POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

MEC6616
Aérodynamique Numérique

Semaine 3 : Présentation du module de maillage

El Hadji Abdou Aziz NDIAYE

Hiver 2022

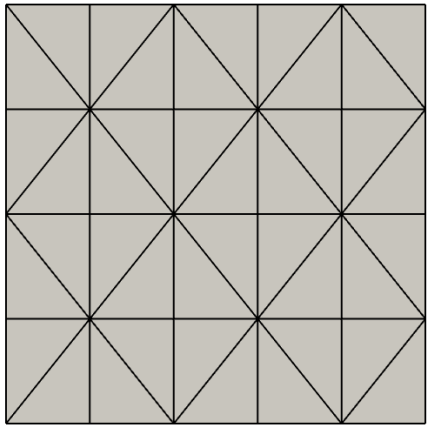
Organisation du module

- Le module comporte 3 classes: Mesh, MeshGenerator, MeshConnectivity
- **Mesh**: contient les données du maillage et de la connectivité
- **MeshGenerator**: permet de générer les maillages en utilisant l'API de Gmsh
- **MeshConnectivity**: calcule la connectivité du maillage généré

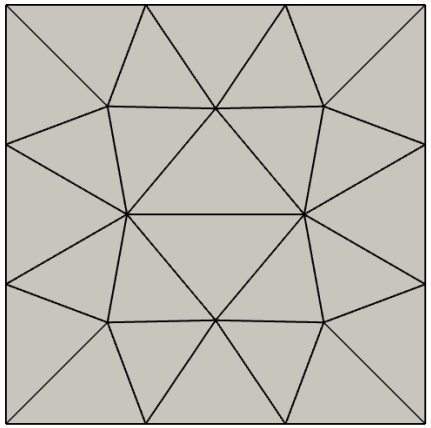
Génération des maillages

- Les maillages sont générés dans la classe MeshGenerator.
- Ils peuvent être non-structurés ou transfinis
- Pour chaque type de maillage, il est possible de contrôler le type des éléments : quadrilatère, triangle ou mixte (triangle + quadrilatère)
- Les méthodes de la classe retournent des objets de type Mesh contenant les données des maillages générés.

transfini (triangles)



non-structuré (triangles)



MeshGenerator
- verbose: bool
+ MeshGenerator(verbose) + rectangle(...): Mesh + back_step(...): Mesh + circle(...): Mesh + quarter_annulus(...): Mesh - generate_mesh_object(...): Mesh

Stockage des données des maillages

- La classe Mesh contient l'ensemble des données d'un maillage.
- Lors de la génération du maillage on stocke les informations suivantes:
  - Coordonnées des nœuds
  - Connectivité élément -> nœuds (Liste des nœuds qui compose chaque élément)
  - Liste des nœuds des faces situées aux frontières du domaine.

Mesh / Maillage
+ number_of_nodes: int + number_of_elements: int + number_of_faces: int + number_of_boundary_faces: int + node_to_xcoord: ndarray<number_of_nodes> double + node_to_ycoord: ndarray<number_of_nodes> double + element_to_nodes: ndarray int + element_to_nodes_start: ndarray<number_of_elements+1> int + boundary_faces_nodes: list of ndarray + face_to_nodes: ndarray<number_of_faces,2> int + face_to_elements: ndarray<number_of_faces,2> int + boundary_face_to_tag: ndarray<number_of_boundary_faces> uint8
+ Mesh(...) + get_node_to_xcoord(i_node: int): double + get_element_to_nodes(i_element: int): ndarray int ...

Stockage des données des maillages

- msh = mesher.RectGmsh([0.0, 1.0, 0.0, 1.0], {‘mesh_type’:’TRI’, ‘lc’:0,5})

Coordonnées des nœuds:

