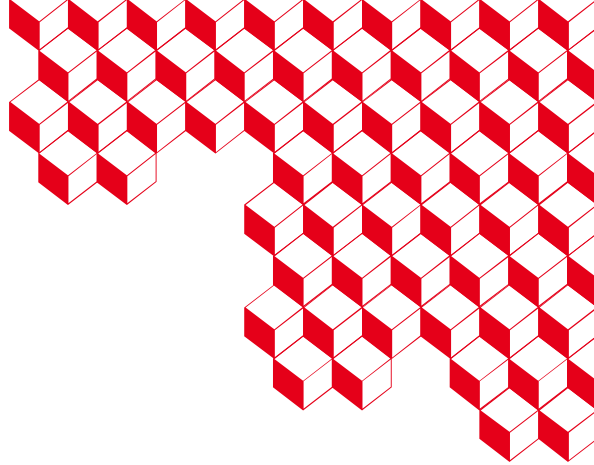


Rencontres Arcane

4ème édition

Nouvelles fonctionnalités

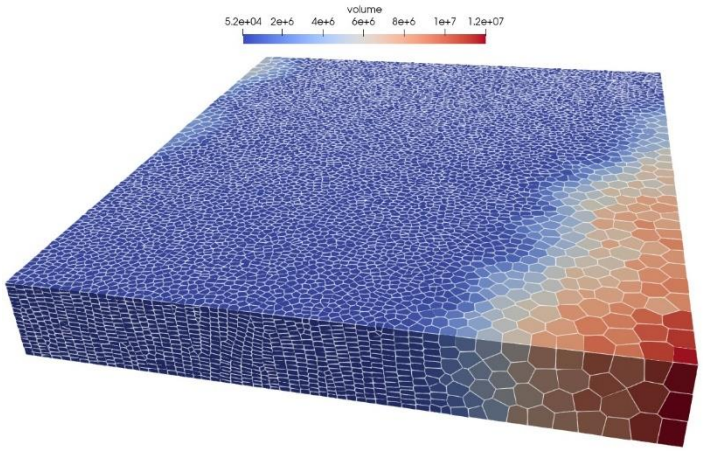
Lundi 24 mars 2025

Sommaire

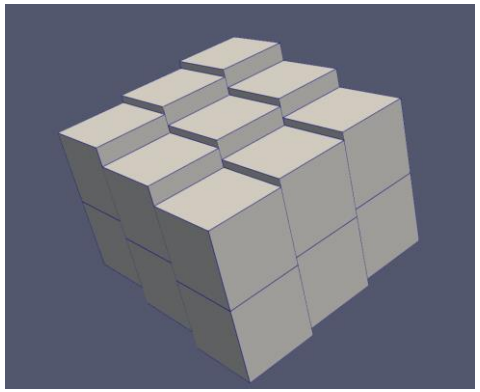
1. **Support des maillages polyédriques**
2. **AMR par patch pour les maillages cartésiens**
3. **Raffinement de maillages par subdivision**
4. **Modification des options du jeu de données via les arguments de la commande de lancement**

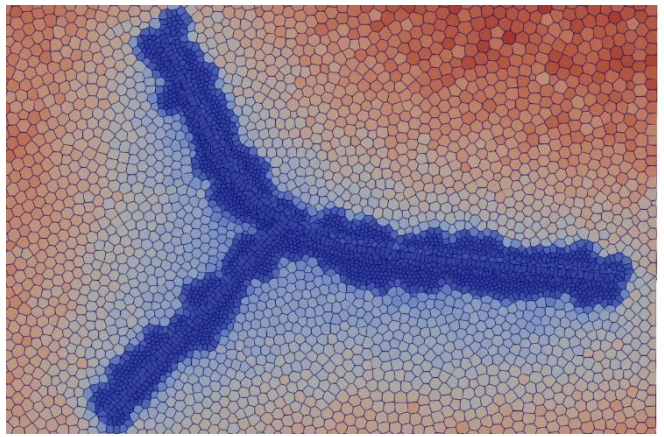
1. Support des maillages polyédriques

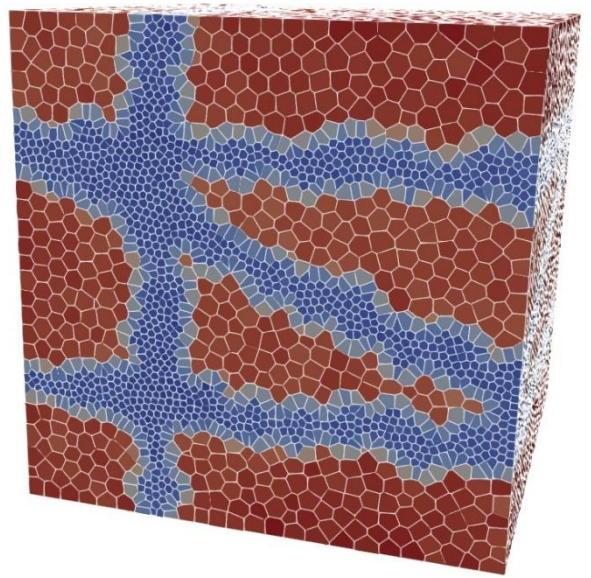
Support des maillages polyédriques



- Utilisation du polyédrique à IFPEN dans deux contextes
  - « Vrai » polyédriques
  - Maillages mis en conformité
- Génération pour mieux prendre en compte les géométries complexes
  - Thèse Alexandre Marin (Laurent Astart– Alexandra Bac)
- Mise en conformité de maillages CPG
  - Maillages faillés
  - Maillages LGR



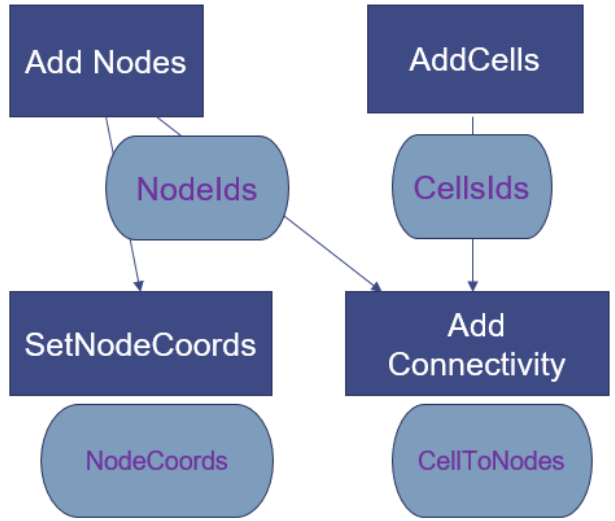




Support des maillages polyédriques

- Nouvelle implémentation de `IMesh`
  - Implémentation encore partielle
- Utilisation de la bibliothèque Neo (arcaneframework/tools/neo)
- Neo fournit un graphe de propriétés et d'algorithmes pour les opérations sur le maillage
  - Construction graphe de dépendances des propriétés du maillage, coordonnées d'items, connectivités…
  - Les opérations sur le maillage sont ordonnées pour satisfaire les dépendances entre propriétés
- Garanti l'intégrité et l'ordre des opérations de maillage
  - Dans un contexte de maillage évolutif (ajouts/suppressions d'items imbriqués)
  - Pour gérer de nombreuses connectivités (maillage, non-conformité ; numériques)
    - Migration d'items en parallèle

```
Neo::Mesh mesh {"MeshName"};
auto cell_family = mesh.addFamily(Neo::ItemKind::IK_Cell, "CellFamily");
auto node_family = mesh.addFamily(Neo::ItemKind::IK_Node, "NodeFamily");
// Arrays cell_uids, node_uids, cell_to_nodes, node_coords, nb_node_per_cell are given
Neo::FutureItemRange added_cells, added_nodes;
mesh.scheduleAddItems(cell_family,cell_uids,added_cells);
mesh.scheduleAddItems(cell_family,node_uids,added_nodes);
mesh.scheduleSetItemCoords(node_family, added_nodes, node_coords);
mesh.scheduleAddConnectivity(cell_family, future_cells, node_family,
                             nb_node_per_cell, cell_to_nodes, "cell_to_nodes");
mesh.applyScheduledOperations();
```



