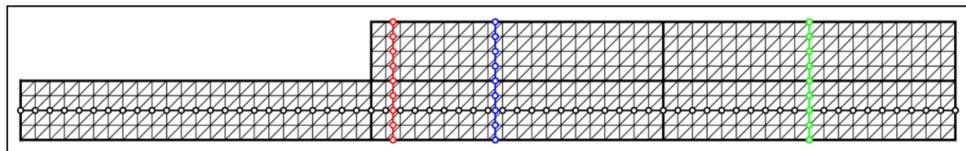


FIGURE 129 – Localisation des points de mesures - Canal divergent

Les paramètres de simulation et de calibration sont répertoriés dans le tableau suivant :

TABLE 1 – Paramètres de calibration - Canal divergent

Nombre d'éléments	832		
Nombres de noeuds du maillage	4893		
Nombres de paramètres à calibrer	5		
Nombre d'échantillons	256		
Taille du modèle d'ensemble	10		
Points d'observations	108		
Zone	Bathymétrie	Intervalle de recherche	Paramètre désiré
n_1	Béton	$[1, 26 - 1, 54] \times 10^{-2}$	$1, 40 \times 10^{-2}$
n_2	Sable	$[1, 53 - 1, 87] \times 10^{-2}$	$1, 70 \times 10^{-2}$
n_3	Coupes de roches	$[2, 26 - 2, 75] \times 10^{-2}$	$2, 50 \times 10^{-2}$
n_4	Pavés	$[3, 16 - 3, 84] \times 10^{-2}$	$3, 50 \times 10^{-2}$
n_5	Gravier	$[2, 53 - 3, 07] \times 10^{-2}$	$2, 80 \times 10^{-2}$

Les résultats obtenus découlent d'une initialisation avec plan horizontal. Un débit de $350\text{dm}^3.\text{s}^{-1}$ est imposé à l'entrée et une hauteur d'eau $h = 20\text{cm}$ est fixée à la sortie.

7.1.2 Résultats

Le modèle d'ensemble a été construit à partir des mesures de la hauteur d'eau. Pour ce faire, un ensemble de 10 réseaux à propagation avant, a été formé en initialisant les poids de manière aléatoire. Les sorties individuelles ont été moyennées pour obtenir la solution de sortie de l'ensemble. Les valeurs obtenues pour les coefficients de Manning, avec les deux modèles, sont répertoriées dans le tableau ci-dessous. Elles se révèlent remarquablement satisfaisantes, affichant des erreurs relatives maximales inférieures à 5 %.

TABLE 2 – Résultats de la calibration - Canal divergent

Zone	n_1	n_2	n_3	n_4	n_5
Valeur cible	$1,4 \times 10^{-2}$	$1,7 \times 10^{-2}$	$2,5 \times 10^{-2}$	$3,5 \times 10^{-2}$	$2,8 \times 10^{-2}$
Algorithme Génétique (AG)					
Valeur calibrée	$1,42 \times 10^{-2}$	$1,64 \times 10^{-2}$	$2,42 \times 10^{-2}$	$3,52 \times 10^{-2}$	$2,74 \times 10^{-2}$
Erreurs relatives (%)	1,23	3,32	3,26	0,43	2,00
Optimisation par essaim de population (OEP)					
Valeur calibrée	$1,39 \times 10^{-2}$	$1,62 \times 10^{-2}$	$2,62 \times 10^{-2}$	$3,60 \times 10^{-2}$	$2,92 \times 10^{-2}$
Erreurs relatives (%)	1,11	4,92	4,64	2,93	4,31

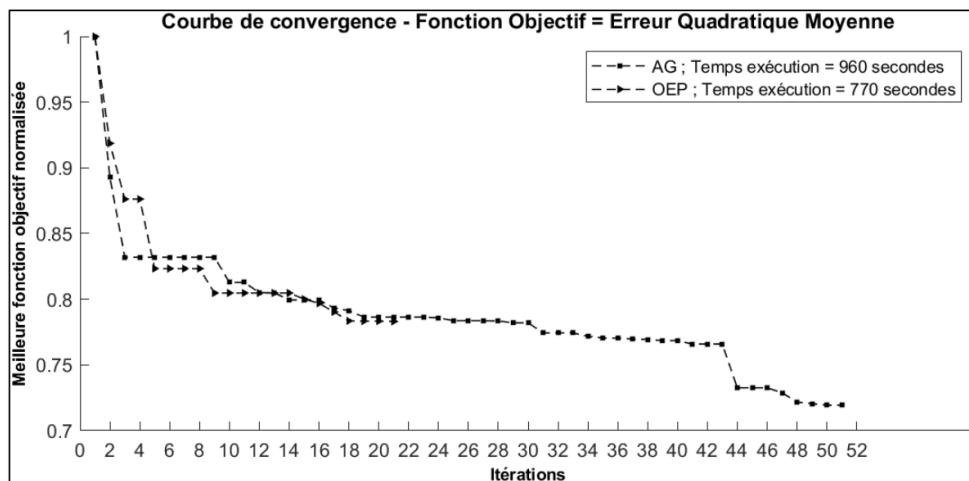


FIGURE 130 – Historique de convergence des algorithmes d'optimisation (AG et PSO) - Canal divergent