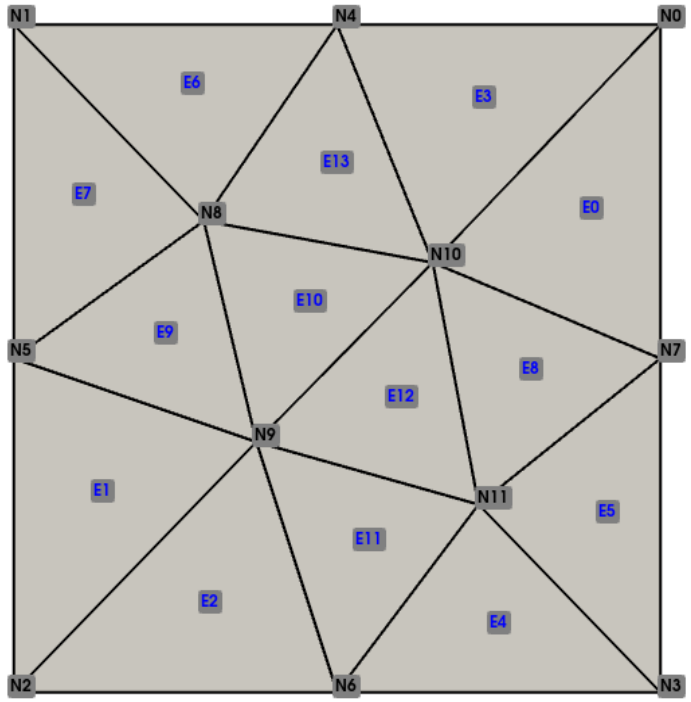
- msh.node_to_xcoord = array([1. , 0. , 0. , 1. , 0.5, 0. , 0.5, 1. , 0.29375, 0.375, 0.64791667, 0.71875])

Connectivité élément - nœuds:

- msh.element_to_nodes = array([7, 0, 10, 5, 2, 9, 2, 6, 9, 0, 4, 10, 6, 3, 11, 3, 7, 11, 4, 1, 8, 1, 5, 8, 7, 10, 11, 8, 5, 9, 8, 9, 10, 9, 6, 11, 10, 9, 11, 4, 8, 10])
- msh.element_to_nodes_start = array([0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42])
- Par exemple, pour obtenir les nœuds de l’élément 10, il faut faire:
  - start = msh.element_to_nodes_start[10] # = 30
  - fin = msh.element_to_nodes_start[11] # = 33

  nœuds_elem10 = msh.element_to_nodes[start:fin] # [8,9,10]