Support des maillages polyédriques

- Ajout d'un lecteur de maillage polyédrique 3D dans Arcane au format vtk
  - Maillage VTK de type UNSTRUCTURED_GRID
  - Utilisation de maille de type 42
- Lecteur de fichier au format .vtk ou .vtu (xml)
- Support par défaut des variables et groupes
  - pour les mailles
  - pour les nœuds
- Possibilités de fournir un fichier supplémentaire pour les informations de faces
  - Maillage VTK des faces de type POLYDATA
  - Fichier maillage.vtk.vktfaces.vtk ou maillage.vtu.vtufaces.vtp
  - Support des variables sur les faces, groupe de faces
- Export au format Ensight

Support des maillages polyédriques

- Fichier de maillage au format vtu

```
<?xml version="1.0" encoding="UTF-8"?>
<VTKFile type="UnstructuredGrid" version="0.1" byte_order="LittleEndian">
    <Information>Original file cross_a_2x1x1.egrid converted using GeoPolyMesh CPG2VTP tool</Information>
    <UnstructuredGrid>
        <Piece NumberOfPoints="16" NumberOfCells="2">
            <Points>
                <DataArray type="Float64" NumberOfComponents="3" format="ascii">0.000000 0.000000 50.000000 0.000000 0.000000 150.000000
            </Points>
            <Cells>
                <DataArray type="Int32" Name="connectivity" format="ascii">0 1 3 4 5 6 7 8 10 11 2 3 8 9 10 11 12 13 14 15 </DataArray>
                <DataArray type="Int32" Name="offsets" format="ascii">10 20 </DataArray>
                <DataArray type="UInt8" Name="types" format="ascii">42 42 </DataArray>
                <DataArray type="Int32" Name="faces" format="ascii">8 4 0 3 4 1 4 6 8 7 5 4 1 6 5 0 3 3 11 4 3 8 10 7 4 10 8 11 3 5 4 11
                <DataArray type="Int32" Name="faceoffsets" format="ascii">41 82 </DataArray>
            </Cells>
            <CellData>
                <DataArray type="Int32" Name="CellFlags" format="ascii">1 1</DataArray>
                <DataArray type="Int32" Name="CellArrayFlags" NumberOfComponents="3" format="ascii">1 2 3 1 2 3</DataArray>
                <DataArray type="Int32" Name="GROUP_HALF_CELL" format="ascii">0 1</DataArray>
            </CellData>
            <PointData>
                <DataArray type="Int32" Name="NodeFlags" format="ascii">1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</DataArray>
                <DataArray type="Int32" Name="NodeArrayFlags" NumberOfComponents="3" format="ascii">1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
                <DataArray type="Int32" Name="GROUP_FIRST_CELL_NODES" format="ascii">1 1 0 1 1 1 1 1 1 0 1 1 0 0 0 0</DataArray>
            </PointData>
        </Piece>
    </UnstructuredGrid>
</VTKFile>
```

Support des maillages polyédriques

- Fichier de maillage pour les faces au format vtp

```
<?xml version="1.0" encoding="UTF-8"?>
<VTKFile type="PolyData" version="0.1" byte_order="LittleEndian">
    <Information>Face data for original file cross_a_2x1x1.egrid converted using GeoPolyMesh CPG2VTP tool</Information>
    <PolyData>
        <Piece NumberOfPoints="16" NumberOfPolys="15">
            <Points>
                <DataArray type="Float64" NumberOfComponents="3" format="ascii">0.000000 0.000000 50.000000 0.000000 0.000000 150.000000 100.000000 0
            </Points>
            <Polys>
                <DataArray type="Int32" Name="connectivity" format="ascii">0 3 4 1 6 8 7 5 1 6 5 0 3 11 4 8 10 7 10 8 11 3 4 11 8 6 1 0 5 7 10 3 2 12
                <DataArray type="Int32" Name="offsets" format="ascii">4 8 12 15 18 22 27 32 36 40 43 46 50 55 60</DataArray>
            </Polys>
            <CellData>
                <DataArray type="Int32" Name="FaceFlags" format="ascii">1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</DataArray>
                <DataArray type="Int32" Name="FaceArrayFlags" NumberOfComponents="3" format="ascii">1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1
                <DataArray type="Float64" Name="FaceArrayReal" NumberOfComponents="3" format="ascii">1. 2. 3. 1. 2. 3. 1. 2. 3. 1. 2. 3. 1. 2. 3. 1. 
                <DataArray type="Int32" Name="GROUP_HALF_FACE" format="ascii">0 1 0 1 0 1 0 1 0 1 0 1 0 1 0</DataArray>
            </CellData>
        </Piece>
    </PolyData>
</VTKFile>
```

Support des maillages polyédriques

- Utilisation de la nouvelle balise de maillage `meshes`
- Requiert le service VtkPolyhedralCaseMeshReader
  - Jeux de données de tests unitaires dans Arcane, testMeshPolyhedral-*.arc
- Travaux de consolidation en 2025
  - Parallélisme
  - Interopérabilité avec les briques Graph, Submesh
  - Pas d'AMR envisagé

```
<meshes>
  <mesh>
    <filename>faultx1_2x1x1.vtk</filename>
    <specific-reader name="VtkPolyhedralCaseMeshReader">
      <print-mesh-infos>true</print-mesh-infos>
      <print-debug-infos>false</print-debug-infos>
    </specific-reader>
  </mesh>
</meshes>
```

2. AMR par patch pour les maillages cartésiens

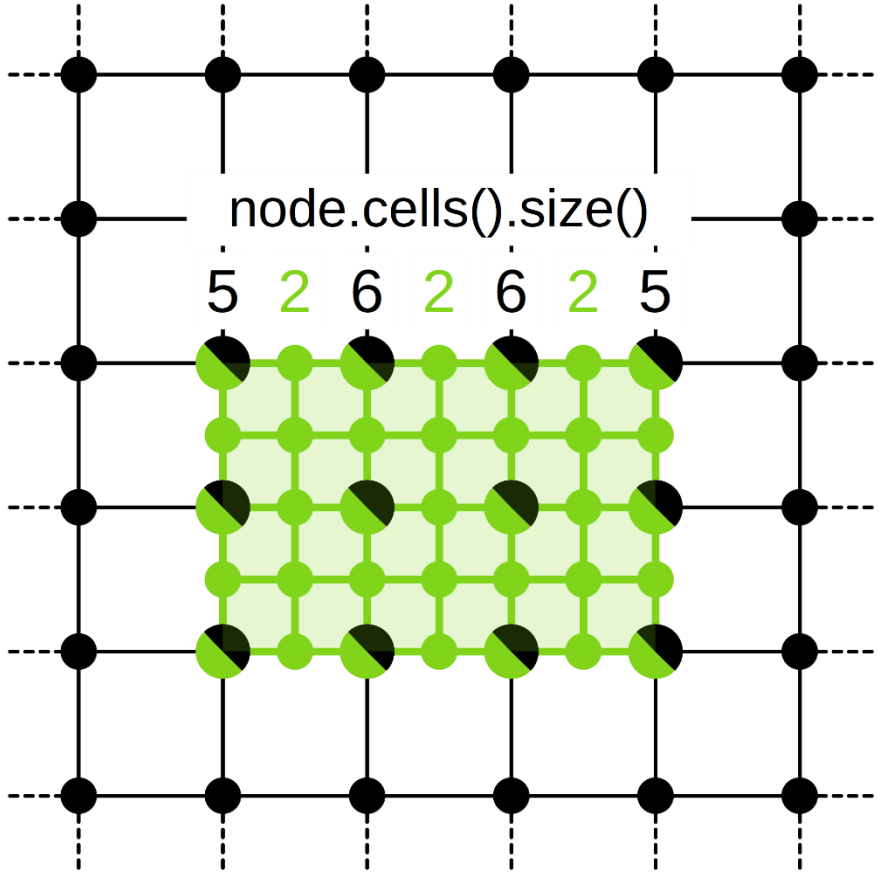
AMR par patch pour les maillages cartésiens