La calibration réalisée par l'algorithme génétique a nécessité 960 secondes pour 51 itérations, tandis que la calibration par optimisation par essaim de particules a demandé 770 secondes pour 21 itérations. Ces résultats démontrent clairement l'efficacité et la rapidité inhérente à la méthodologie proposée pour l'identification précise des coefficients de Manning.

7.2 Écoulement dans une rivière avec obstacle non submersibles

Le second test est réalisé sur un domaine de rivière avec des obstacles en pierres. Le but est d'évaluer la robustesse de la méthodologie dans un contexte de géométrie complexe, se rapprochant de la réalité, en restant relativement simple à reproduire. Les dimensions du canal sont de 28,5 mètres sur 20 mètres, avec une bathymétrie constante, de profondeur 4 mètres, et la présence de deux obstacles en forme de cylindres, chacun ayant un diamètre de 4 mètres. La figure 131 illustre une décomposition du domaine en 4 sousdomaines, ainsi que la position des 28 points de mesures. Ces points de mesure ont été choisis de façon stratégique (c'est-à-dire, avant les obstacles, après les obstacles et autour des obstacles).

7.2.1 Initialisation et paramétrage

Pour cette configuration expérimentale, un débit de $62\text{m}^3.\text{s}^{-1}$ est imposé à l'entrée et le niveau d'eau fixé à $h = 0.0\text{m}$ à la sortie. La simulation de l'écoulement a été initialisée à partir d'un plan horizontal et calculée sur une période de 6 minutes. Les paramètres initiaux de calibration sont présentés dans le tableau ci-après :

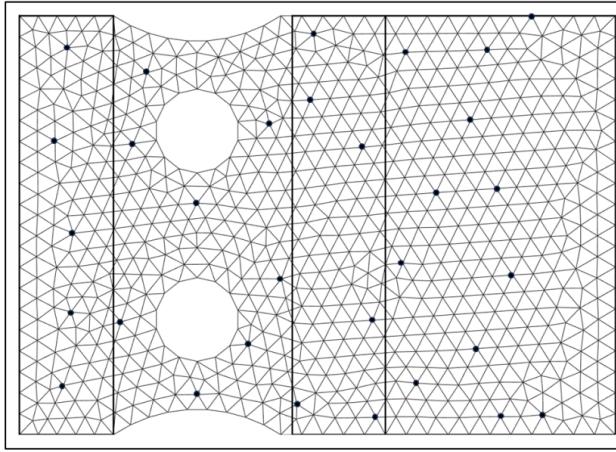


FIGURE 131 – Décomposition de la rivière avec obsacle en 4 sous-domaines et localisation des points de mesures

TABLE 3 – Paramètres de calibration - Rivière avec obstacles en pierres

Nombre d'éléments	1408		
Nombres de noeuds du maillage	767		
Nombres de paramètres à calibrer	4		
Nombre d'échantillons	256		
Taille du modèle d'ensemble	10		
Points d'observations	28		
Zone	Bathymétrie	Intervalle de recherche	Paramètre désiré
n_1	Gravier fin	$[2, 16 - 2, 64] \times 10^{-2}$	$2,40 \times 10^{-2}$
n_2	Gravier	$[3, 16 - 3, 84] \times 10^{-2}$	$3,50 \times 10^{-2}$
n_3	Sable	$[1, 80 - 2, 20] \times 10^{-2}$	$2,00 \times 10^{-2}$
n_4	Pavés	$[2, 71 - 3, 29] \times 10^{-2}$	$3,00 \times 10^{-2}$

7.2.2 Résultats

Les résultats obtenus ont également été générés à partir d'un modèle de substitution basé exclusivement sur la variable de hauteur d'eau et sont synthétisés ci-après.