Maillage



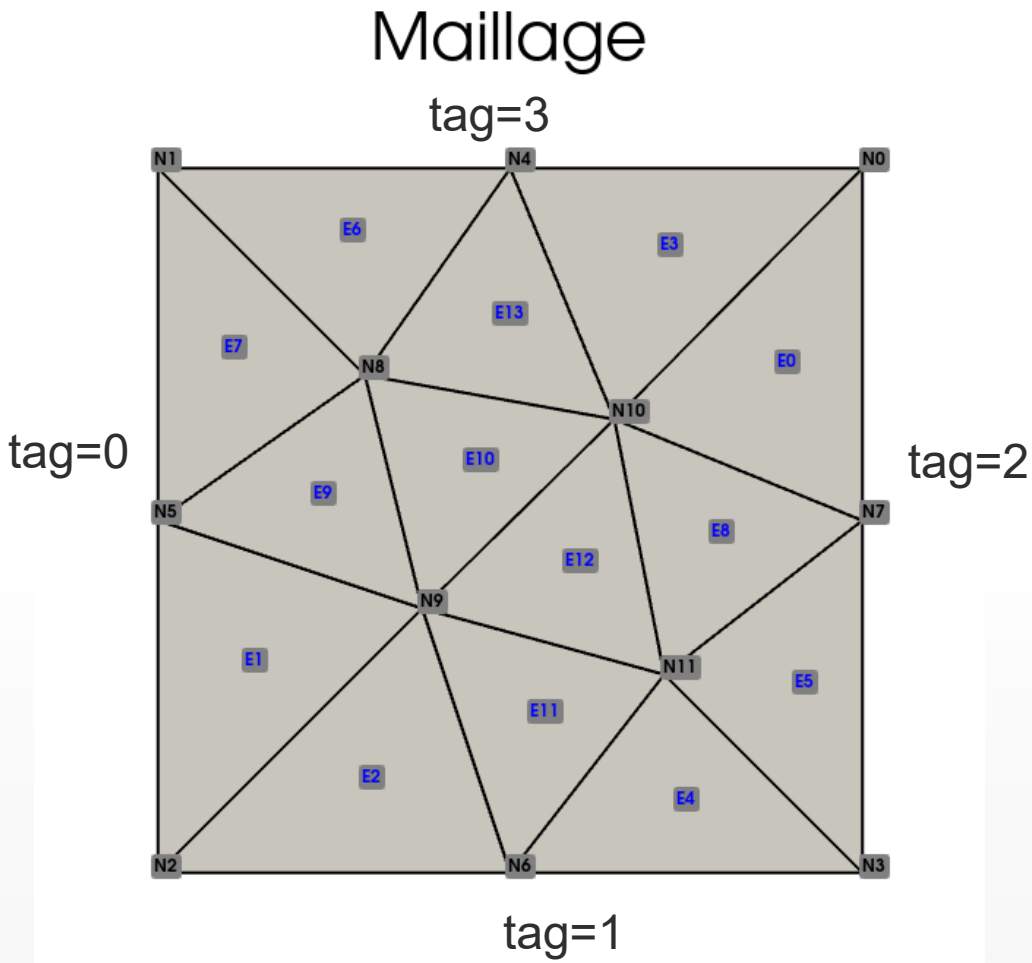
Stockage des données des maillages

Nœuds des faces frontières

- msh.boundary_faces_nodes = [array([1, 5, 5, 2]), array([2, 6, 6, 3]), array([3, 7, 7, 0]), array([0, 4, 4, 1])]

Exemple pour tag = 1:

- msh.boundary_faces_nodes[tag] = array([2, 6, 6, 3])
- On a deux faces pour la frontière 1:
  - F_a : noeud 2 et noeud 6
  - F_b : noeud 6 et noeud 3

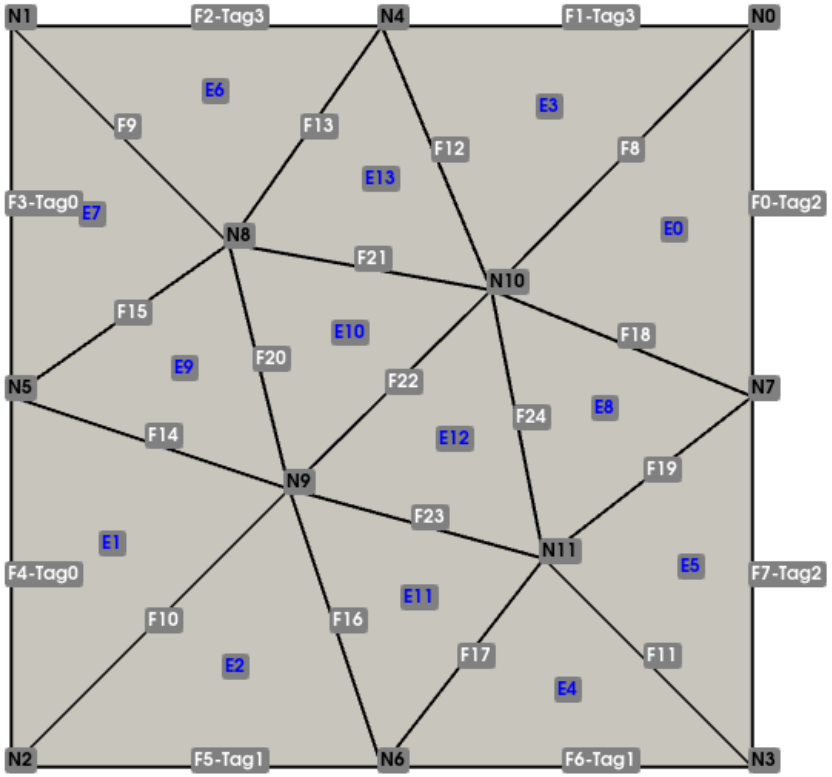


Connectivité du maillage

- Avec la table de connectivité élément => nœuds et la liste des nœuds des faces frontières, il est possible de construire les autres tables de connectivité.
- Ces tables sont:
  - **Face => nœuds**: liste des (deux) nœuds de chaque face
  - **Face => éléments**: liste des (deux) éléments voisins à chaque face
  - **Face frontière => tag**: tag de chaque face frontière
- La classe MeshConnectivity permet de construire ces tables.
- Notes importantes:
  - Les faces frontières correspondent aux faces d'index 0 à $number\_of\_boundary\_faces - 1$
  - Dans la connectivité Face=>éléments, le premier élément correspond au voisin situé à gauche de la face.
  - Les faces frontières n'ont pas de voisin à droite. La valeur -1 est assignée à la place.

