Promotion 2014/2015

Université HL Batna

N d'ordre : 03/DM/FST/UHLB/2015

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

**UNIVERSITE HADJ LAKHDAR BATNA**
**Faculté de Technologie**
**Département de Mécanique**

# MEMOIRE DE FIN D'ETUDES

Présenté pour obtenir le diplôme de

## MASTER

**Spécialité : ENERGETIQUE**

Par :

**MARAF Salah Eddine**

---

# EVALUATION DU CHAMP AERODYNAMIQUE AUTOUR D'UN OBJET 3D

---

*Soutenu le 26/06/2015*

Encadré par : **Dr. L. MESSAOUDI**

# *DÉDICACE*

*Je dédie ce travail*

*à mon père, à ma mère,*
*à toute ma famille*
*et à tous mes ami(e)s.*

*S. Maaraf.*

# *REMERCIEMENTS*

*Je tiens à remercier . . .*

*Je tiens aussi à remercier, à l'avance, les membres du jury qui seront désignés pour évaluer ce travail.*

*Je remercie également tous les . . .*

*S. Maaraf.*

# TABLE DES MATIÈRES

# LISTE DES FIGURES

# LISTE DES TABLEAUX

# INTRODUCTION GÉNÉRALE

L'utilisateur typique d'ordinateurs possède des quantités de logiciels qu'il a acquis de par le passé et qu'il n'utilise plus désormais. Il a peut-être acheté un ordinateur plus performant ou changé de constructeur, ce qui a empêché ses programmes de continuer à fonctionner. Les logiciels sont peut-être devenus obsolètes, ou ont cessé de répondre à ses besoins. Il a peut-être acheté deux ordinateurs ou plus, et ne souhaite pas débourser quoi que ce soit pour obtenir une deuxième copie du même logiciel. Quelle qu'en soit la raison, le logiciel qui lui a coûté de l'argent voici quelques années ne remplit plus son rôle. Cette situation est-elle inévitable ? Si tel était le cas, vous continueriez sans doute à utiliser les logiciels que vous avez achetés il y a des années. Ce sont là quelques-uns des droits que l'Open Source vous garantit. La définition de l'Open Source est une déclaration des droits de l'utilisateur d'ordinateur. Elle définit certains droits que la licence d'un logiciel doit vous garantir pour pouvoir être qualifiée d'Open Source. Ceux qui ne proposent pas des programmes Open Source trouvent que la concurrence est de plus en plus rude avec ceux qui ont compris que les utilisateurs commencent à apprécier et à exiger les droits qu'ils auraient dû avoir depuis toujours. Des programmes comme ceux qui constituent le système d'exploitation Gnu/Linux et le navigateur web de Netscape ont acquis une grande popularité, en prenant la place de logiciels couverts par des licences plus restrictives. Les sociétés qui utilisent du logiciel Os profitent des avantages liés à un modèle de développement très réactif, souvent mis en place par plusieurs sociétés qui coopèrent, et dont une grande partie du travail est fournie par des individuels qui ont tout simplement besoin d'une amélioration qui réponde mieux à leurs besoins. Les volontaires qui ont créé des produits comme Linux n'existent, et les sociétés ne peuvent coopérer, que grâce aux droits fournis par l'Open Source. Le programmeur moyen se sentirait lésé s'il investissait énormément de travail dans un programme, pour constater que le propriétaire de ce dernier se contente de vendre la version améliorée sans rien lui proposer en retour. Mais ces mêmes programmeurs n'ont pas peur d'apporter des

contributions à l'Open Source, car ils savent que les droits suivants leur sont alors garantis : le droit de faire des copies du programme, et de distribuer ces copies. le droit d'accéder au code source du logiciel, un préliminaire nécessaire pour pouvoir y apporter des modifications, le droit d'améliorer le programme. Ces droits comptent pour celui qui contribue à améliorer des logiciels car ils maintiennent tous les développeurs au même niveau. Quiconque le souhaite a le droit de vendre un programme Open Source, aussi les prix seront-ils bas et le développement destiné à conquérir de nouveaux marchés sera-t-il rapide. Quiconque investit de son temps à construire des connaissances sous la forme d'un programme Open Source peut fournir de l'aide sur ce dernier, et cela fournit aux utilisateurs la possibilité de mettre en place leur propre assistance technique, ou de choisir parmi des assistances techniques en concurrence les unes avec les autres. Tout programmeur peut spécialiser un programme Open Source pour le rendre utilisable dans des marchés spécifiques afin d'atteindre de nouveaux consommateurs. Ceux qui se lancent dans de telles entreprises ne sont pas tenus d'acquitter des royalties ou la moindre licence. La raison de la réussite de cette stratégie vient du fait que l'économie de l'information est fondamentalement différente de l'économie qui règle la consommation des autres produits. On peut copier une information telle qu'un programme d'ordinateur pour un prix dérisoire. L'électricité qu'il en coûte ne dépasse par l'eurocent, pas plus que l'utilisation de l'équipement nécessaire à cet acte. En comparaison, on ne peut pas recopier une miche de pain sans dépenser une livre de farine [1].

Linux est un système d'exploitation (un logiciel système au même titre que WINDOWS de MICROSOFT, NETWARE de NOVELL, MAC OS d'APPLE, SOLARIS de SUN, etc....) qui joue le rôle d'interface entre les composants matériels d'un ordinateur, les logicielles et les utilisateurs. Linux peut être utilisé comme un serveur ou comme station de travail. LINUX est multi-tache, multi-utilisaeur et multi-processeur, multi-plateformes. LINUX est un logiciel OPEN SOURCE (c'est-à-dire libre, ouvert et gratuit...) qui a été développé gracieusement par une communauté d'informaticien reliés entre eux par internet. LINUX existe depuis 1991 a l'initiative de Linus Torvalds. LINUX est un système puissant, robuste, modulaire, souple et évolutif. Il existe des dizaines de distributions Linux dont les principales sont [3] :

| | | | |
|---|---|---|---|
| - Debian | - Fedora | - Slackware | - Mandriva |
| - Linux Mint | - OpenSUSE | - Red Hat | - Ubuntu |

Parmi les distributions Linux, il existe aussi des distributions destinées spécialement aux scientifiques telles que *Scientific Linux* et *CAELinux*. Cette dernière, que nous avons choisi pour notre travail, a été dévelopée par EDF (Electricité de France) et le CEA (Conservatoir l'Energie Atomique) et dès son installation, tous les programmes et codes fournis gratuitement sont disponibles et prêt à être utilisés.

On trouve dans cette distribution une grande variété de programmes (CAO, CAM, CAE / FEA / CFD) destinés à l'ingénierie afin de concevoir, mailler et simuler des phénomènes physiques couplés ou non linéaires et non-linéaires dans divers domaines fluides et solides ainsi que plusieurs programmes de visualisation des résulats. Parmi ces logiciels on distingue :

- CAD / FAO : FreeCAD, LibreCAD, PyCAM et la plate forme Salome-Meca.
- Solveurs éléments finis : Code_Aster, Code_Saturne et Elmer
- Solveur volumes finis : OpenFOAM.
- Mailleurs : Gmsh, MeshLab, Ingrid, NetGen, BlockMesh et SnappyHexMesh.
- Outils de génération G -Code pour le fraisage CNC, de prototypage avec PyCAM ou l'impression 3D avec Cura.
- Outils de visualisation : Salome, Gmsh, ParaView, QtiPlot, Grace et LabPlot.

Avec les outils et solveurs open-source intégrés à CAELinux, on peut pratiquement simuler la quasi-totalité des phénomènes physiques les plus complexes : non linéaire, thermo-mécanique, couplés, dynamique, interaction fluide-structure, sismique/non-linéaire dynamique explicite, contacts, visco-plasticité, dynamique des fluides, échange de chaleur, transfert de chaleur par convection et rayonnement [4].

On note que la plate forme Salome-Meca est une suite contenant plusieurs logiciels et qui permet de traiter un problème physique depuis la conception jusqu'à la visualisation des résultats et plus encore.

L'ojectif de notre travail est de pouvoir utiliser ces outils open source gratuits pour résoudre un problème d'aérodynamique, à savoir l'étude de l'écoulement autour d'un objet tridimensionnel appelé *corps Ahmed* (Ahmed Body). Pour effectuer cette étude, plusieurs méthodes s'offrent à nous. Nous avons choisi d'utiliser Salome-Meca pour la conception 3D, BlockMesh et SnappyHexMesh pour le maillage, le code OpenFOAM basé sur la méthode des volumes finis comme solveur et les outils ParaView et QtiPlot pour la visualisation. N'oublions pas, bien sûr, le code LaTex avec l'interface Lyx utilisés pour écrire ce manuscrit.

Ce document est structuré de la manière suivante :

Nous commencerons, dans le premier chapitre, par introduire le corps Ahmed et sa conception avec Salome. Nous passerons ensuite au maillage avec Salome et SnappyHexMesh dans le second chapitre. Après avoir détaillé la manière d'utiliser le solveur OpenFoam, le troisième chapitre est consacré à la simulation de l'écoulement tridimensionnel dont les résultats sont présentés au quatrième chapitre.

Vu que l'utilisation de l'open source est une première au niveau de notre département, nous avons essayé de donner le plus de détails possibles pour faciliter l'utilisation de futures travaux et afin de ne pas allourdir inutilement ce document, nous avons mis ces détails en annexe.

# CHAPITRE 1

# CONCEPTION DU CORPS AHMED

## 1.1 Introduction

Le modèle Ahmed (dit *Ahmed Body*) a reçu une attention globale de la communauté scientifique concernée par la dynamique des fluides et cela depuis sa première apparition lorsque Ahmed at al. [1] l'ont utilisé dans une soufflerie pour reproduire des caractéristiques d'écoulements que l'on retrouve sur une voiture. Le modèle a aussi été étudié par le «European Research Community On Flow, Turbulence, And Combustion» (ERCOF- TAC) afin de valider différents codes d'analyse numérique des fluides (CFD 2). En fait, l'association ERCOFTAC a tenu des ateliers sur la simulation du modèle Ahmed à plus d'une reprise. Le premier objectif de ce projet de maîtrise a été de créer un cas de CFD qui reproduit les caractéristiques de l'écoulement sur le corps Ahmed et surtout son coefficient de traînée. Reproduire le coefficient de traînée expérimental a été un défi pour tous les chercheurs en CFD qui se sont penchés sur le modèle Ahmed car il est difficile de reproduire numériquement le début et la fin de la zone décollée que l'on retrouve sur la paroi oblique arrière du véhicule.

## 1.2 Géométrie

Une géométrie communément appelée le modèle Ahmed et qui a été originalement conçue par Ahmed [1] a été choisie en tant que modèle de départ pour les simulations car elle est couramment utilisée dans la littérature et des résultats expérimentaux [1, 10] et numériques [5, 6, 7, 8] d'écoulements sur ce modèle ont été publiés par différents auteurs. La figure (1.1) ci-dessous donne la géométrie et les dimensions en millimètres du corps Ahmed ; l'angle $\varphi$ est appelé *angle oblique,* c'est l'angle entre l'horizontale (plan x-z) et la paroi oblique arrière du véhicule dont le devant se trouve à gauche.