TABLE 4 – Résultats de la calibration - Rivière avec obstacles non submersibles en pierres

Zone	n_1	n_2	n_3	n_4
Valeur cible	$2,4 \times 10^{-2}$	$3,5 \times 10^{-2}$	$2,0 \times 10^{-2}$	$3,0 \times 10^{-2}$
Algorithme Génétique (AG)				
Valeur calibrée	$2,38 \times 10^{-2}$	$3,51 \times 10^{-2}$	$2,08 \times 10^{-2}$	$2,97 \times 10^{-2}$
Erreurs relatives (%)	0,77	0,33	4,04	1,16
Optimisation par essaim de population (OEP)				
Valeur calibrée	$2,42 \times 10^{-2}$	$3,51 \times 10^{-2}$	$2,09 \times 10^{-2}$	$2,96 \times 10^{-2}$
Erreurs relatives (%)	0,64	0,21	4,44	1,53

L'évolution des courbes de convergence des différentes approches, accompagnées de leurs temps d'exécution respectifs, permet d'observer que l'optimisation par essaim de particules converge de manière plus rapide avec moins d'itérations par rapport à l'algorithme génétique. Par ailleurs, les résultats obtenus avec les deux méthodes démontrent une satisfaction équivalente, sans distinction notable entre l'une et l'autre.

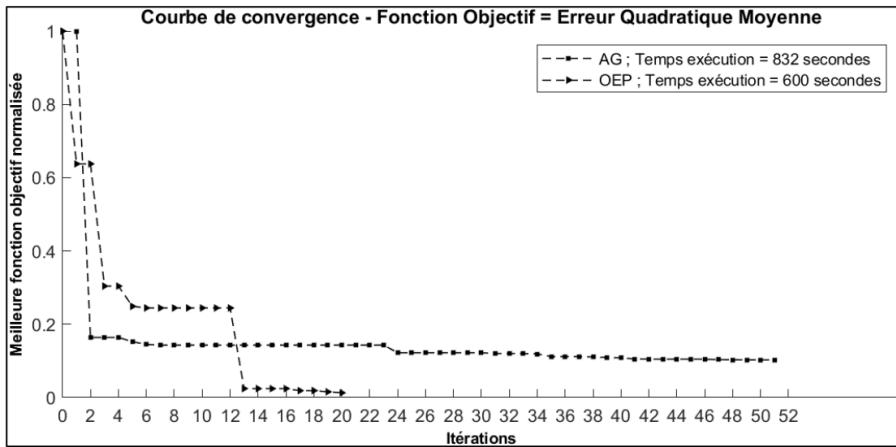


FIGURE 132 – Historique de convergence des algorithmes d'optimisation (AG et PSO) - Rivière avec obstacle non submersible en pierres

7.3 Cas de la rivière des Mille - Îles

Dans cette section on évalue la capacité de la méthodologie proposée à identifier efficacement les valeurs des coefficients de rugosité de Manning sur une bathymétrie complexe et réelle. Ce dernier test est réalisé sur le domaine de la rivière des Mille-Îles, un cours d'eau d'environ 42 kilomètres de long, situé au coeur de la région de Montréal. Le maillage est dense et constitué plus de 480 000 éléments. Les données utilisées pour dans cette section peuvent être trouvées dans le dossier [CuteFlow/Calibration_manning/docs_calibration/Mille_Îles_10percent](#).

Les données sont mesurées à des points stratégiques du chenal, là où des phénomènes d'intérêts se produisent pendant l'écoulement. Ces points stratégiques ont été déterminés à partir d'une étude préliminaire dont les résultats sont illustrés à la figure 133.

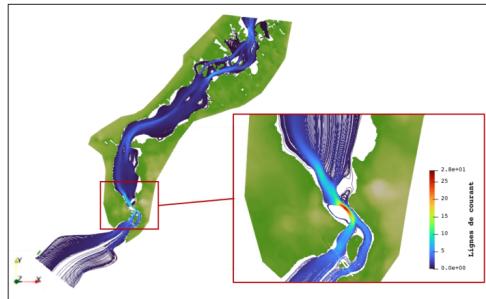


FIGURE 133 – Résultats numériques de l'étude préliminaire de l'écoulement : Lignes de courant avec coloration basée sur la quantité de mouvement - Rivière des Mille-Îles

Le maillage étudié est subdivisé en 4 sous domaines illustrés par la figure 134 ci-dessous. Le domaine d'intérêt est le lit d'écoulement (bathymétrie > 0.1m) délimité par trois sous domaines, notamment : zone 1, zone 2 et zone 3.