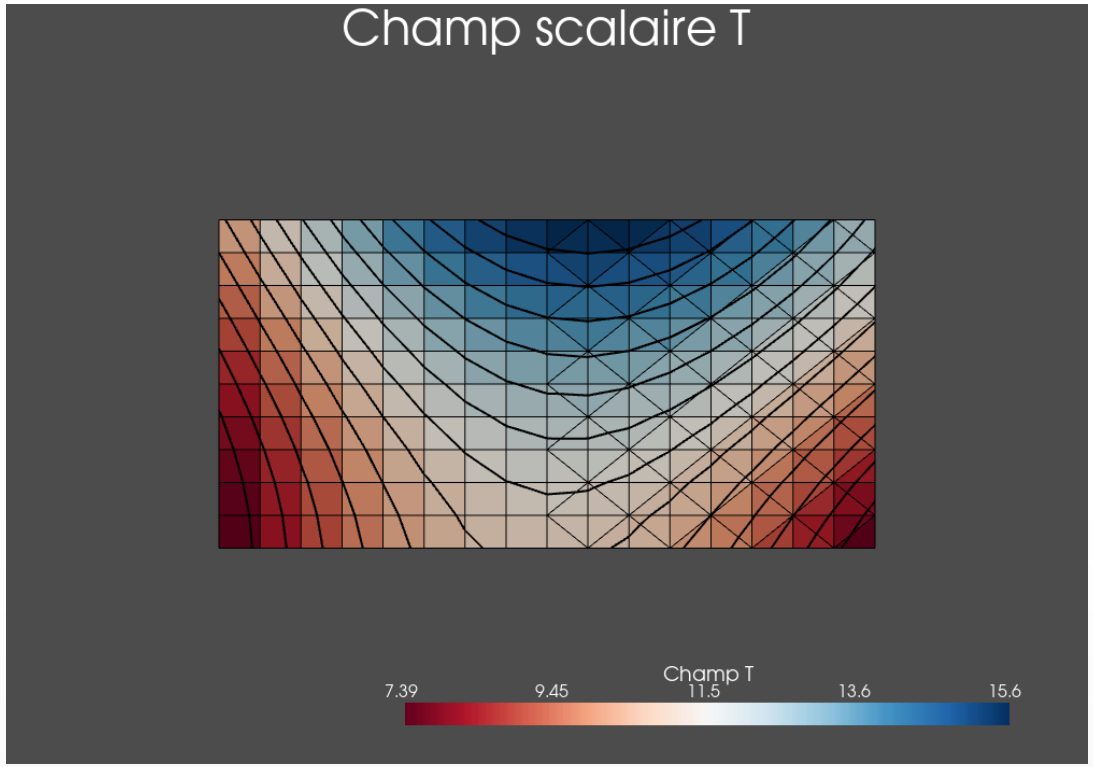
Connectivité du maillage

- face_to_nodes = array([[7, 0], [0, 4], [4, 1], [1, 5],
  [5, 2], [2, 6], [6, 3], [3, 7], [0, 10], [1, 8], [2, 9],
  [3, 11], [4, 10], [4, 8], [5, 9], [5, 8], [6, 9], [6, 11],
  [7, 10], [7, 11], [8, 9], [8, 10], [9, 10], [9, 11], [10, 11]])
- face_to_elements = array([[0, -1], [3, -1], [6, -1], [7, -1],
  [1, -1], [2, -1], [4, -1], [5, -1],   [0, 3],   [6, 7],   [1, 2],
  [4, 5], [3, 13], [6, 13], [1, 9],   [7, 9],   [2, 11], [4, 11],
  [0, 8], [5, 8], [9, 10], [10, 13], [10, 12], [11, 12], [8, 12]])
- Pour la table boundary_face_to_tag voir figure



Visualisation des maillages

- On utilise le module pyvista pour visualiser les maillages.
- Les figures des maillages de cette présentation ont été obtenu avec pyvista
- La fonction `prepare_data_for_pyvista()` de la classe MeshPlotter permet de faciliter l'utilisation de pyvista
- Le module pyvistaqt permet de générer les figures sans arrêter l'exécution du code.