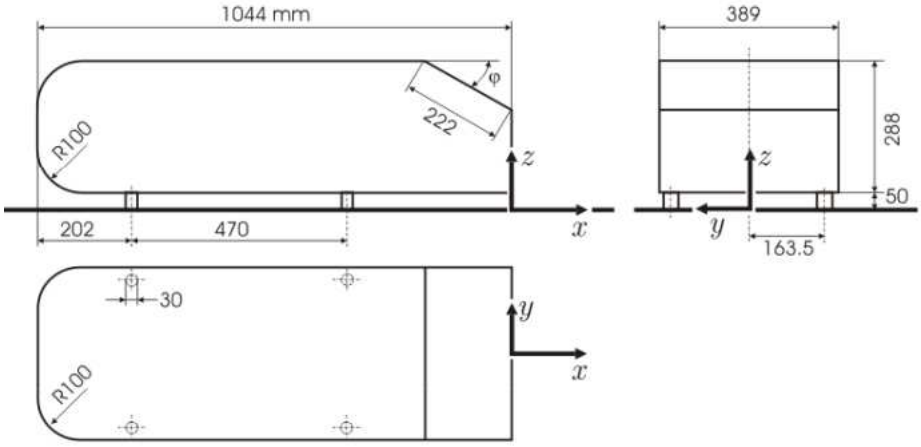



Figure 1.1: *Dessin du corps d'Ahmed.*

## 1.3 Conception avec Salome

*Salome* est un logiciel libre multiplateforme de conception assistée par ordinateur (CAO). Il est basé sur une architecture libre et flexible faite de composants réutilisables. Les fichiers exécutables tout comme les sources sont disponibles sur le site internet officiel du logiciel.

*Salome* est une solution multiplateforme et opensource (Licence GPL) permettant de réaliser des modèles de pièces en trois dimensions et des simulations (ou calculs) numériques. Capable de réaliser des extrusions, révolutions, opérations booléennes...etc, il permet de réaliser bon nombre de pièces 3D. *Salome* n'est pas un logiciel de CAO avec l'objectif d'avoir une représentation géométrique de bureau d'études (les plans avant fabrication). C'est en réalité un logiciel pour préparer des modèles géométriques pour la simulation numérique (il y en a de toutes sortes : calcul de la tenue mécanique, de la thermique, des écoulements fluides). D'ailleurs *Salome* ne fait que préparer le modèle discret (éléments finis). La simulation est à effectuer avec d'autres logiciels (Code_Aster, OpenFOAM, ... etc).

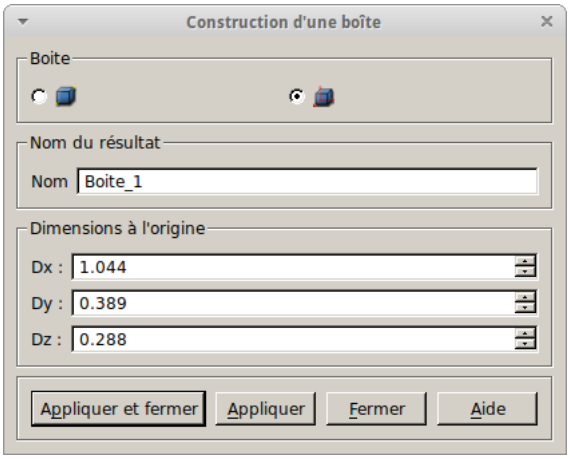
### 1.3.1 Démarche pas-à-pas

Lancer à partir du menu principal la suite *Salome,* changer le langage (Fichier $\Longrightarrow$ préférences) et suivre les étapes ci-dessous :

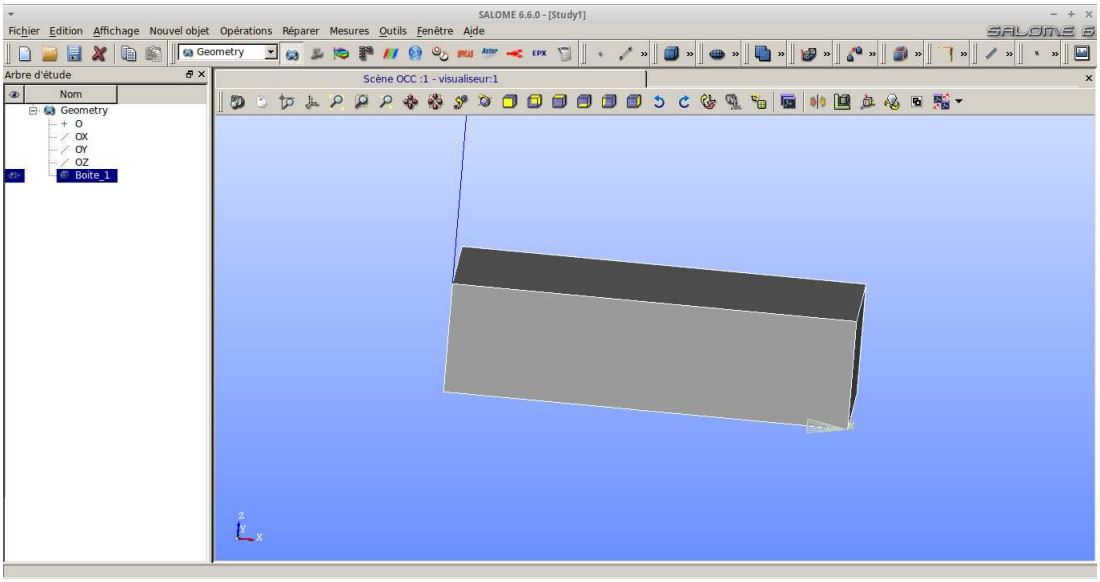
– Appuyez sur le bouton  pour lancer le module de géométrie et fixer l'unité

de mesure sur "*mètre*" dans (Fichier $\Longrightarrow$ Propriété...) ;

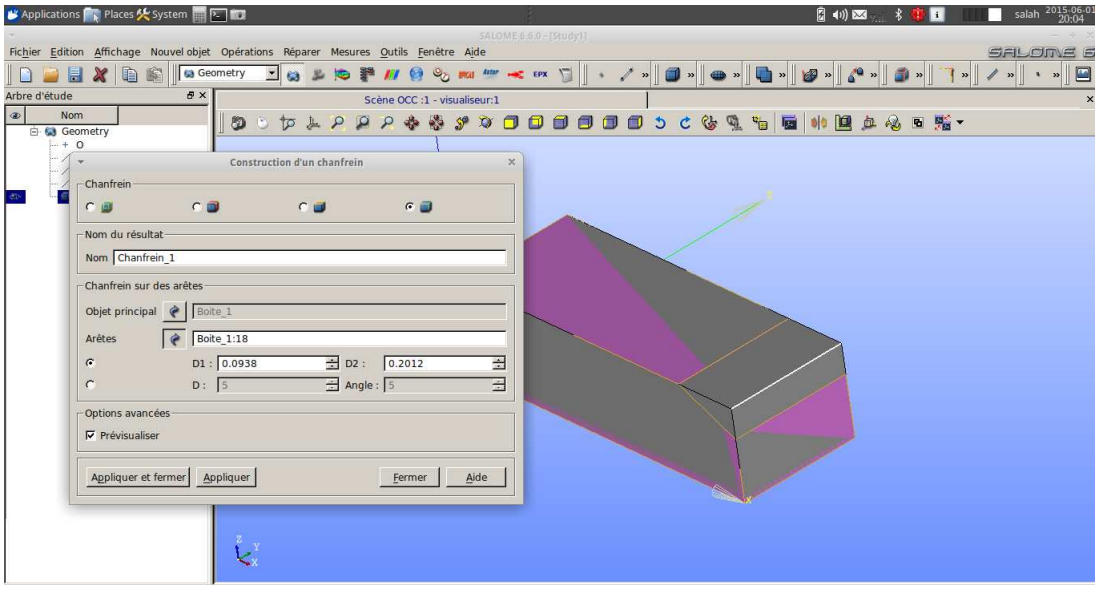
– Appuyez sur le bouton  ;



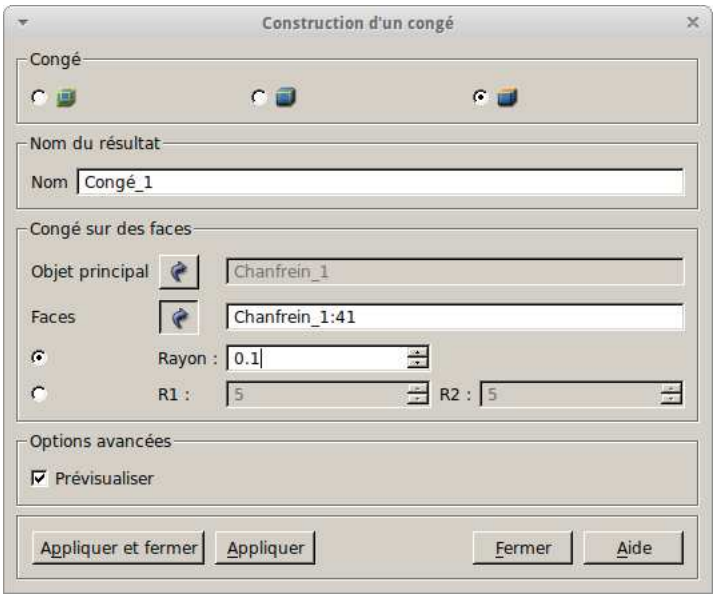

– Appuyez sur *Appliquer et fermer*, o aura ;



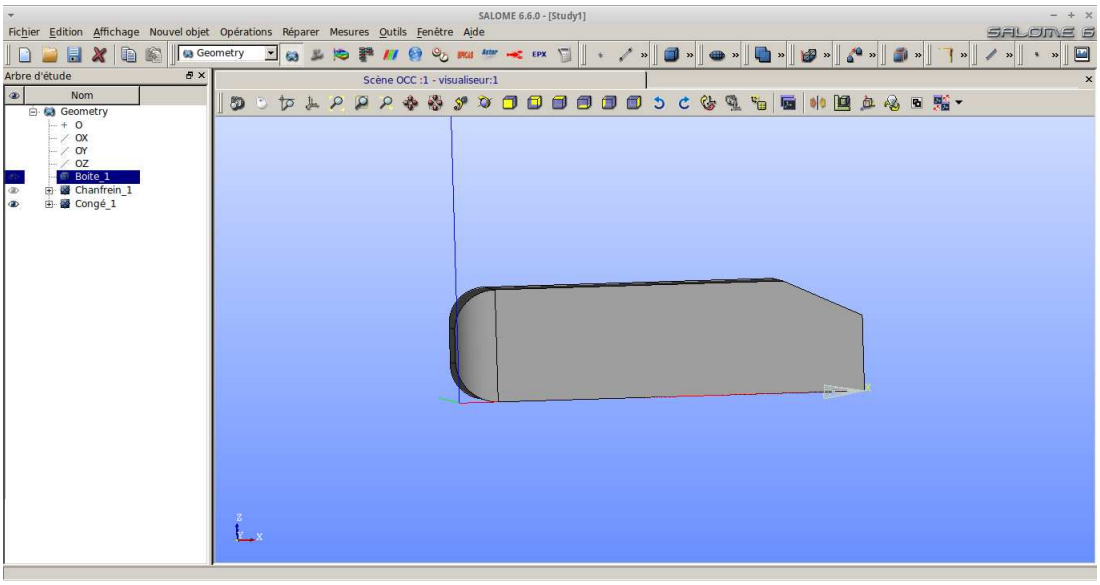

– Puis sur  pour créer le chanfrein ;