Motivations :
- Optimiser l'AMR pour les maillages cartésiens
- Dupliquer les nœuds entre les niveaux

Pour activer ce type d'AMR, il faut ajouter `amr-type='3'` dans le jeu de données (attribut de <mesh> ou <maillage>, compatible 2D et 3D) :

```
<mesh amr-type='3'>
   <meshgenerator>
     <cartesian>
       <nsd>2 2</nsd>
       <origine>0.0 0.0</origine>
       <lx nx='8' prx='1.0'>8.0</lx>
       <ly ny='8' pry='1.0'>8.0</ly>
     </cartesian>
   </meshgenerator>
 </mesh>
```

Niveau 0 : Patch 0
Niveau 1 : Patch 1



AMR pour les **maillages non structurés** sans duplication des nœuds.
(`amr-type='1'`)

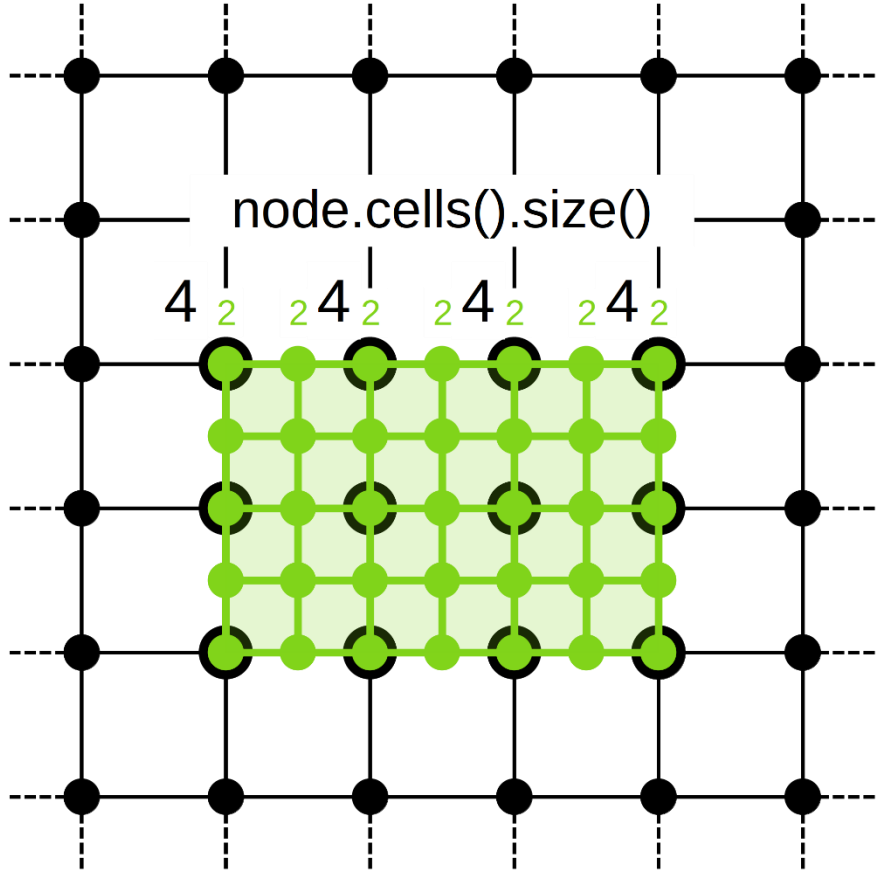
AMR par patch pour les maillages cartésiens

Motivations :
- Optimiser l'AMR pour les maillages cartésiens
- Dupliquer les nœuds entre les niveaux

Pour activer ce type d'AMR, il faut ajouter **amr-type='3'** dans le jeu de données (attribut de <mesh> ou <maillage>, compatible 2D et 3D) :

```
<mesh amr-type='3'>
   <meshgenerator>
     <cartesian>
       <nsd>2 2</nsd>
       <origine>0.0 0.0</origine>
       <lx nx='8' prx='1.0'>8.0</lx>
       <ly ny='8' pry='1.0'>8.0</ly>
     </cartesian>
   </meshgenerator>
 </mesh>
```

Niveau 0 : Patch 0
Niveau 1 : Patch 1

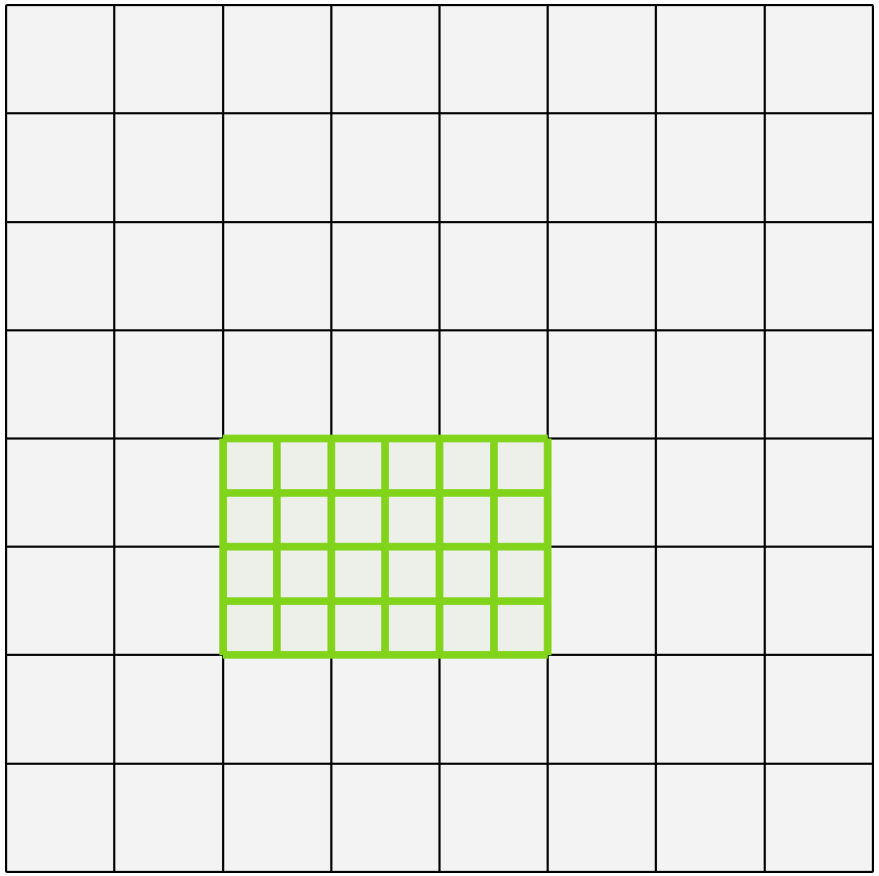


AMR pour les **maillages cartésiens**
avec duplication des nœuds.
(amr-type='3')

AMR par patch pour les maillages cartésiens

```
cartesian_mesh = ICartesianMesh::getReference(defaultMesh());

// cartesian_mesh->refinePatch({{position},{length}});
cartesian_mesh->refinePatch({{2.0, 2.0},{3.0, 2.0}});
cartesian_mesh->computeDirections();
```