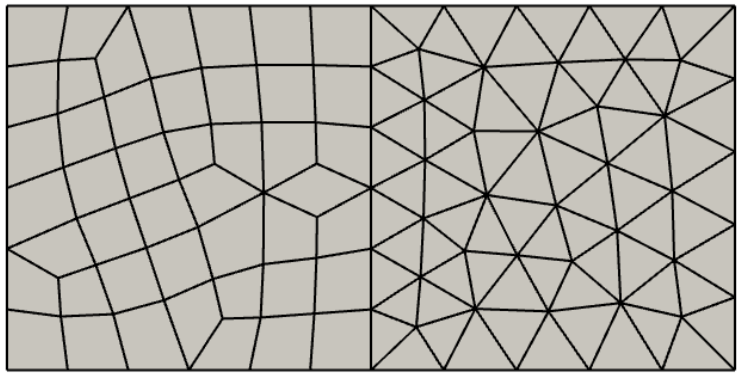


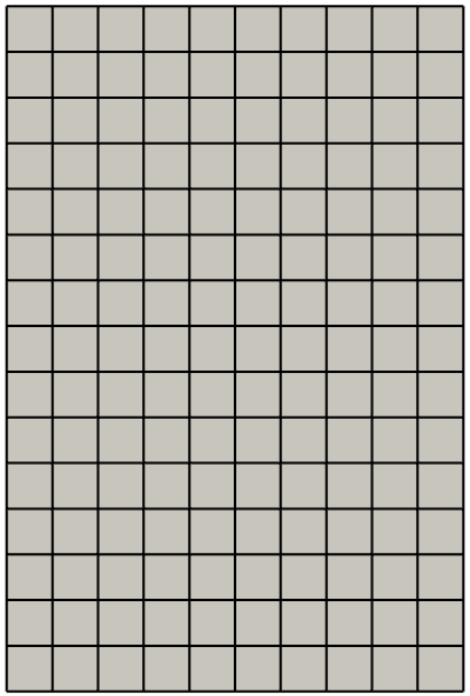
Exemples de maillages

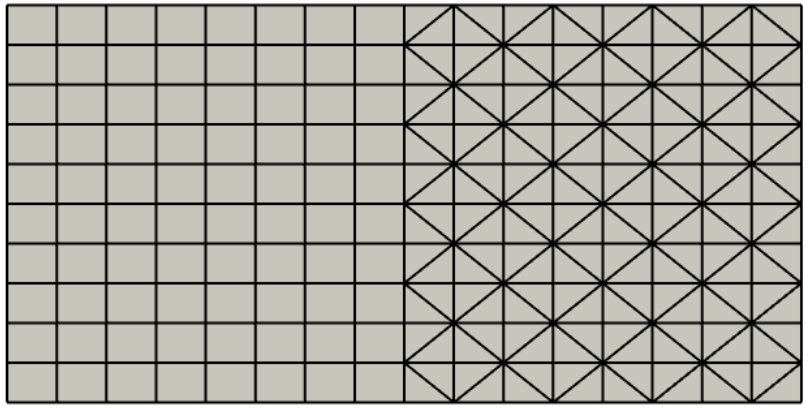
- Vous pouvez exécuter le fichier exemple qui accompagne le module pour tester les différents types de maillages.
- Il faut pour cela, installer les modules suivants au préalable:
  - **GMSH**: pip install gmsh
  - **Pyvista**: pip install pyvista
  - **Pyvistaqt**: pip install pyvistaqt