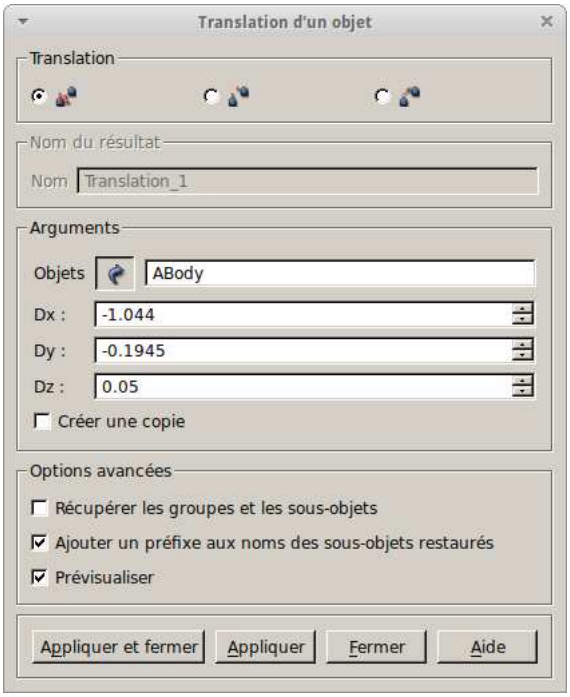

– Appuyez sur *Appliquer et fermer* ensuite sur  pour créer le congé 3D ;



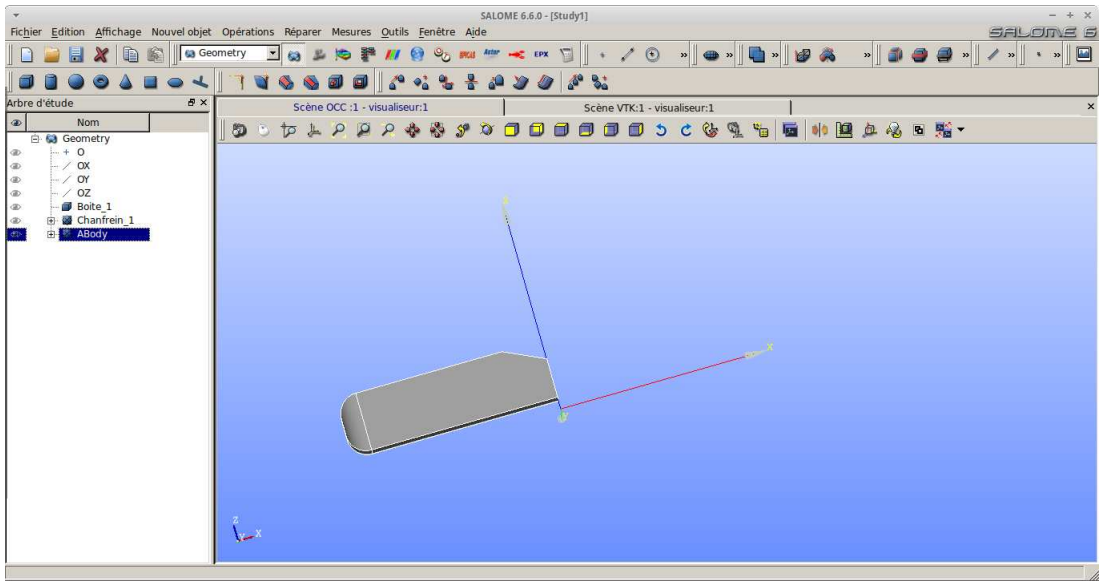

– Appuyez sur *Appliquer et fermer* ;



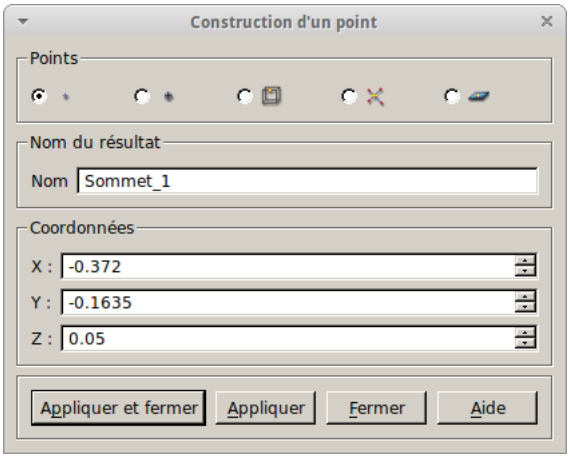

– Puis sur  pour translater l'objet ;



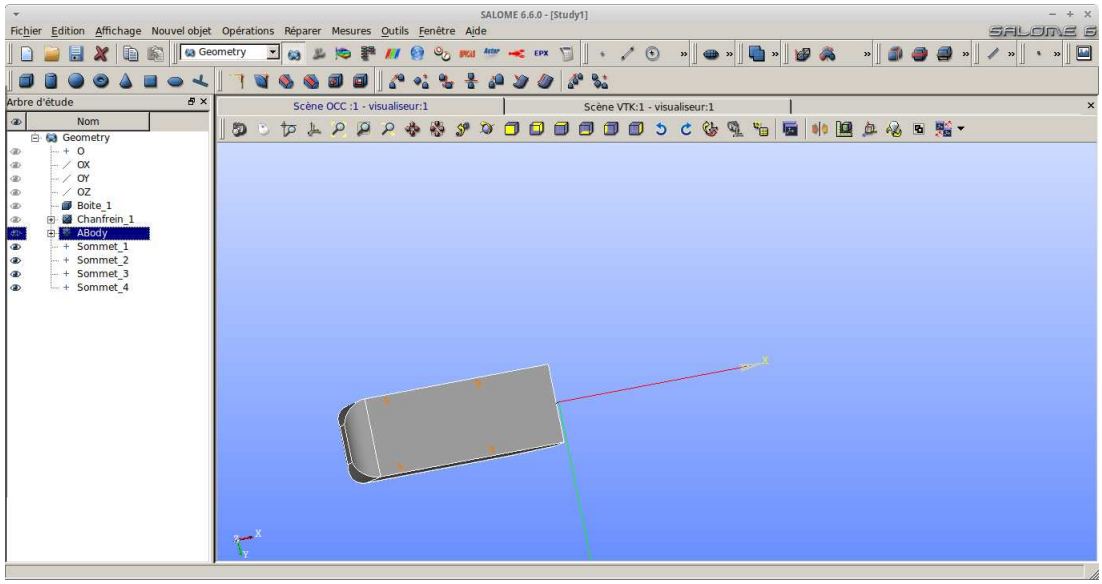

– Appuyez sur *Appliquer et fermer* ;



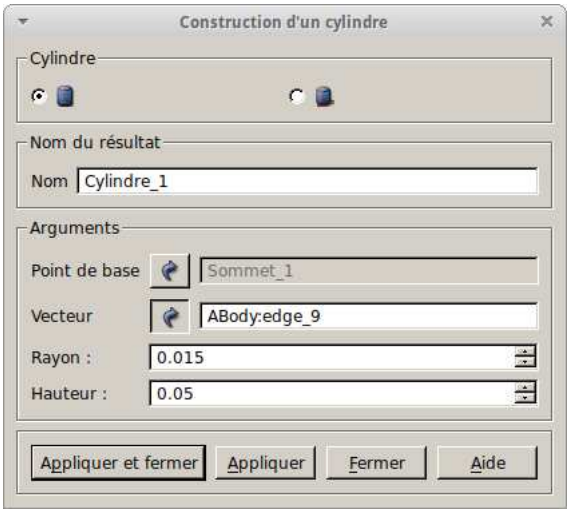

– Pour la construction des roues, appuyez sur  ;

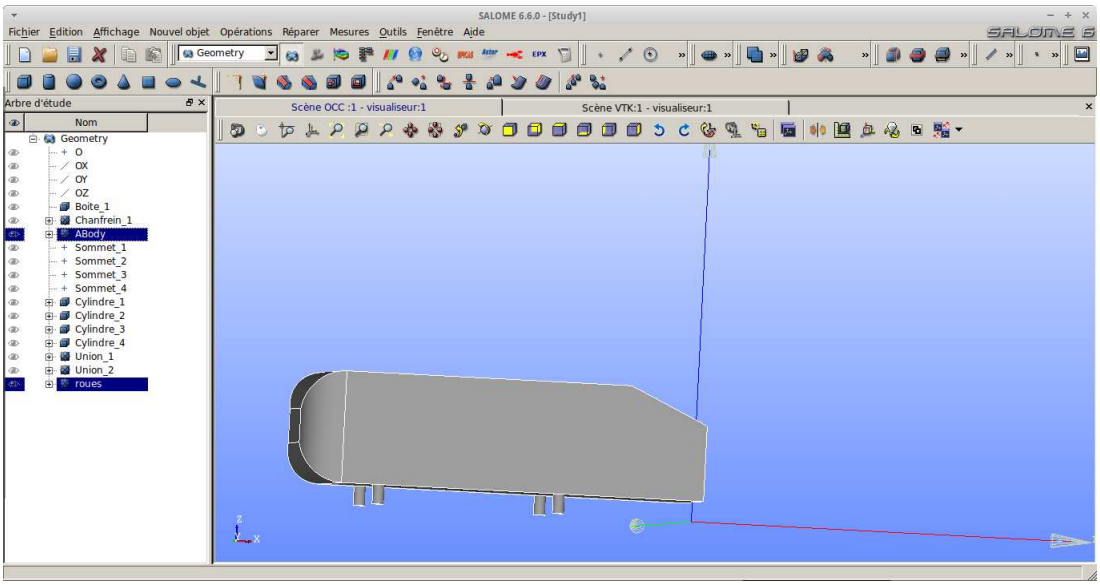



En translatant et copiant vers les 3 autres positions, onaura ;



– Appuyer sur  pour contruire les cylindres (translation + copies) ;




– A cette étape, on exporte les objets *ABody* et *Roues* au format STL dont nous aurons besoin par la suite.

**Remarque :**

Les fichiers STL sont des fichiers qui peuvent être sous format texte ou binaire qui caractérisent des modèles 3D découpés en surfaces triangulaires appelées «*facettes*».

## 1.4 Conclusion

Dans ce chapitre, nous avons essayé de maîtriser la suite *Salome* en tant que logiciel de CAO/DAO afin de dessiner le *corps Ahmed.* Nous remarquons bien sa puissance à dessiner cette géométrie 3d complexe en un minimum de temps. Nous allons voir, dans le prochain chapitre, l'utilisation de Salome en tant que mailleur.

# CHAPITRE 2

# MAILLAGE DU CORPS AHMED

## 2.1 Introduction

Nous allons détailler, dans ce chapitre, comment mailler le *corps Ahmed* soit en utilisant *Salome*, soit les outils spécifiques à *OpenFoam*.

## 2.2 Maillage avec Salome

Dans un premier temps, nous ne voulions pas sortir de la plateforme *Salome* pour effectuer toute notre étude. Nous avons investi beaucoup de temps à apprendre à mailler avec *Salome*. Tout marchait bien jusqu'au moment où nous avons utilisé des boîtes de raffinement autour du *corps Ahmed*. Ces dernières n'étant pas reconnus par le solveur en tant qu'entités virtuelles mais plutôt en tant que boîtes solides ! Ceci nous a poussé à utiliser les outils de maillage fournis avec *OpenFoam*. Afin de faire bénéficier le lecteur de l'utilisation de *Salome* comme mailleur, nous avons mis les détails de notre expérience en (Annexe A).

## 2.3 Maillage avec les outils d'OpenFoam

Il faut d'abord savoir qu'*OpenFOAM*, au contraire de Fluent, est à la fois capable de créer des maillages et de résoudre des équations par la méthode des volumes finis à partir de ses propres maillages. *OpenFOAM* permet de faire des maillages entièrement structurés, de types « *blockMesh* », ou de type « *snappyHexMesh* » qui est un outil permettant, à partir d'une simple géométrie 3D, de mailler le domaine qui l'entoure en quelques opérations seulement. La figure suivante montre un maillage généré avec *OpenFOAM* autour du corps Ahmed (Fig.2.1).