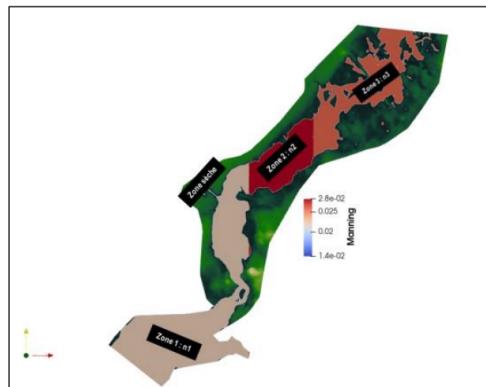


FIGURE 134 – Décomposition de la rivière des Mille Iles en 03 sous-domaines

7.3.1 Initialisation et paramétrage

Les paramètres d'écoulement, notamment les hauteurs d'eau, sont recueillis et comparés en 420 points, repartis le long de trois lignes situées stratégiquement le long du canal d'écoulement. La figure 135 présente les lignes d'observation sur le modèle numérique.

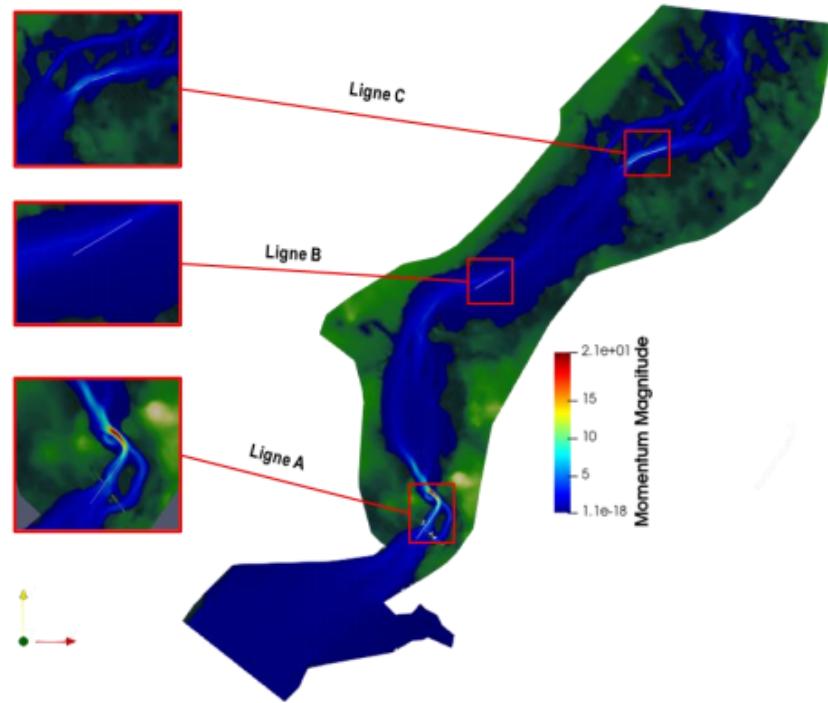


FIGURE 135 – Lignes de mesure des données - Rivière des Mille-Îles

Pour ce test, le coefficient de variation a été fixé à 10% et les algorithmes d'optimisation légèrement modifiés. Les paramètres de simulation et de calibration sont répertoriés dans le tableau suivant :

TABLE 5 – Paramètres de calibration - Rivière avec des Mille Iles

Nombre d'éléments	481930
Nombres de noeuds du maillage	243161
Nombres de paramètres à calibrer	3
Nombre d'échantillons	128
Taille du modèle d'ensemble	10
Points d'observations	420

Hyperparamètres - Algorithmes d'optimisation			
Algorithme Génétique	Algorithme Génétique Hybride		
- PopulationSize = 25	- PopulationSize = 5		
- MaxGenerations = 200	- MaxGenerations = 20		
OEP	OEP Hybride		
- SwarmSize = 25	- SwarmSize = 25		
- MaxIterations = 200	- MaxIterations = 200		
Zone	Bathymétrie	Intervalle de recherche	Paramètre désiré
n_1	Sable	$[1, 77 - 2, 63] \times 10^{-2}$	$2, 20 \times 10^{-2}$
n_2	Gravier	$[2, 25 - 3, 35] \times 10^{-2}$	$2, 80 \times 10^{-2}$
n_3	Coupes de roches	$[2, 09 - 3, 11] \times 10^{-2}$	$2, 60 \times 10^{-2}$

7.3.2 Résultats

Les résultats obtenus sont générés à partir d'un modèle d'ensemble fondé sur les mesures de la hauteur d'eau le long du canal et sont consignés dans le tableau ci-dessous.