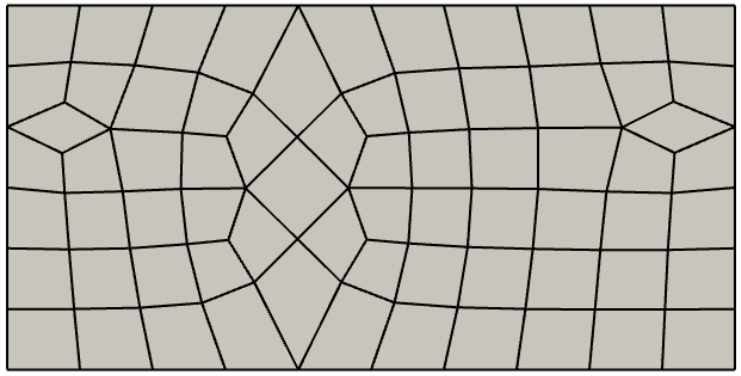
Exemples de maillages

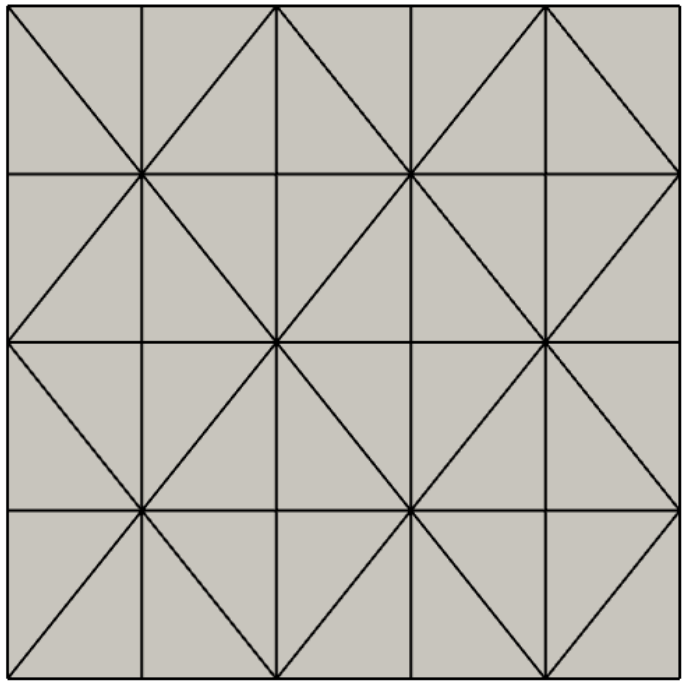
- Géométrie du rectangle





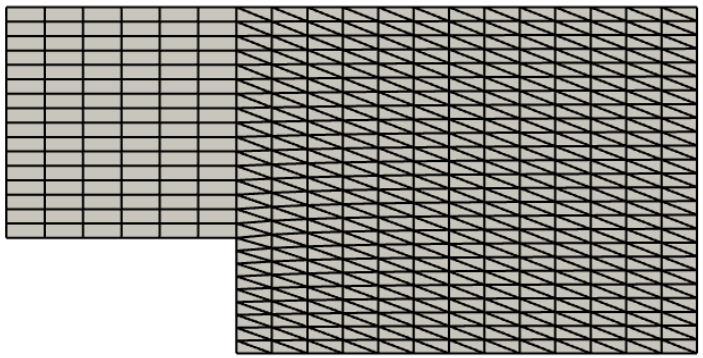


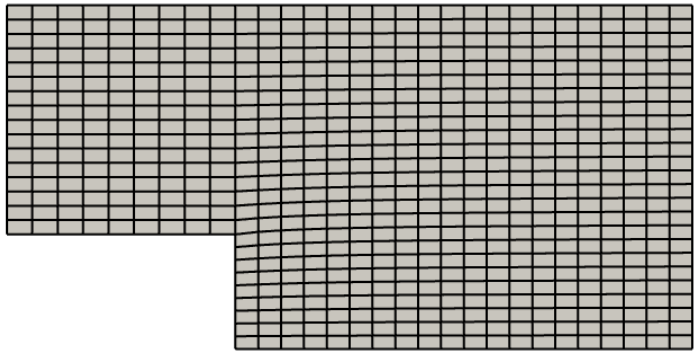


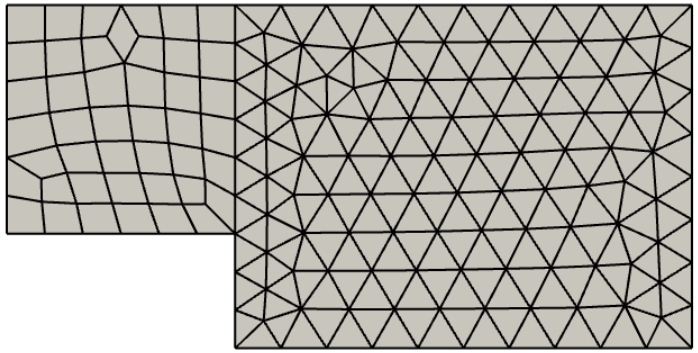