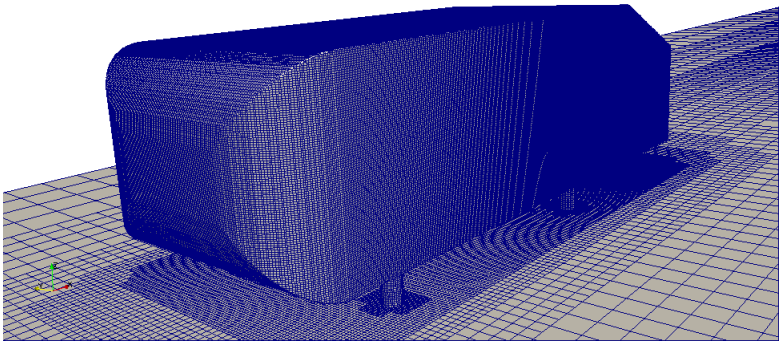
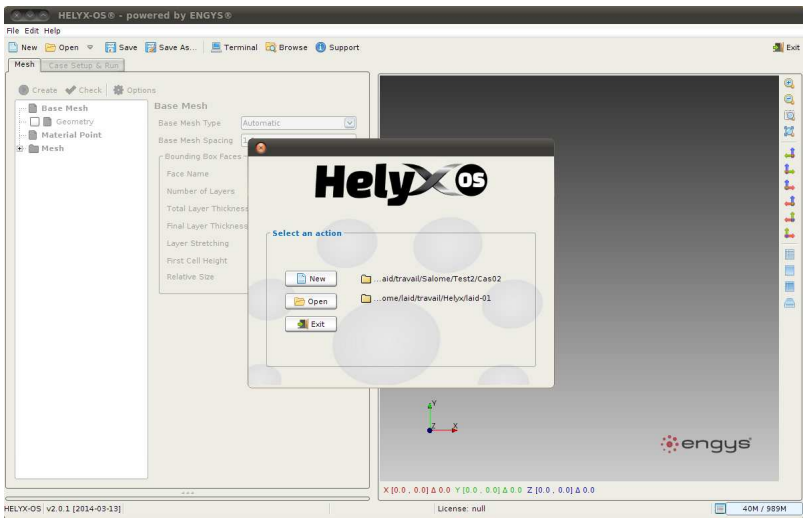
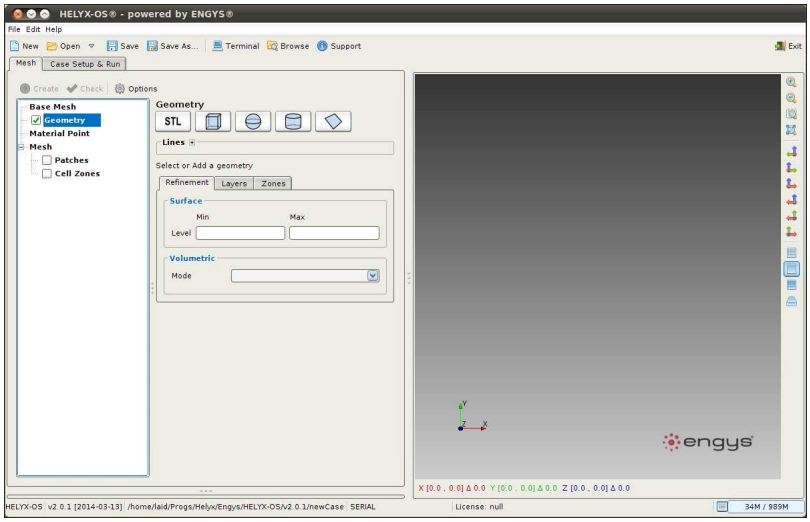


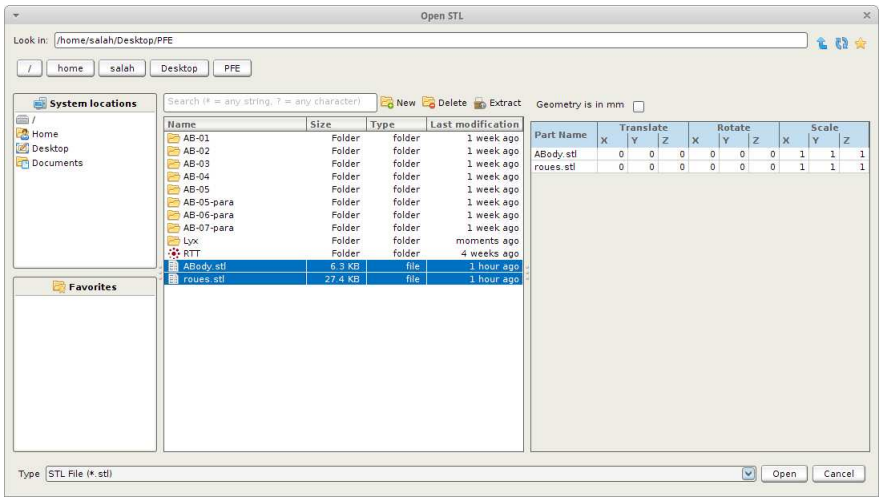
Figure 2.1: *Maillage généré avec OpenFOAM autour du corps Ahmed.*

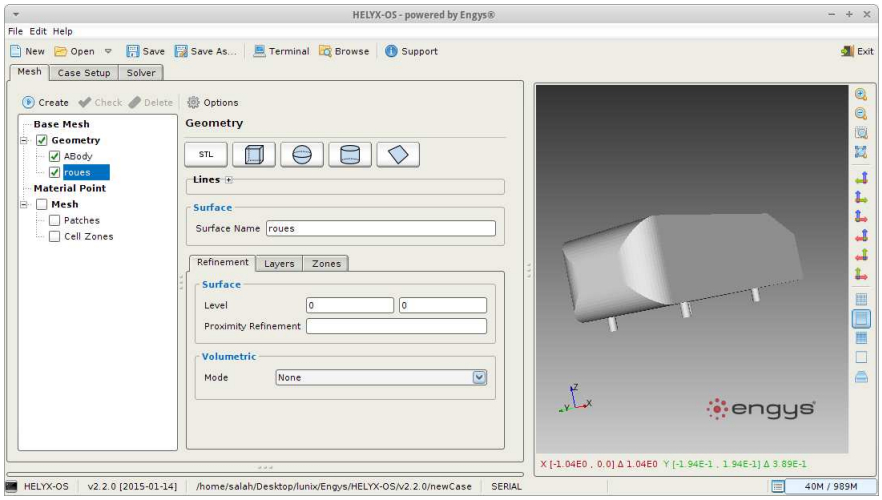
Avant tout, nous allons utiliser le logiciel *Helyx-OS* disponible aussi sous *CAELinux* afin de préparer le model avant le maillage :



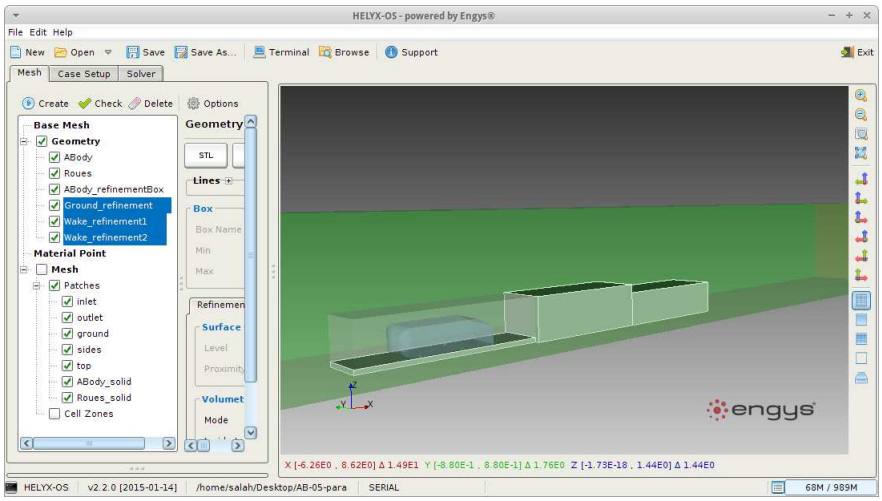




On importe les fichiers STL par *Geometry* $\rightarrow$*STL*



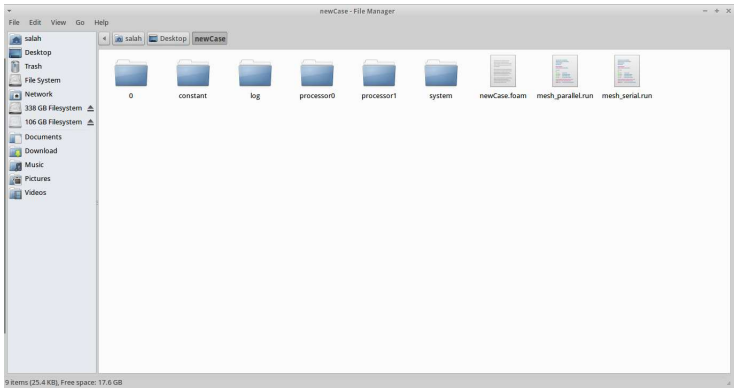

Appuyez sur  et créer les 4 blocs de raffinement,




On sauvegarde ce fichier afin de créer l'arborescence puis nous allons mailler avec d'autres outils car *Helyx-OS* est un GUI commercial pour *OpenFoam*. Nous pouvons aussi obtenir cette arborescence en utilisant *Discretizer* livré aussi avec *CAELinux*.

## 2.4 Création du maillage avec OpenFoam

La structure des répertoires sauvegardée précédement est :




Effacer tout, sauf les trois dossiers 


### 2.4.1 Utilisation de blockMesh

Pour des géométries simples, l'utilitaire de génération de maillage *blockMesh* (fourni avec *OpenFOAM*®), peut être utilisé pour créer des maillages structurés multiblocs. Le maillage est généré à partir d'un fichier de dictionnaire nommé *blockMeshDict* situé dans le répertoire *constante/polyMesh* [11].

#### Création du tunnel (Wind tunnel)

Nous avons besoin d'un tunnel représentant la veine d'essai et ainsi le domaine de simulation.