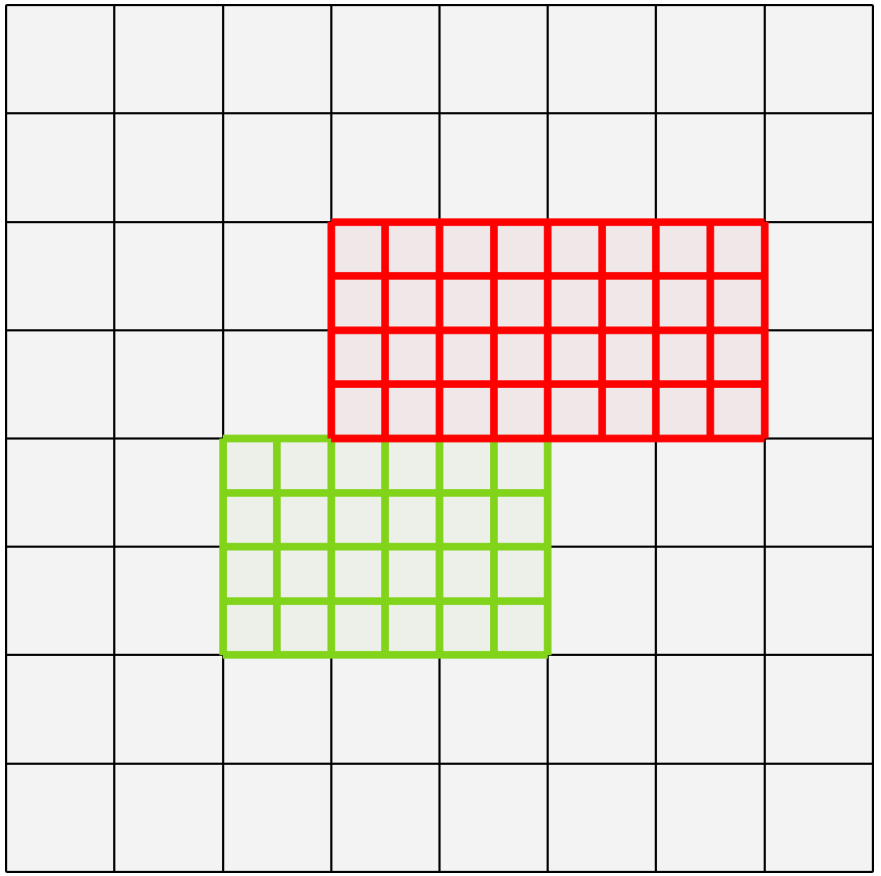


Niveau 0 : Patch 0
Niveau 1 : Patch 1

AMR par patch pour les maillages cartésiens

```
cartesian_mesh = ICartesianMesh::getReference(defaultMesh());

// cartesian_mesh->refinePatch({{position},{length}});
cartesian_mesh->refinePatch({{2.0, 2.0},{3.0, 2.0}});
cartesian_mesh->refinePatch({{3.0, 4.0},{4.0, 2.0}});
cartesian_mesh->computeDirections();
```

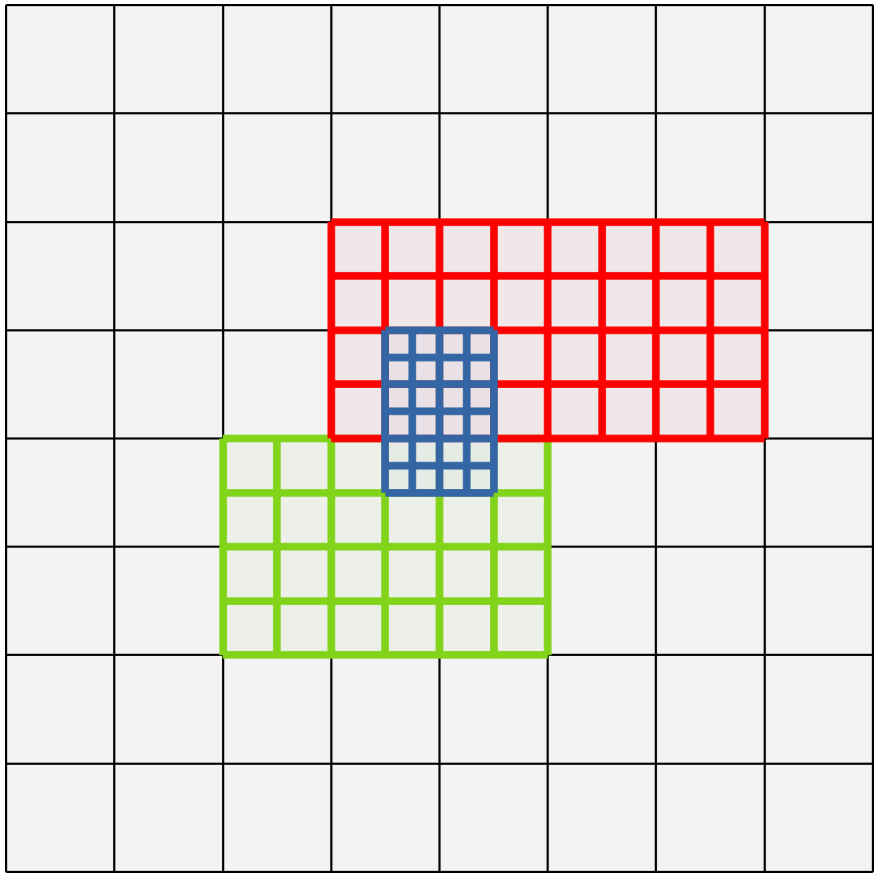


Niveau 0 : Patch 0
Niveau 1 : Patch 1 / Patch 2

AMR par patch pour les maillages cartésiens

```
cartesian_mesh = ICartesianMesh::getReference(defaultMesh());

// cartesian_mesh->refinePatch({{position},{length}});
cartesian_mesh->refinePatch({{2.0, 2.0},{3.0, 2.0}});
cartesian_mesh->refinePatch({{3.0, 4.0},{4.0, 2.0}});
cartesian_mesh->refinePatch({{3.5, 3.5},{1.0, 1.5}});
cartesian_mesh->computeDirections();
```



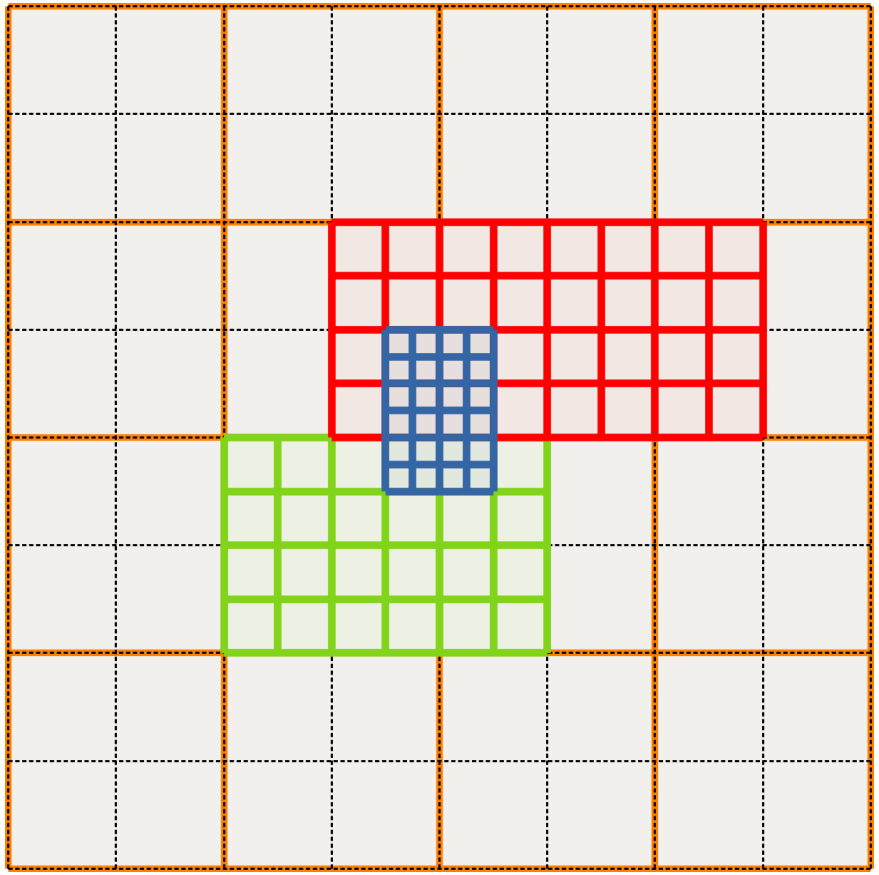
Niveau 0 : Patch 0
Niveau 1 : Patch 1 / Patch 2
Niveau 2 : Patch 3