TABLE 6 – Résultats de la calibration - Rivière avec obstacles non submersibles en pierres

Zone	n_1	n_2	n_3
Valeur cible	$2,20 \times 10^{-2}$	$2,80 \times 10^{-2}$	$2,60 \times 10^{-2}$
Algorithme Génétique (AG) [Temps de calcul : 1346s]			
Valeur calibrée	$2,174 \times 10^{-2}$	$3,043 \times 10^{-2}$	$2,604 \times 10^{-2}$
Erreur relative (%)	1,18	8,69	0,15
Algorithme Génétique Hybride(AG+Fmincon) [Temps de calcul : 241s]			
Valeur calibrée	$2,204 \times 10^{-2}$	$2,742 \times 10^{-2}$	$2,590 \times 10^{-2}$
Erreur relative (%)	0,18	2,07	0,38
Optimisation par essaim de population (OEP) [Temps de calcul : 931s]			
Valeur calibrée	$2,209 \times 10^{-2}$	$2,480 \times 10^{-2}$	$2,590 \times 10^{-2}$
Erreur relative (%)	0,40	11,42	0,38
Optimisation par essaim de population Hybride (OEP+Fmincon) [Temps de calcul : 230s]			
Valeur calibrée	$2,204 \times 10^{-2}$	$2,742 \times 10^{-2}$	$2,590 \times 10^{-2}$
Erreur relative (%)	0,18	2,07	0,38

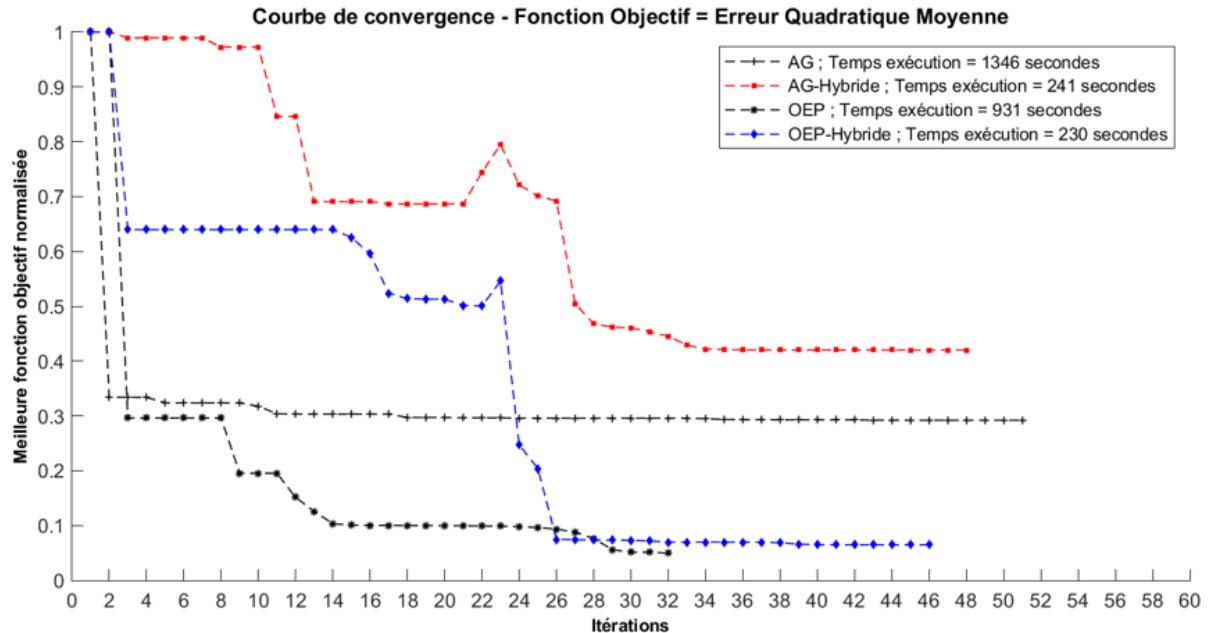


FIGURE 136 – Historique de convergence des algorithmes d'optimisation (AG, AGH, PSO et PSOH) - Rivière des Milles-Îles

Ces résultats confirment une fois de plus la supériorité de la calibration avec Optimisation par Essaim de Particules par rapport à l'Algorithme Génétique. De plus, on remarque une réelle amélioration de la vitesse de convergence après modification des fonctions de bases. L'ajout de la fonction Hybride, **Fmincon**, permet de réduire le temps d'exécution tout en garantissant l'obtention de la solution optimale.

Références

- [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), p. 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 et Azzeddine 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 et 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 (mai 2007). DOI : <10.1080/15502280601149577>.
- [7] Arun SUTHAR et Azzeddine SOULAÏMANI. *Internship report parallelization of shallow water equations solver : Cuteflow*. Juill. 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 et 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), p. 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>.