Dans le dossier 

on déplace le fichier 

dans */constant/polyMesh* 


Dans ce fichier se trouve :

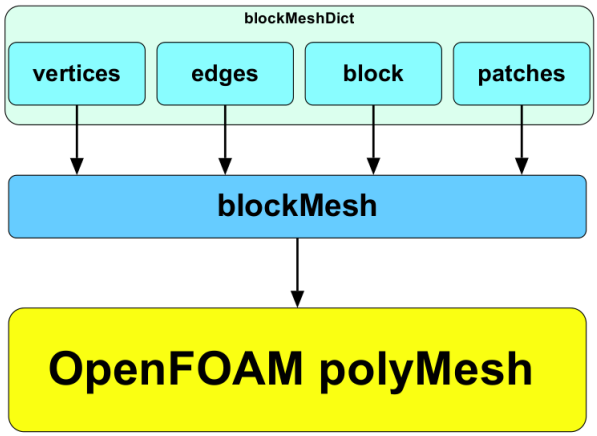



Figure 2.2: *Structure du fichier blockMeshDict.*

Dans notre cas, la taille du tunnel est : $x : [-6.63, 8.25]$, $y : [-0.88, 0.88]$, $z : [0, 1.44]$ qui est recommandée par ERCOFTAC. Le contenu du fichier *blockMeshDict* est :

```
    convertToMeters 1;
    vertices
     (
      ( -6.63 -0.88 0)
      ( 8.25 -0.88 0)
      (  8.25 0.88 0)
      ( -6.63 0.88 0)
      ( -6.63 -0.88 1.44)
      (  8.25 -0.88 1.44)
      (  8.25 0.88 1.44)
      ( -6.63 0.88 1.44)
    );
    blocks
    ( hex
      ( 0 1 2 3 4 5 6 7)
      ( 93 11 9) simpleGrading
      ( 1 1 1)
    );
    edges
    (
    );
   patches
(
   patch inlet  (  (0 4 7 3)  )
   patch outlet (  (1 2 6 5)  )
   wall ground  (  (0 3 2 1)  )
   symmetryPlane sides (  (3 7 6 2) (0 1 5 4) )
   symmetryPlane top   (  (4 5 6 7)  )
);
```

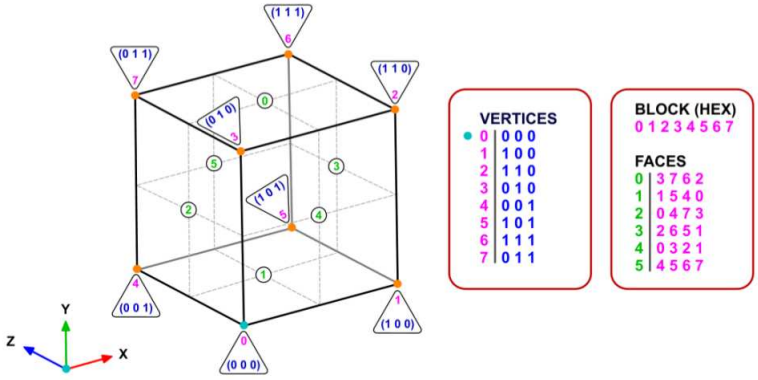



Figure 2.3: *Logique du fichier blockMeshDict.*

A partir de la ligne de commande et dans le répertoire de travail, on lance les commandes suivantes :

– ~/ABody$ **blockMesh**

qui permet de générer le maillage structuré défini dans le dictionnaire et

– ~/ABody$ **paraFoam**

qui permet de visualiser rapidement le résultat du maillage avec *ParaView*.

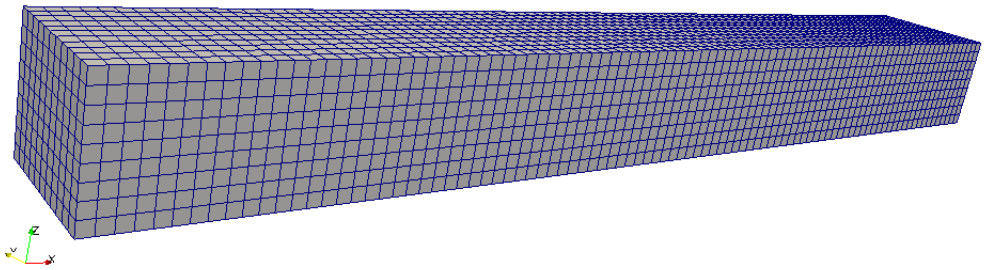


Figure 2.4: *Maillage du tunel par blockMesh.*

### 2.4.2 Utilisation de SnappyHexMesh

Pour des géométries complexes, l'utilitaire de génération de maillage *snappyHexMesh* (fourni avec *OpenFOAM*®), peut être utilisé. Il génère des mailles 3D contenant des hexaèdres à partir de la géométrie définie par le format stéréolithographie (STL). Le maillage est généré à partir d'un fichier de dictionnaire nommé *snappyHexMeshDict* situé dans le répertoire *système* et le fichier STL de la géométrie de surface triangulée situé dans le répertoire *constant/triSurface* [11].

La commande «*snappyHexMesh*» permet de modifier un maillage *OpenFOAM* existant (typiquement, un maillage de type «*blockMesh*») à l'aide d'une géométrie surfacique en trois dimensions sous format STL. Plus simplement, pour mailler un domaine autour d'une géométrie quelconque, par exemple le corps Ahmed il suffit d'avoir sa géométrie sous format STL, de créer un simple maillage de type blockMesh représentant le domaine autour d'Ahmed Body et de « demander » à *snappyHexMesh* de faire le reste de façon tout à fait automatique, comme illustré dans la figure suivante.

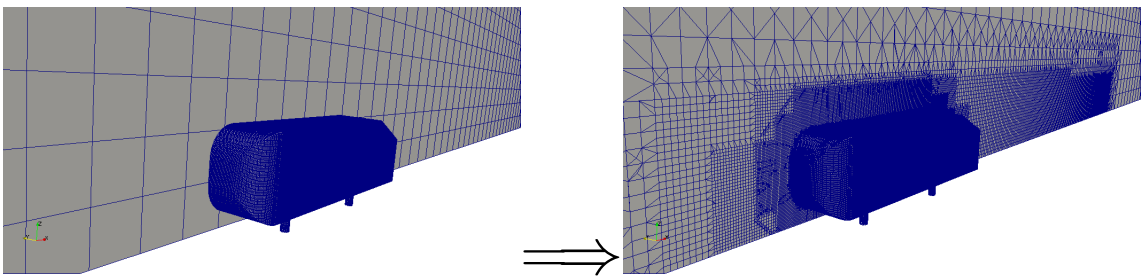


Figure 2.5: *Principe de snappyHexMesh.*

Ce processus de raffinement est basé sur un principe itératif simple : A chaque itération de raffinement, les cellules qui rencontrent les surfaces de la géométrie STL sont raffinées, et ainsi de suite. Cette méthode fait partie des principaux atouts d'*OpenFOAM* car elle permet en quelques minutes d'obtenir un maillage qui prendrait des heures entières (voire des jours) à obtenir avec des logiciels de maillage payants. De plus, le principal avantage de cet algorithme est qu'il ne nécessite pas une géométrie très propre (il peut y avoir des discontinuités, des petits trous, des intersections dans la géométrie sans perturber l'exécution de l'algorithme). En conséquence, tout ou presque est « maillable » sous *OpenFOAM* grâce à cet algorithme, du moment que l'on est capable d'avoir un fichier STL représentant la géométrie à étudier. En outre, *snappyHexMesh* permet de raffiner à l'intérieur de parallélépipèdes spécifiés par l'utilisateur sans l'usage de surface géométrique STL n'importe où dans le domaine et d'ajouter un raffinement de type « couche limite » autour des zones. L'autre atout incomparable de *snappyHexMesh* est que certains indices de qualité du maillage (aspect ratio, volume maximum, volume minimum, skewness) sont des paramètres d'entrée, si bien que l'algorithme de *snappyHexMesh* fait en sorte que ces indices soient respectés et fait automatiquement, au cours des itérations, les corrections nécessaires au maillage. Ainsi, la qualité du maillage n'est plus subie mais choisie dès le départ par l'utilisateur. L'algorithme agit en trois étapes :

1. Le raffinement du maillage autour de la géométrie et la suppression des mailles à l'intérieur (ou à l'extérieur) de celle-ci.

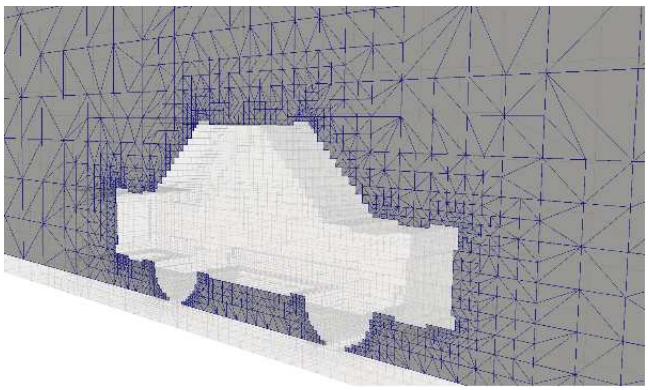


Figure 2.6: *Première étape de snappyHexMesh : raffinement.*

2. La modification de la forme des cellules aux extrémités du domaine afin d'épouser les contours de la géométrie surfacique.

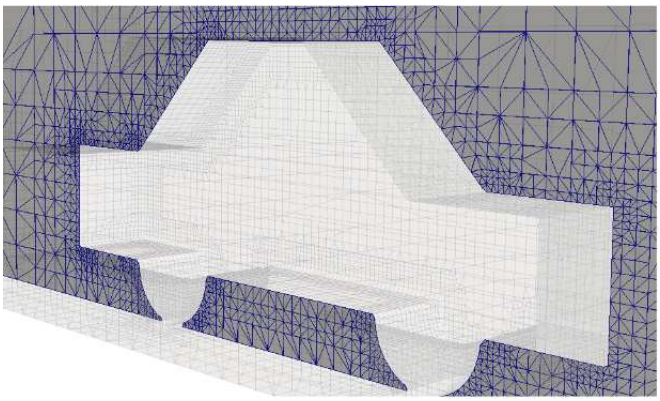


Figure 2.7: *Seconde étape de snappyHexMesh : découpage des cellules.*

3. L'ajout de couches limites proche des zones spécifiées (en décalant les mailles du maillage existant vers l'intérieur du domaine pour laisser la place aux mailles de type « couche limite »).

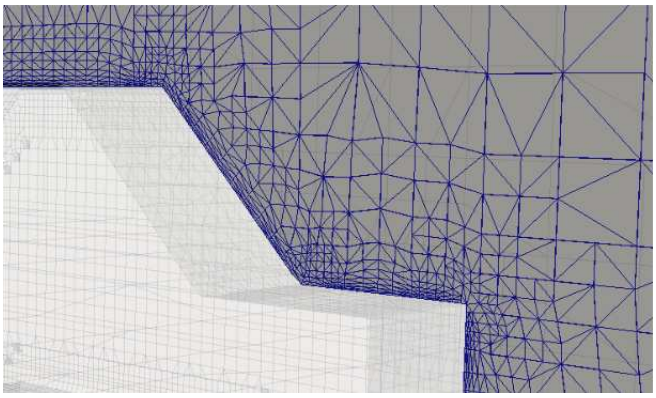


Figure 2.8: *Troisième étape de snappyHexMesh : ajout des couches limites.*

Lors de l'exécution de la commande *snappyHexMesh*, les trois étapes sont stockées dans trois dossiers temporels différents et ils peuvent être visualisés sous *ParaView* en faisant défiler les pas de temps.

**Note** : Avant de relancer *snappyHexMesh* depuis une console, il est nécessaire d'effacer tous les fichiers de maillage créés (sauf le fichier « *blockMeshDict* ») sinon quoi *OpenFOAM* occasionne une erreur à son exécution.