AMR par patch pour les maillages cartésiens

```
cartesian_mesh = ICartesianMesh::getReference(defaultMesh());

// cartesian_mesh->refinePatch({{position},{length}});
cartesian_mesh->refinePatch({{2.0, 2.0},{3.0, 2.0}});
cartesian_mesh->refinePatch({{3.0, 4.0},{4.0, 2.0}});
cartesian_mesh->refinePatch({{3.5, 3.5},{1.0, 1.5}});
cartesian_mesh->computeDirections();

Ref<ICartesianMeshAMRPatchMng> coarser =
    CartesianMeshUtils::cartesianMeshAMRPatchMng(cartesian_mesh);
coarser->createSubLevel();
cartesian_mesh->computeDirections();
```



Niveau 0 : Patch 4
Niveau 1 : Patch 0
Niveau 2 : Patch 1 / Patch 2
Niveau 3 : Patch 3

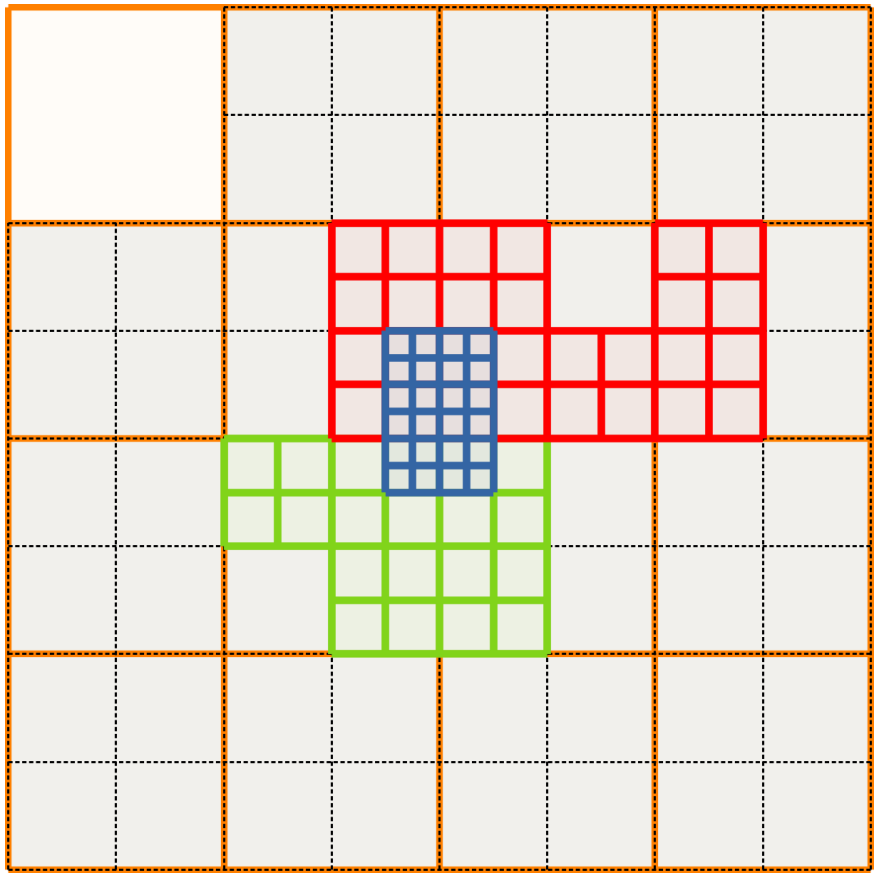
AMR par patch pour les maillages cartésiens

```
cartesian_mesh = ICartesianMesh::getReference(defaultMesh());

// cartesian_mesh->refinePatch({{position},{length}});
cartesian_mesh->refinePatch({{2.0, 2.0},{3.0, 2.0}});
cartesian_mesh->refinePatch({{3.0, 4.0},{4.0, 2.0}});
cartesian_mesh->refinePatch({{3.5, 3.5},{1.0, 1.5}});
cartesian_mesh->computeDirections();

Ref<ICartesianMeshAMRPatchMng> coarser =
    CartesianMeshUtils::cartesianMeshAMRPatchMng(cartesian_mesh);
coarser->createSubLevel();
cartesian_mesh->computeDirections();

cartesian_mesh->coarseZone({{0.0, 6.0},{2.0, 2.0}});
cartesian_mesh->coarseZone({{5.0, 5.0},{1.0, 1.0}});
cartesian_mesh->coarseZone({{2.0, 2.0},{1.0, 1.0}});
cartesian_mesh->computeDirections();
```

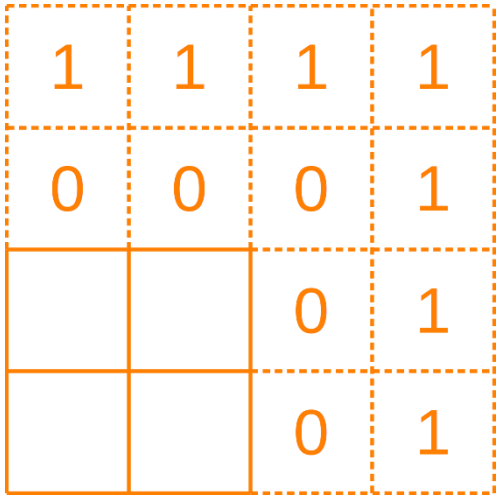


Niveau 0 : Patch 4
Niveau 1 : Patch 0
Niveau 2 : Patch 1 / Patch 2
Niveau 3 : Patch 3

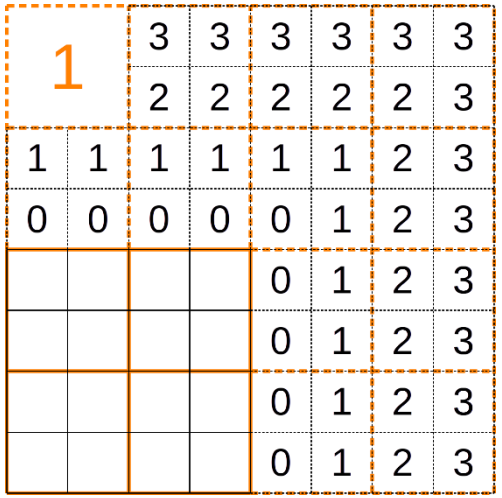
AMR par patch pour les maillages cartésiens

Prochaines étapes :

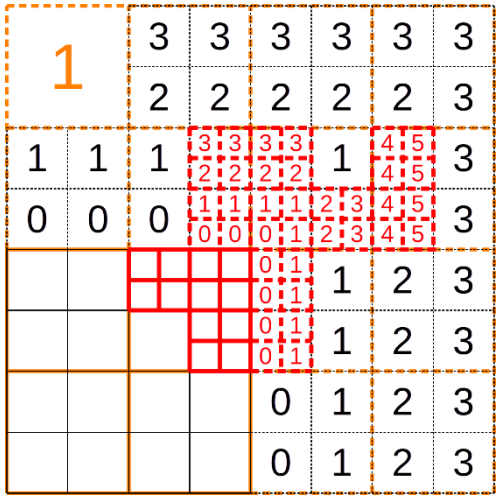
- Pouvoir créer et détruire des patchs pendant le calcul
- Eviter les patchs « irréguliers »
- S'assurer du support GPU
- Ecrire la documentation
- Option pour définir le nombre de couches de mailles fantômes sur les niveaux raffinés