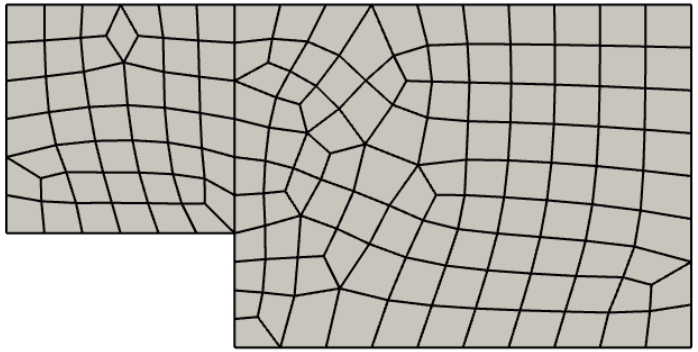
Exemples de maillages

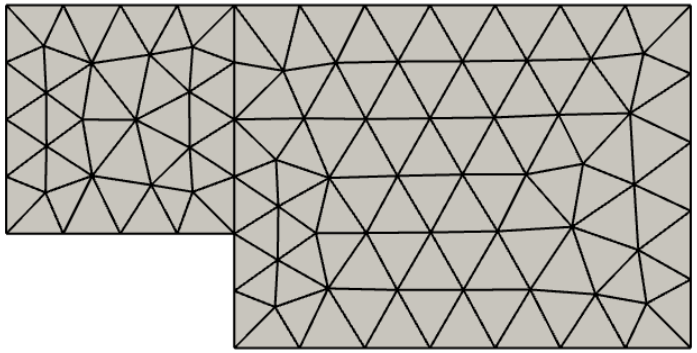
- Géométrie de la marche descendante

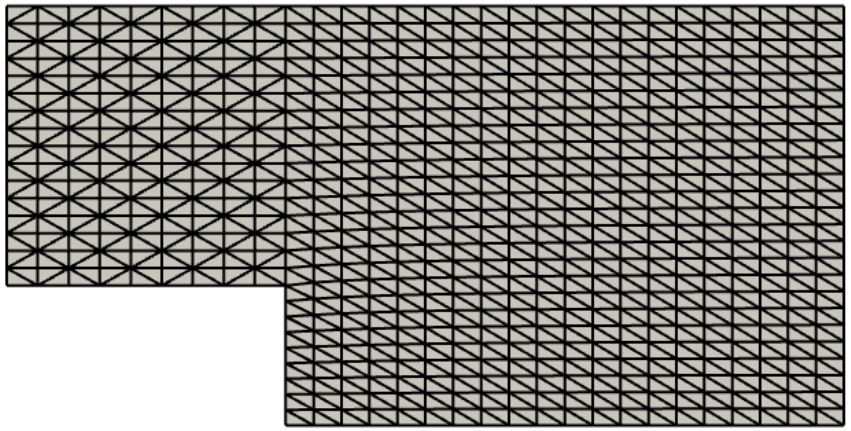






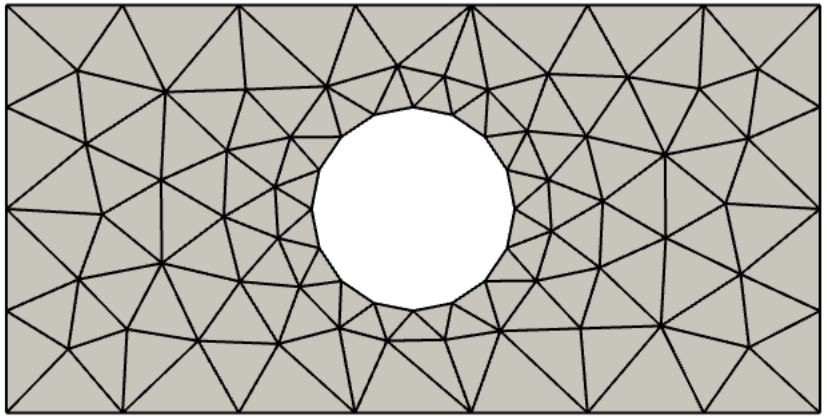