#### 2.4.2.1 Syntaxe du fichier « *snappyHexMeshDict* »

La syntaxe du fichier dictionnaire snappyHexMeshDict est la suivante :

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ACTIVATION_DES_OPTIONS

geometry
{
    LISTE_DES_GEOMETRIES_STL_ET_ZONES_DE_RAFFINEMENT
};

castellatedMeshControls
{
    PARAMETRES_DU_RAFFINEMENT
}
snapControls
{
    PARAMETRES_DE_LA_DECOUPE_DES_MAILLES_POUR_EPOUSER_LA_GEOMETRIE
}

addLayersControls
{
    PARAMETRES_DES_RAFFINEMENT_DE_TYPE_COUCHE_LIMITE
}

meshQualityControls
{
    PARAMETRES_PROPRES_AU_MAILLAGE_DANS_SON_ENSEMBLE
}

debug 0; ← Cette ligne n'intéresse pas l'utilisateur

mergeTolerance 1E-6; ← Valeur par défaut donnant de bons résultats :
                                aucune modification n'est nécessaire
```

Figure 2.9: *Syntaxe et du fichier dictionnaire snappyHexMeshDict.*

#### 2.4.2.2 Activation des options

L'outil *snappyHexMesh* peut effectuer les trois étapes présentées plus haut (raffinement, découpe des cellules et couches limites), une seule d'entre elles ou deux d'entre elles selon le choix de l'utilisateur. Il est possible, par exemple, de ne faire que la première étape, puis, une fois satisfait par le raffinement, passer à la seconde et ainsi de suite. S'il existe des dossier temporels contenant des fichiers de maillage dans le dossier de travail, l'exécution du *snappyHexMesh* se base sur le dernier d'entre eux. Dans le fichier « *snappyHexMeshDict* », ACTIVATION_DES_OPTIONS doit être remplacé par :

- castellatedMesh **VALEUR** ;
- snap **VALEUR** ;
- addLayers **VALEUR** ;

avec **VALEUR** valant « **false** » ou « **true** » et le mot clef « **castellatedMesh** » désignant l'étape de raffinement, « **snap** » celle de découpage et « **addLayers** » l'ajout de couches limites. Par exemple, pour ne faire que la première étape, la bonne syntaxe est la suivante :

- castellatedMesh **true** ;
- snap **false** ;
- addLayers **false** ;

#### 2.4.2.3 Raffinement

La troisième partie du fichier «*snappyHexMeshDict*» est consacrée au paramétrage de la première étape de raffinement. Dans cette partie, PARAMETRES_DU_RAFFINEMENT doit être remplacé par le code suivant :

```
maxLocalCells VALEUR;

maxGlobalCells VALEUR;

minRefinementCells VALEUR;

nCellsBetweenLevels VALEUR;

features
(
        ← Dans le cadre de ce document, nous ne nous intéressons pas à cette partie qui peut rester vide.
 );

refinementSurfaces
{
        NOM_DE_LA_GEOMETRIE_STL
        {
            level (VALEUR_MIN  VALEUR_MAX);
        }
}

 resolveFeatureAngle VALEUR;

 refinementRegions
```

Autant de fois qu'il y a de géométries STL

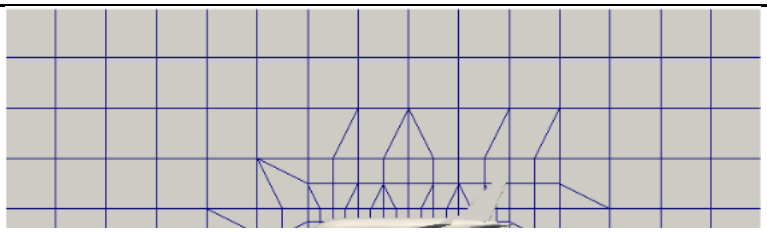
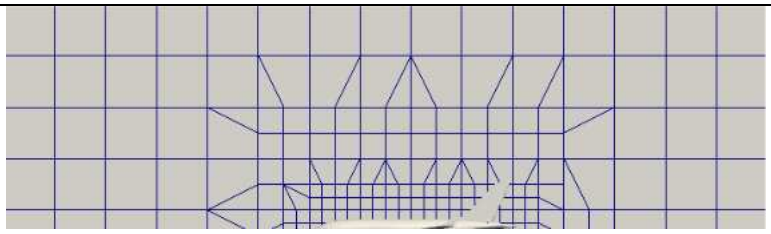
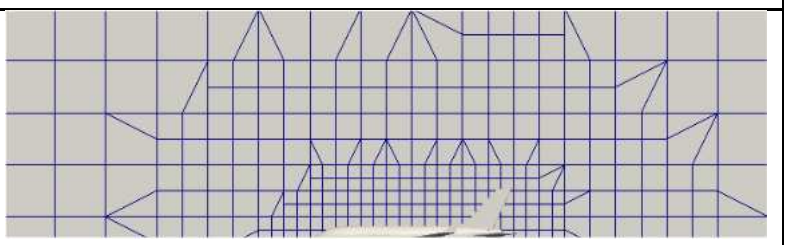
```
 {
        NOM_DE_LA_ZONE_DE_RAFFINEMENT
        {
            mode inside;
            levels ((1e15 VALEUR));
        }
 }

 locationInMesh COORDONNEES_D'UN_POINT_DANS_LE_DOMAINE;
```

Autant de fois qu'il y a de zones à raffiner

«maxGlobalCells» est le nombre de cellules maximum que l'on veut donner au maillage. A chaque étape de raffinement, le nombre total de cellules est calculé et s'il est supérieur à la VALEUR indiquée (exp. «1000000»), le raffinement s'arrête. Dans le cas d'un calcul en parallèle (c'est-à-dire multiprocesseur), «maxLocalCells» est le nombre maximum de cellules par processeur. Dans le cas d'un calcul mono-processeur, la VALEUR attribuée à l'un et l'autre de ces paramètres peut être identique. Aussi, «minRefinementCells» correspond au nombre de cellules en dessous duquel le processus

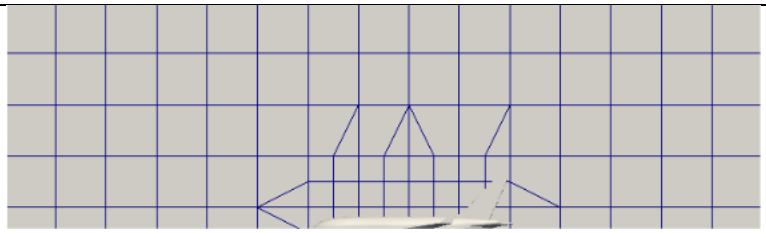
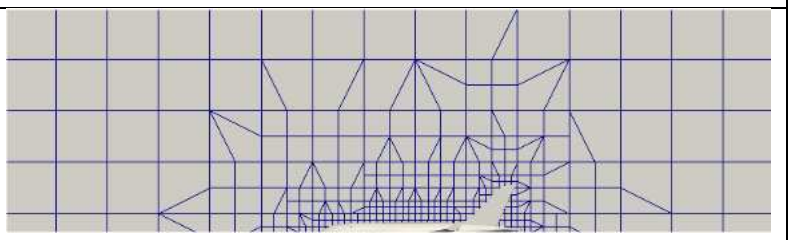
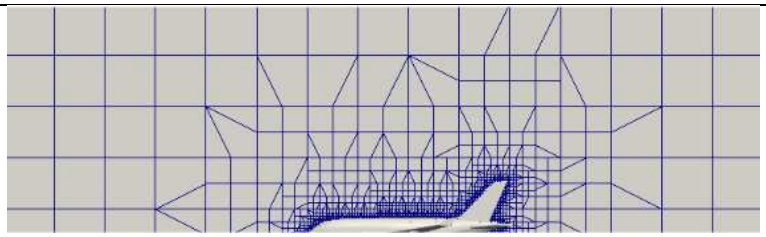
de raffinement s'arrête. En effet, le nombre de cellules raffinées augmente fortement dans les premières itérations puis diminue de façon presque asymptotique vers zéro. «minRefinementCells» permet de préciser à partir de combien de cellules le processus s'arrête .

D'autre part, pendant le processus de raffinement, chaque cellule raffinée peut conduire au raffinement de ses cellules voisines bien que celles-ci ne soient pas en contact avec la géométrie STL. «nCellsBetweenLevels» correspond au nombre de cellules raffinées par ce biais. Le tableau suivant donne des exemples de raffinement d'un maillage autour d'une géométrie d'avion (A380) qui comprend deux itérations de raffinement avec plusieurs valeurs de « nCellsBetweenLevels ».

| nCellsBetweenLevels | résultat |
|---|---|
| «1» |  |
| «2» |  |
| «3» |  |

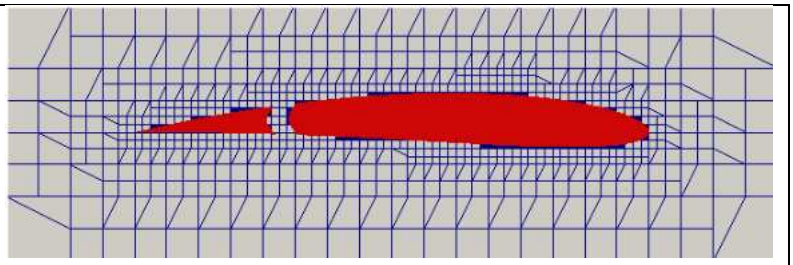
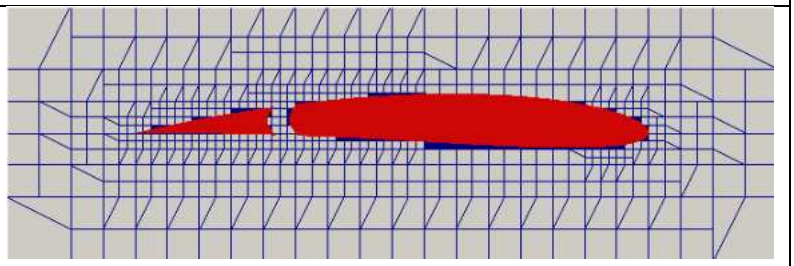
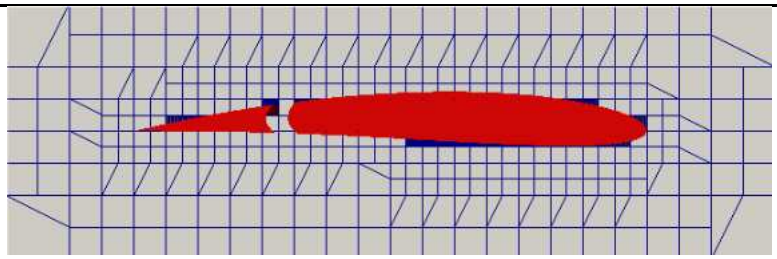
Tabeau 2.1: *Influence de « nCellsBetweenLevels » sur le maillage.*

Vient ensuite le choix du nombre d'itération de raffinement pour chaque géométrie STL. Dans la syntaxe présentée plus haut, NOM_DE_LA_GEOMETRIE_STL est le nom donné à la géométrie STL dans la seconde partie du ficher «*snappyHexMesh*» et non pas le nom du fichier STL se trouvant dans le dossier «*constant/triSurface*» du dossier de travail. VALEUR_MIN et VALEUR_MAX de « level » correspondent au nombre d'itérations de raffinement autour de la géométrie. Habituellement, VALEUR_MIN est égale à VALEUR_MAX moins 1. Les deux valeurs peuvent être égales. Le tableau suivant représente un maillage modifié autour de la même géométrie que précédemment avec différentes VALEUR_MIN et VALEUR_MAX de « level ».

| level | résultat |
|---|---|
| «1» |  |
| «3» |  |
| «5» |  |

Tabeau 2.2: *Influence de « level » sur le maillage.*

**Note** : le nombre de niveaux est fortement influant sur le temps de modification du maillage. De plus, *snappyHexMesh* permet de réserver la dernière itération aux parties de la géométrie STL les plus accidentées. D'après la documentation officielle, le «resolveFeatureAngle» est un angle vu par la cellule au dessous duquel celle-ci n'est pas raffinée lors de la dernière itération de raffinement. Cette définition étant peu intuitive, on peut tout simplement retenir que plus cet angle est grand et plus le raffinement de la dernière itération se concentrera sur les parties accidentées de la géométrie STL. Le tableau suivant montre un maillage modifié avec *snappyHexMesh* autour d'un profil d'aile avec différentes VALEUR du «resolveFeatureAngle».

| resolveFeatureAngle | résultat |
|---|---|
| « 0 » |  |
| « 120 » |  |
| « 180 » |  |

Tabeau 2.3: *Influence de «resolveFeatureAngle» sur le maillage.*

Enfin, dans «refinementBox» sont spécifiés les options de raffinement de la ou des zone(s) parallélépipédique(s) facultative(s) définie(s) dans la seconde partie du «*snappyHexMeshDict*». Il existe trois modes de raffinement : «inside» (à l'intérieur de la zone de raffinement), «outside» (à l'extérieur de celle-ci) et « distance » (jusqu'à une certaine distance de celle-ci). Seule le permier mode est traité ici. Pour les deux autres modes, l'utilisateur se référera à la documentation officielle d'*OpenFOAM* [11]. Concernant le premier mode, la syntaxe est donnée ici :