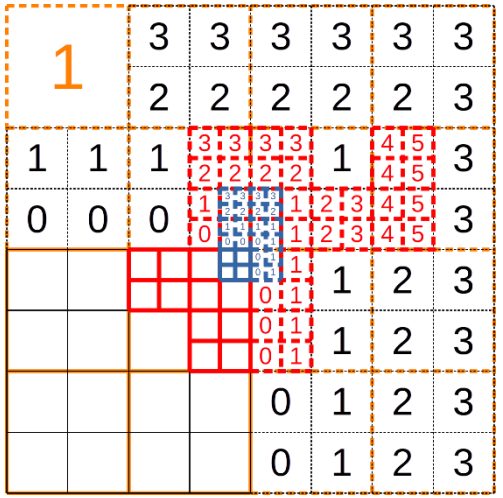
Niveau 0
2 couches de mailles fantômes



Niveau 1
4 couches de mailles fantômes



Niveau 2
8 couches de mailles fantômes



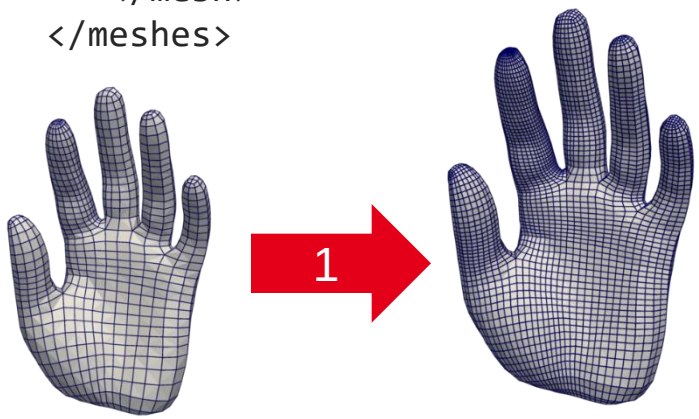
Niveau 3
16 couches de mailles fantômes

3. Raffinement de maillages par subdivision

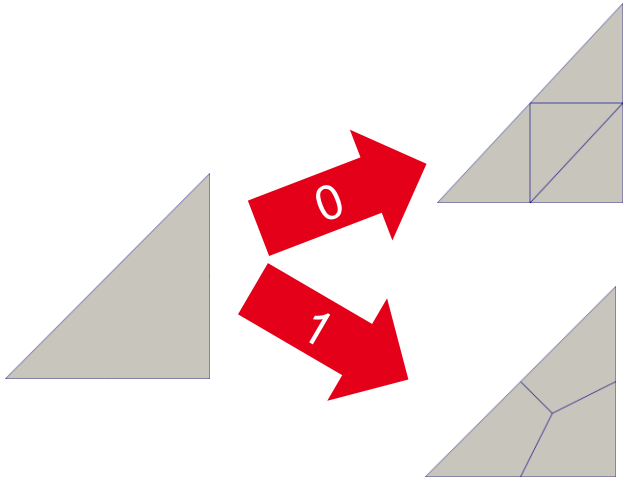
Raffinement de maillages par subdivision

Motivation:
- Eviter le stockage de fichiers de maillage de grande taille

```
<meshes>
   <mesh>
     <subdivider>
        <nb-subdivision>1</nb-subdivision>
        <different-element-type-output>
        0
        </different-element-type-output>
     </subdivider>
   </mesh>
</meshes>
```



nb-subdivision [0 ; N]



different-element-type-output [0 ; 1]

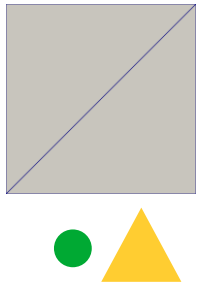
```xml
<?xml version="1.0"?>
<case codename="ArcaneTest" xml:lang="en" codeversion="1.0">
  <arcane>
    <title>Tube a choc de Sod</title>
    <timeloop>ArcaneHydroLoop</timeloop>
  </arcane>
  <meshes>
    <mesh>
    <filename internal-partition="true">sod.vtk</filename>
    <!--Notre contribution-->
    <subdivider>
      <nb-subdivision>1</nb-subdivision>
    </subdivider>
    <partitioner>MeshPartitionerTester</partitioner>
    <initialization>
      ...
    </initialization>
    </mesh>
  </meshes>
  <arcane-checkpoint>
    <do-dump-at-end>false</do-dump-at-end>
  </arcane-checkpoint>
  <!--Postprocessing-->
  <arcane-post-processing>
  ...
  </arcane-post-processing>
  <!-- Configuration du module hydrodynamique -->
  <simple-hydro>
    ...
  </simple-hydro>
</case>
```

Fichier .arc entier pour micro-hydro

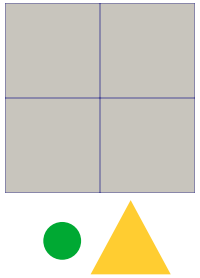
Motifs de raffinement présents aujourd'hui et validation

Testé avec ● Volume positif ◆ Groups ■ Micro hydro ▲ Arcane fem (**En cours**)

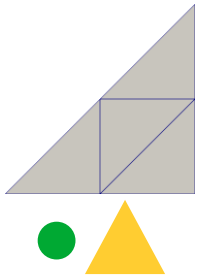
Quad to tri



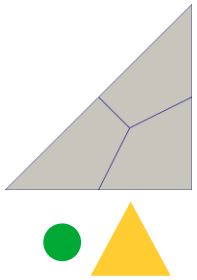
Quad to quad



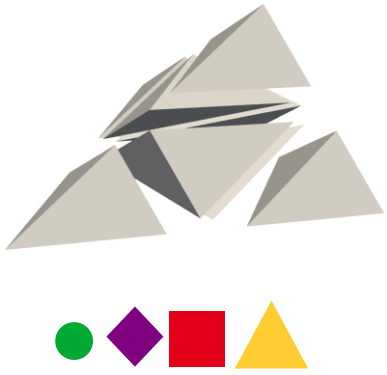
Tri to tri



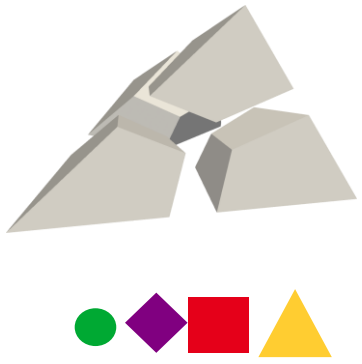
Tri to quad



Tet to tet



Tet to hex



Hex to tet24

Hex to hex

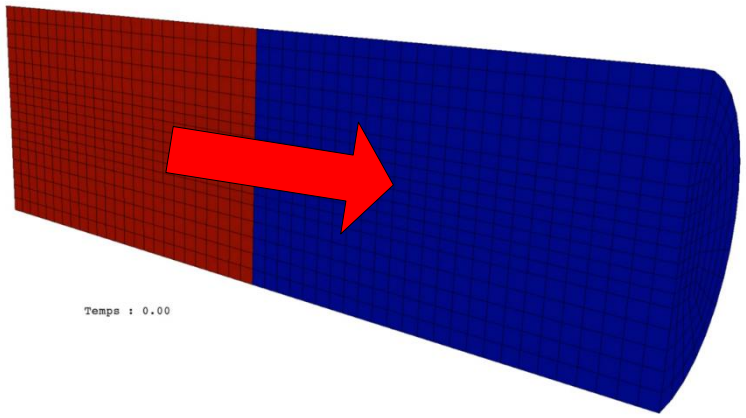
Hex to tet5

Le subdiviser ne gère pas ce motif

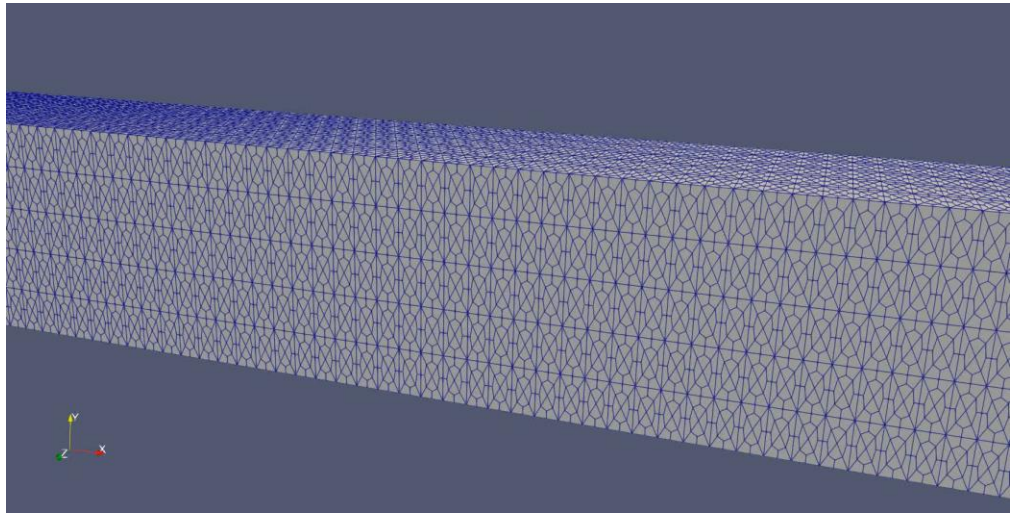
Détail micro hydro et validation supplémentaire