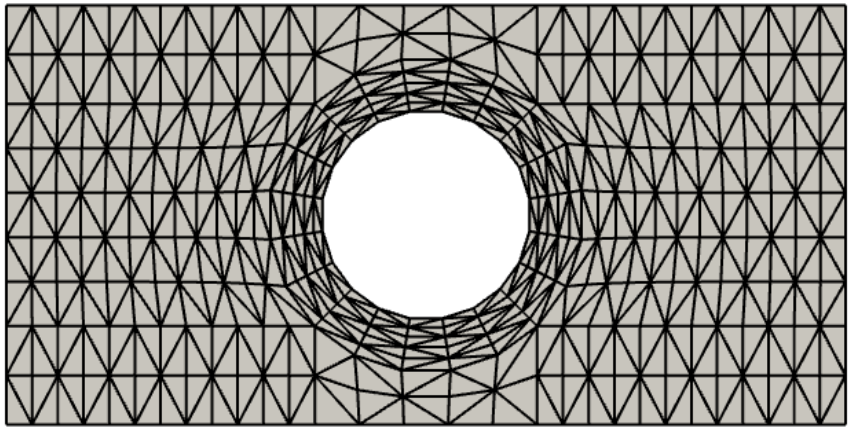


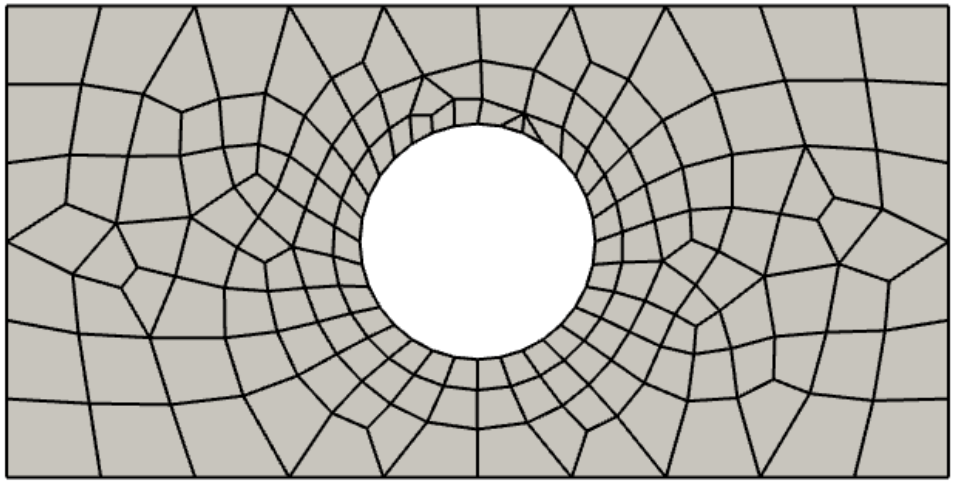


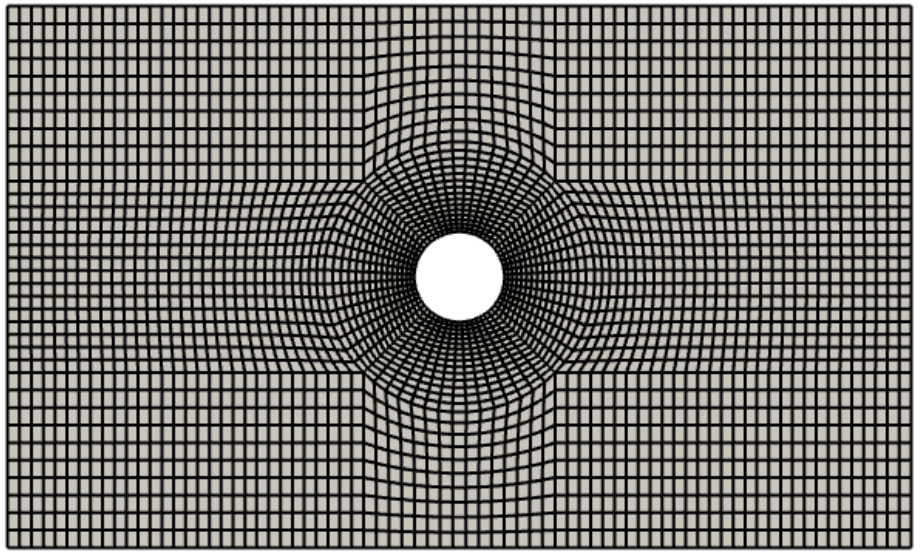
Exemples de maillages

- Géométrie du cercle









Exemples de maillages

- Géométrie du quart d'anneau