```
NOM_DE_LA_ZONE_DE_RAFFINEMENT
        {
            mode inside;
            levels ((1e15 VALEUR));
        }
```

Il suffit ici de remplacer VALEUR par le nombre d'itérations souhaité de raffinements dans la zone. Le nombre « 1e15 » ne se rapporte à rien et est ignoré pendant l'exécution de *snappyHexMesh*. Dernièrement, COORDONNEES_D'UN_POINT_DANS_LE_DOMAINE sont tout simplement les coordonnées entre parenthèses (exp. (1 1 1)) d'un point quelconque du domaine. Cette donnée est indispensable à *snappyHexMesh* pour savoir s'il conserve le maillage qui est à l'intérieur ou à l'extérieur de la géométrie STL.

**- Découpe des cellules :**

Dans la syntaxe du fichier «*snappyHexMeshDict*» présenté en début de chapitre, PARAMETRES_DE_LA_DECOUPE_DES_MAILLES_POUR_EPOUSER_LA_GEOMETRIE peut être remplacé par les lignes suivantes :

```
nSmoothPatch 3;
tolerance 4;
nSolveIter 30;
nRelaxIter 5;
```

En effet, les valeurs données dans le cas « **motorBike** » du dossier « **tutorials** » d'*OpenFOAM* donnant de bons résultats et la modification de ces paramètres n'ayant à priori pas d'influence flagrante sur le maillage, il n'est pas nécessaire de les changer en première approche.

**- Couches limites :**

La syntaxe de PARAMETRES_DES_RAFFINEMENT_DE_TYPE_COUCHE_LIMITE est la suivante :

```
relativeSizes VALEUR;

    layers
    {
        NOM_DE_LA_ZONE_SUR_LAQUELLE_METTRE_DES_COUCHES_LIMITES
        {
            nSurfaceLayers NOMBRE_DE_COUCHES;
        }
    }

    expansionRatio VALEUR;

    finalLayerThickness VALEUR;

    minThickness VALEUR;

    nGrow 1;
    featureAngle 30;
    nRelaxIter 3;
    nSmoothSurfaceNormals 1;
    nSmoothNormals 3;
    nSmoothThickness 10;
    maxFaceThicknessRatio 0.5;
    maxThicknessToMedialRatio 0.3;
    minMedianAxisAngle 130;
    nBufferCellsNoExtrude 0;
    nLayerIter 50;
```

Autant de fois qu'il y a de zones sur lesquelles mettre des couches limite

La VALEUR de « relativeSize » doit être soit « true » soit « false » selon que l'utilisateur souhaite respectivement travailler en relatif (la taille des couches est rapportée

à la taille de la maille la plus proche de la zone) ou en absolu (la taille des couches est exprimée en mètres).

Aussi, il est bien important de savoir et de comprendre que les couches limites sont, avec *snappyHexMesh*, associées à des zones telles que décrites dans le chapitre sur le maillage de type «*blockMesh*» (exp. «bigInlet», «upperWall», ...etc) et non pas aux géométries STL définies dans la seconde partie du fichier «*snappyHexMeshDict*». Ainsi, il est possible d'ajouter des couches limites même sur les extrémités du domaine qui ne sont pas en contact avec la géométrie STL. D'autre part, alors que les options de raffinement sont communes à toute la géométrie d'un même fichier STL, le nombre de couches peut être différent sur les différentes parties d'une même géométrie STL si celle-ci contient plusieurs zones (les autres options relatives au raffinement de type «couche limite» sont, par contre, communes à toutes les zones du maillage). Dans la syntaxe ci-dessus, NOM_DE_LA_ZONE_SUR_ LAQUELLE_METTRE_DES_COUCHES_LIMITES doit donc être remplacé par le nom de la zone sur laquelle l'utilisateur souhaite ajouter un raffinement de type «couche limite» et non pas par le nom qu'il a donné à la géométrie STL dans la seconde partie du dictionnaire «*snappyHexMeshDict*».

**Note** : Il est possible de connaître le nom des zones d'une géométrie STL en utilisant *ParaView* (commande «*paraFoam*» dans une console à la racine du dossier de travail) après avoir fait au moins la première étape de *snappyHexMesh* (le raffinement). Les noms des zones apparaissent sous les zones du domaine définies dans le *blockMeshDict* (exp. «stlSurface_patch0») à gauche. Une autre option est d'ouvrir le fichier «boundary» se trouvant dans le dossier *constant/polyMesh* du dossier temporel créé par *snappyHexMesh* pour avoir la liste de toutes les zones du maillage.

Pour chaque zone concernée, NOMBRE_DE_COUCHES correspond au nombre de couches souhaitées sur cette zone.

La VALEUR de «expansionRatio» correspond au rapport de l'épaisseur de la couche n+1 sur celle de la couche n en partant de la zone vers l'intérieur du domaine.

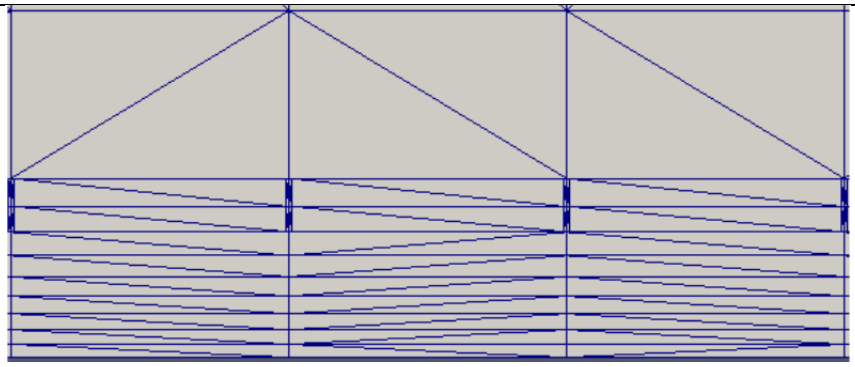
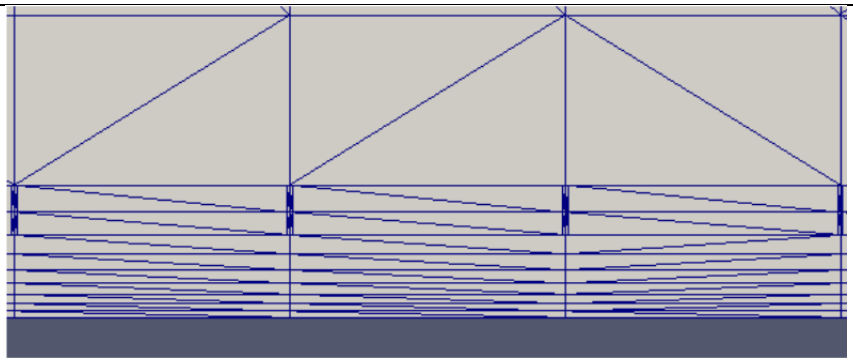
| expansionRatio | résultat |
|---|---|
| « 1.1 » |  |
| « 1.2 » |  |

Figure 2.10: *Influence de «expansionRatio» sur la taille des couches limites.*

La VALEUR de «finalLayerThickness» correspond à l'épaisseur de la couche la plus éloignée de la zone en question. Les autres paramètres par défaut offrent de bon résultats et leur modification n'a pas d'effet sensible sur le maillage final, c'est pourquoi il n'est pas nécessaire de les modifier en première approche.

**Note** : Les couches limites ne se voient pas immédiatement sous *ParaView* car elles sont « à l'intérieur » du maillage et n'apparaissent qu'à l'aide d'un plan de coupe ou d'un «clip».

**- Qualité du maillage :**

Dans la syntaxe présenté en début de chapitre, PARAMETRES_PROPRES_AU_MAILLAGE_DANS_SON_ENSEMBLE peut être remplacé par les lignes suivantes :