Détail validation métier :

- Hextohex
- Hextotet24 + tettotet
- Simple vérification si échec



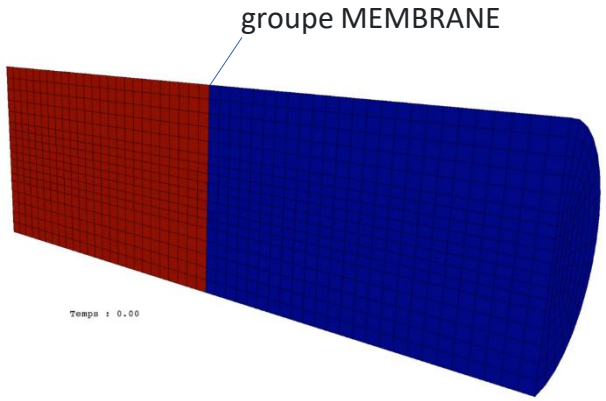
La simulation micro hydro de arcane



La simulation micro hydro de arcane avec raffinement hextotet24 + tettohex

Gestion des groups

- Les groupes peuvent permettre de stocker des informations pour la simulation
- Les nouveaux éléments générés doivent d'une certaine manière hériter de ces informations



- Les cellules(arc) filles doivent hériter des cellules mères
- Les faces (arc) filles doivent hériter des faces mères
- Pas pour les arrêtes (Non généré en 3D)
- Pas pour les nœuds (Quel sens au niveau physique ?)

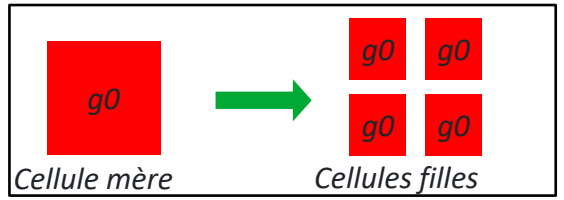


Fig. A

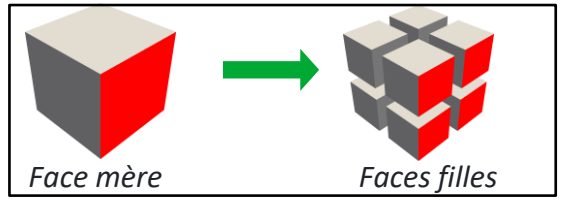
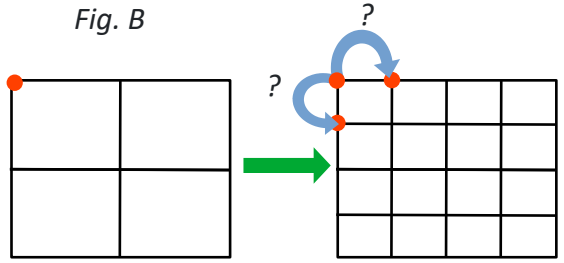


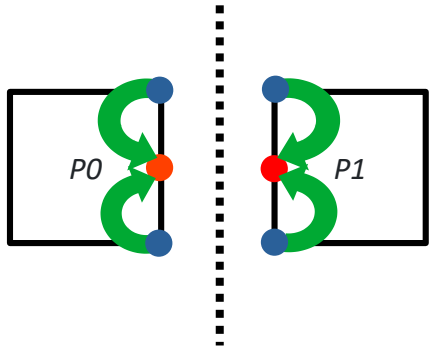
Fig. B

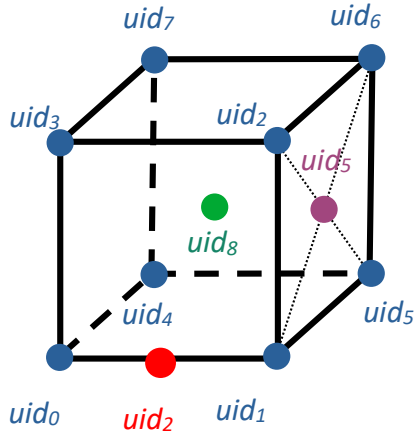


Génération des nouveaux uids et coordonnées

- Utilisation d'un hash de l'uid des anciens noeuds
- Même stratégie pour les uids des nouvelles Cells, Faces, Nœuds
  - Permet de générer toujours les mêmes uids (Debug)
  - Pas de garantie qu'il n'y ait pas de collision (En pratique pas de collision)

- Les coordonnées sont calculées à partir de la position des nœuds dans la cellule (pas de matrice de raffinement avec des coefficients réels)





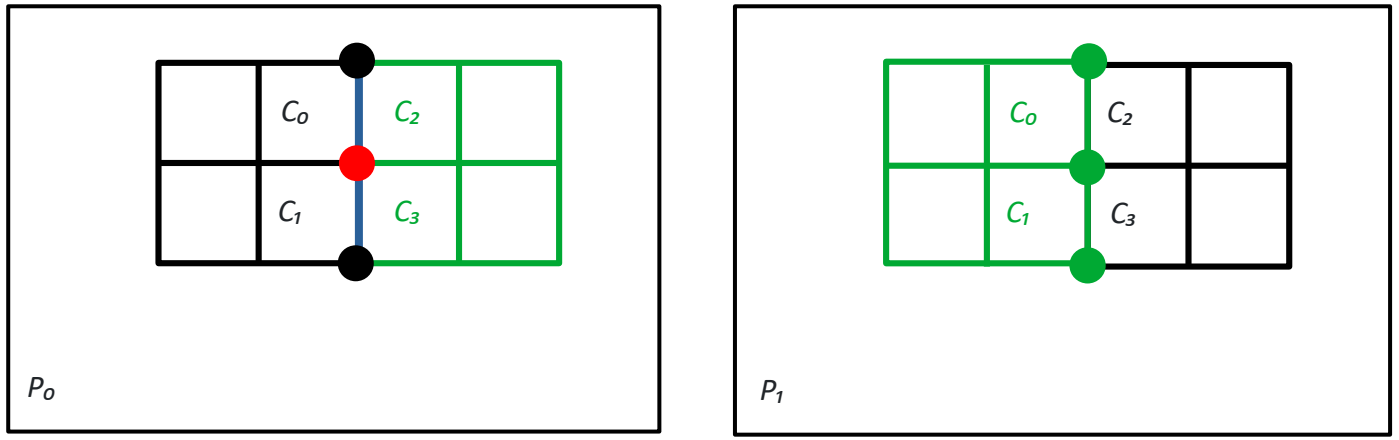
$uid_2 = generateHashUid(sort(\{uid_0, uid_1\}))$

$uid_5 = generateHashUid(sort(\{uid_0, uid_1, uid_2, uid_3\}))$

$uid_8 = generateHashUid(sort(\{uid_0, uid_1, uid_2, uid_3, uid_4, uid_5, uid_6, uid_7\}))$

Gestion des owners

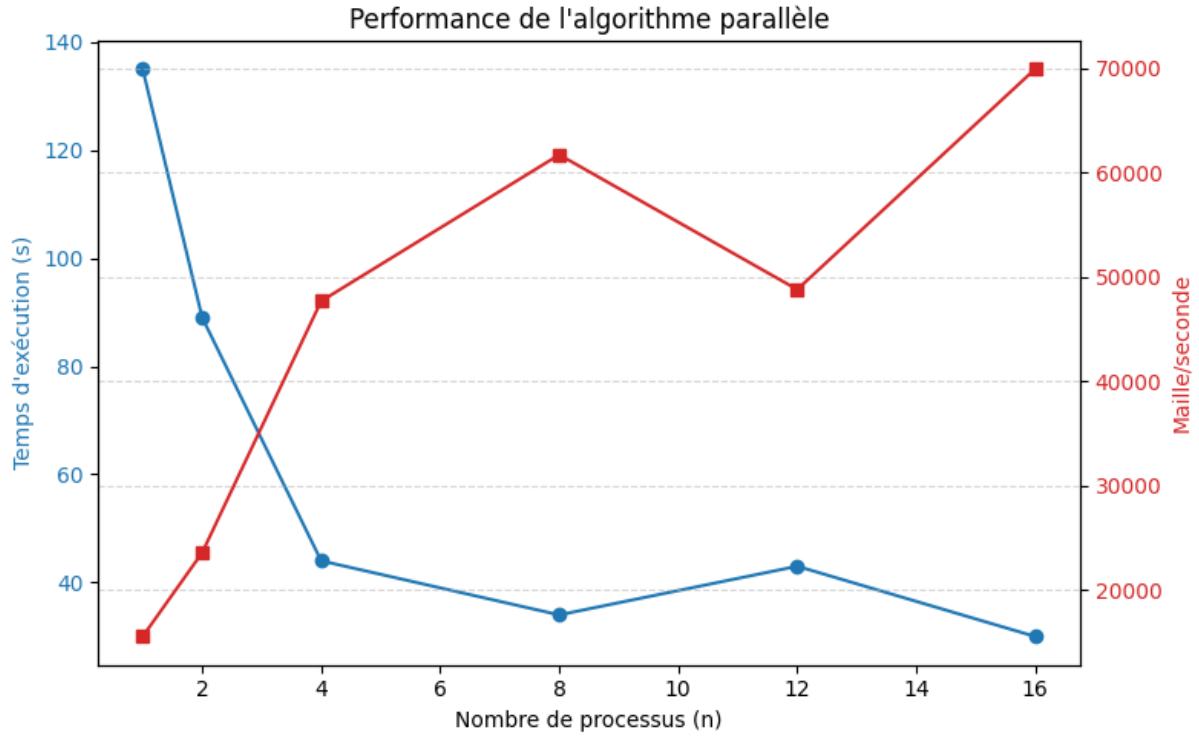
- Stratégie de gestion des owners pour les nouveaux éléments du maillage avec utilisation de ghostLayer
- Grâce au ghostLayer on a toujours les uids des cellules incidentes au nœuds et faces
- Un nœud appartient au propriétaire de la cellule incidente avec le plus petit uid (C_0)
- Une face appartient au propriétaire de la cellule incidente avec le plus petit uid ($C_0 < C_2$ et $C_1 < C_3$)
- Une cellule filles appartient au même processus que le parent



Résultats préliminaires des performances du subdiviseur

- Sur un ordinateur portable i7, 32go RAM
- Génération de ~2 millions de mailles à partir d'un maillage cartésien de 64 mailles*8^5 = 2 097 152 hexa
- Temps total du programme
- Profilage avec Valgrind
  - ~ 40 % du temps UpdateGhostLayer

n	t(secondes)	maille/seconde
1	135	15538
2	89	23569
4	44	47674
8	34	61696
12	43	48783
16	30	69922,1



Travaux en cours et perspectives

- En cours:
  - Continuer de valider les motifs
    - En utilisant arcanefem et microhydro
    - En effectuant plus de raffinement (cluster)
    - Sur plus de maillages

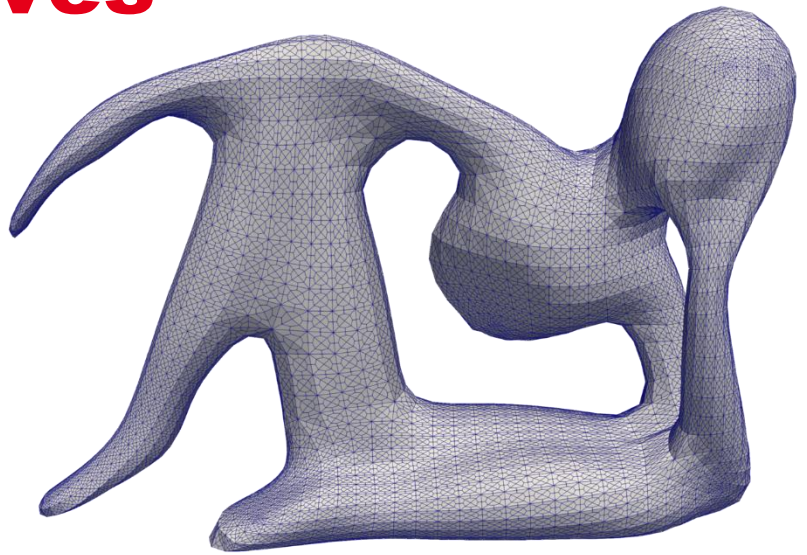


Fig 1. Maillage obtenu a partir d'hexalab et raffiné [HEX]

- Perspectives:
  - Changer la manière de numéroter les nouveaux sommets
    - Implémenter une méthode de renumérotation appelée après le subdiviseur
  - Performance : réduire ou enlever l'utilisation des ghostsLayers pour déduire les owners

[HEX] HexaLab.net: an online viewer for hexahedral meshes, Matteo Bracci, Marco Tarini, Nico Pietroni, Marco Livesu, Paolo Cignoni

4. Modification des options du jeu de données via les arguments de la commande de lancement

Modification des options du jeu de données via les arguments de la commande de lancement

Première méthode : remplacement de symboles

Inconvénient : Ajouter des symboles peut rendre le jeu de données inutilisable sans les bons arguments.

```
<?xml version="1.0"?>
<case codename="ArcaneTest" codeversion="1.0">
  <arcane/> <meshes/>

  <simple-hydro>
    <deltat-cp>@UneValeur@</deltat-cp>
    <array-real>@UnTableau@</array-real>
    <post-processor
      name="Ensight7PostProcessor"
      mesh-name="@MeshPostProcessor@">
      <fileset-size>@NbTimeInOneFile@</fileset-size>
      <binary-file>false</binary-file>
    </post-processor>
  </simple-hydro>
</case>
```

```
./SimpleHydro \
-A,MeshPostProcessor=Mesh1 \
-A,NbTimeInOneFile=10 \
-A,UneValeur=0.1 \
"-A,UnTableau=1.0 2.0 3.0" \
dataset_with_symbols.arc
```

Modification des options du jeu de données via les arguments de la commande de lancement

Deuxième méthode : remplacement via l'adresse de l'option

Avantage : fonctionne avec n'importe quel jeu de données.

Inconvénient : peut rendre la commande de lancement moins lisible.

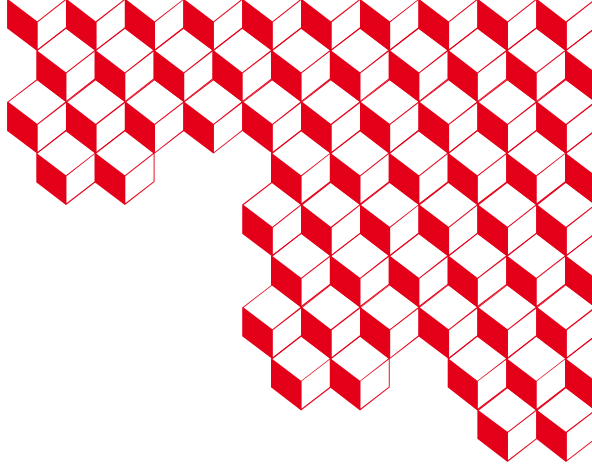
```
<?xml version="1.0"?>
<case codename="ArcaneTest" codeversion="1.0">
  <arcane/> <meshes/>

  <simple-hydro>
    <deltat-cp>0.1</deltat-cp>
    <array-real>1.0 2.0 3.0</array-real>
    <post-processor name="Ensight7PostProcessor">
      <fileset-size>10</fileset-size>
      <binary-file>false</binary-file>
    </post-processor>
  </simple-hydro>
</case>
```

```
./SimpleHydro \
-A,//simple-hydro/post-processor/@mesh-name=Mesh1 \
-A,//simple-hydro/post-processor/fileset-size=10 \
-A,//simple-hydro/deltat-cp=0.2 \
"-A,//simple-hydro/array-real=0.1 0.2 0.3" \
dataset.arc
```

5. Partie GPU

Partie GPU



Merci

https://github.com/arcaneframework/framework
https://discord.gg/tqdu7U2UwH