```
    maxNonOrtho 65;
    maxBoundarySkewness 20;
    maxInternalSkewness 4;
    maxConcave 80;
    minFlatness 0.5;
    minVol 1e-13;
    minArea -1;
    minTwist 0.02;
    minDeterminant 0.001;
    minFaceWeight 0.02;
    minVolRatio 0.01;
    minTriangleTwist -1;
    nSmoothScale 4;
    errorReduction 0.75;
```

Les valeurs par défaut donnant de bon résultats, il n'est pas nécessaire de les modifier. Cependant, l'utilisateur peut consulter la documentation officielle pour plus d'information [11].

**- Maillage du corps Ahmed :**

Nous avons passé énormément de temps à comprendre tous ces paramètres influençant le maillage et nous avons finis par avoir un maillage très acceptable du *corps Ahmed* et pour lequel nous avons eu de bons résultats. Le contenu de notre fichier dictionnaire *snappyHexMeshDict* de maillage est présentée en annexe *B* et la figure (2.11) montre les boîtes de raffinement utilisées autour du *corps Ahmed*.

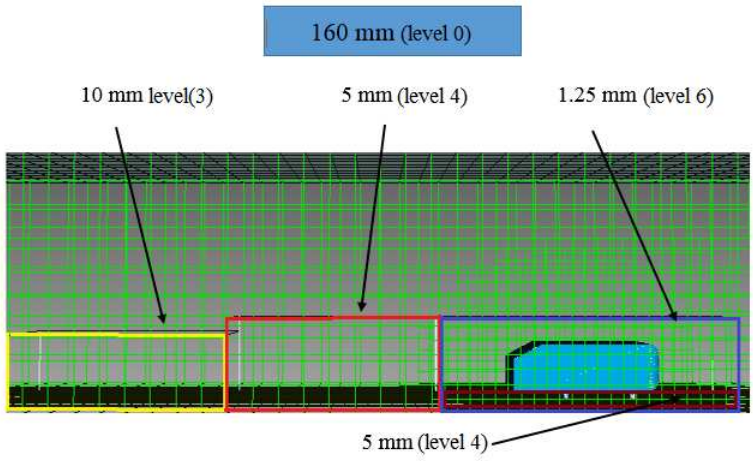



Figure 2.11: *Boîtes de raffinement utilisées autour du corps Ahmed.*

En exécutant la cammande de génération de maillage ci-dessous, le mailleur prendra en compte tous les paramètres contenus dans le dictionnaire et générera le maillage qui sera visible dans *Paraview* en exécutant la seconde commande.

- ~/ABody$ **snappyHexMesh -overwrite**
- ~/ABody$ **paraFoam**

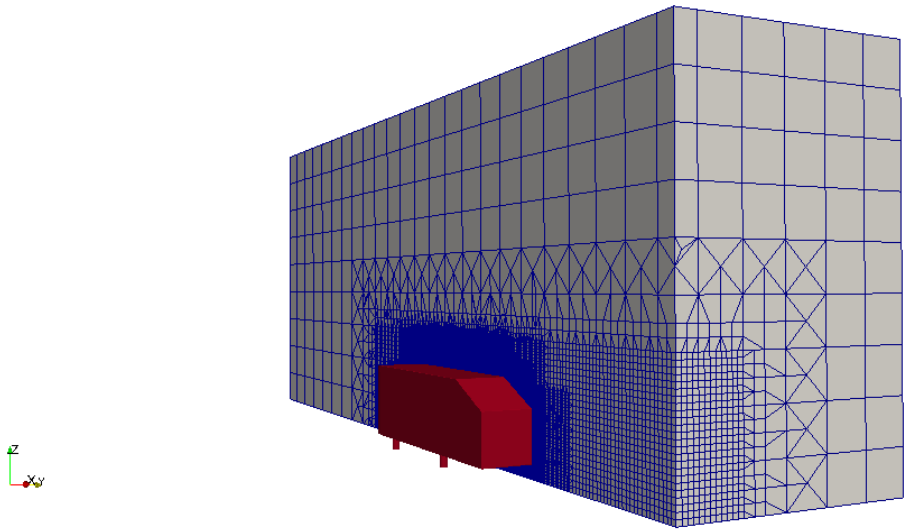


Figure 2.12: *Maillage du corps Ahmed avec snappyHexMesh.*

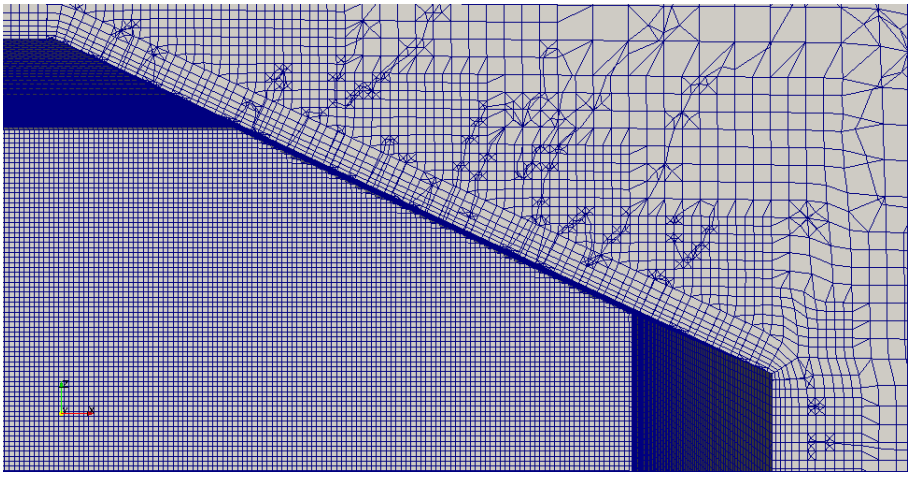


Figure 2.13: *Maillage de type (C.L) proche des parois de la géométrie STL (1.5 Milllion de noeuds).*

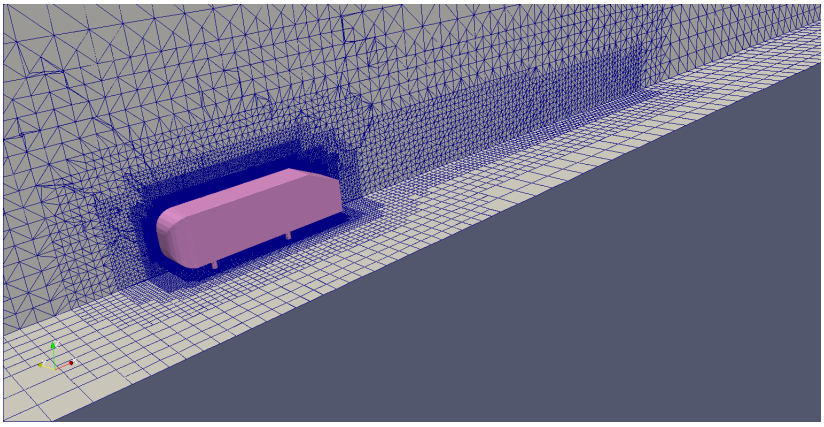


Figure 2.14: *Maillage du corps Ahmed avec blocs de rafinnement.*

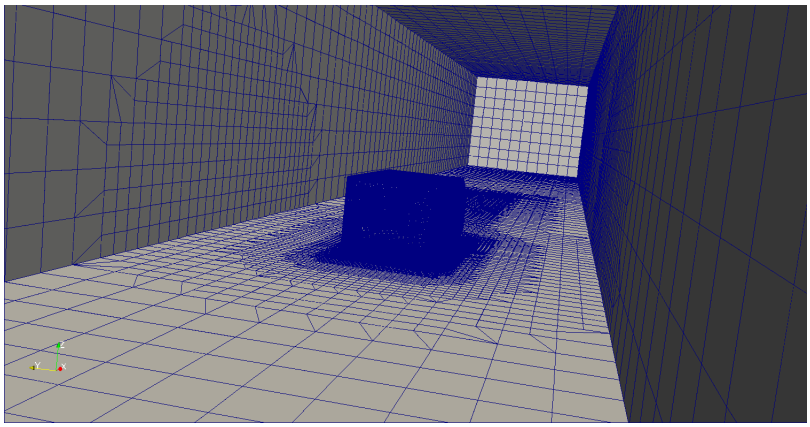


Figure 2.15: *Maillage du corps Ahmed dans le tunel (6.4 Millions de noeuds).*

## 2.5 Conclusion

Dans cette partie, nous avons passé en revue tous les outils nécessaires pour faire un maillage de qualité afin de l'utiliser par la suite dans les simulations. En réalité, c'est la partie la plus délicate durant le processus de simulation. En effet, il ne suffit pas de lancer la commande et de récupérer le maillage généré mais il faut vérifier minutieusement ce maillage car il y a toujours des endroits où il n'est pas conforme. Si c'est le cas, il faut changer certains paramètres dans le dictionnaire et puis relancer la génération. Ce processus peut se répéter des dizaines de fois jusqu'à l'élimination du problème.

Nous allons utiliser le maillage qui nous a donné satisfation pour les simulations numériques qui seront abordées au prochain chapitre.

# CHAPITRE 3

# SIMULATIONS

## 3.1 Introduction

## 3.2 Présentation d'OpenFOAM

*OpenFOAM* (Open Field Operation and Manipulation) est une boite à outils de simulation multi-physiques principalement axé sur la résolution des équations de la mécanique des fluides. Il est distribué depuis 2004 sous licence open source GNU/GPL par la société britannique *OpenCFD Ltd.* Son développement, en C++, a été amorcé par l'Imperial College London qui souhaitait un code de calcul basé sur la méthode des volumes finis et bénéficiant des dernières innovations en termes de langage informatique. Il est principalement constitué d'une bibliothèque logicielle en langage C++ libre3, et de différents outils, sous forme de bibliothèques et applications [12]. Il est livré avec de nombreux solveurs couvrant une large gamme de domaines tels que la combustion, les écoulements compressibles, incompressibles, multiphasiques, avec réactions chimiques, les transferts thermiques... Différents modèles de turbulence (RANS, LES, ...) sont également présents. *OpenFOAM* est distribué avec *ParaView*, un logiciel de post-traitement open source. Pour les utilisateurs préférant utiliser leur outil de visualisation, il existe des modules d'export pour *Fluent*, *EnSight* et *FieldView*. Le code *OpenFOAM* vu comme une bibliothèque C++ prend tout son intérêt lorsqu'il s'agit d'utiliser de nouveaux modèles. En effet, contrairement à la majorité des codes scientifiques écrits de façon séquentielle (souvent en Fortran), *OpenFOAM* profite de la puissance des langages orientés objet (POO). Cette structure sous forme de classes permet de se rapprocher de l'écriture mathématique en termes d'opérateur divergence, rotationnel, gradient, laplacien, dérivée temporelle...etc. Aucune connaissance approfondie du C++ n'est nécessaire pour écrire son modèle dans *OpenFOAM*.

## 3.3 Logique d'OpenFOAM

Maintenant qu'une brève présentation d'*OpenFOAM* a été faite et que l'intérêt d'un tel outil a été présenté, voici les connaissances de bases qu'un utilisateur potentiel doit avoir. En effet, *OpenFOAM* n'étant pas un outil comme les autres, il est nécessaire de bien appréhender sa logique pour pouvoir l'étudier et le comprendre.

### 3.3.1 Les solveurs

Par exemple, la notion de solveur est fondamentale en ce qui concerne *OpenFOAM*. En effet, la première chose à se demander, lors de l'étude d'un cas est celle du solveur à utiliser. Contrairement à *Fluent*, *OpenFOAM* a, du point de vue de l'utilisateur, presque autant de solveurs différents que de type de cas d'étude possibles. Par exemple, il a un solveur nommé «*icoFoam*» dédié uniquement aux écoulements incompressibles laminaires instationnaires. Un autre solveur nommé «*simpeFoam*» est dédié aux écoulements incompressibles turbulents stationnaires. Ainsi, il n'y a pas besoin de choisir les équations à résoudre, comme pour *Fluent*, mais uniquement celui de choisir le bon solveur. Le (tableau.3.1) présente les solveurs qui peuvent être utiles dans le cadre de la CFD.

### 3.3.2 Dossier de travail

Avec *OpenFOAM*, un dossier de travail (maillage et un calcul CFD) est celui sur lequel l'utilisateur est en train de travailler. Sa structure est donnée sous forme d'arbre dans la figure suivante.


Figure 3.1: *Structure du dossier de travail.*

Cette structure n'est pas figée et est susceptible d'être étoffée pendant l'utilisation. Néanmoins, la base est toujours la même.

#### 3.3.2.1 Dossiers temporels

Dans le dossier de travail, les dossiers temporels sont ceux qui contiennent les valeurs des différentes grandeurs calculées au cours du temps (pression, vitesse, intensité turbulente, ... etc) réunies dans des fichiers dont le nom correspond à ces grandeurs (exp. «p»

| Nom du solveur | | Type d'écoulement lié |
|---|---|---|
| Basique | *laplacianFoam* | Diffusion thermique dans un solide |
| | *potentialFoam* | Initialiser un écoulement potientiel simple avant la résolution des équations de N-S |
| Incompressible | *boundaryFoam* | Écoulement permanent 1D avec turbulences |
| | *channelFoam* | Pour un écoulement dans un tube avec de grandes turbulences |
| | *icoFoam* | Transitoire, Laminaire et isotherme pour fluide Newtonien |
| | *nonNewtonianIcoFoam* | Transitoire, laminaire, isotherme, non-Newtonien |
| | *pimpleDyMFoam* | Transitoire, fluide Newtonien sur parois en translation |
| | *pimpleFoam* | Transitoire, grand pas de temps |
| | *pisoFoam* | Transitoire |
| | *simpleFoam* | Permanent, turbulent |
| Compressible | *rhoCentralFoam* | Solveurs basé sur l'algorithme de Kurganov&Tadmor |
| | *rhoPisoFoam* | Non permanent, laminaire ou turbulent |
| | *rhoPorousMRFPimpleFoam* | Géométrie poreuse et MRF, chauffage, climatisation,ventilation, laminaire ou turbulent |
| | *rhoPorousSimpleFoam* | Fluide permanent avec turbulence RANS et traitement poreux géométrie |
| | *rhoSimpleFoam* | Permanent, SIMPLE pour laminaire ou turbulent RANS |
| | *sonicDyMFoam* | Maillage dynamique, transitoire, gaz transsonique/supersonique, laminaire ou turbulent |
| | *sonicFoam* | Transitoire, gaz transsonique/supersonique, laminaire ou turbulent |
| | sonicLiquidFoam | Transitoire, transonqiue/supersonique, laminaire, fluide liquide |
| Multiphase | *bubbleFoam* | 2 fluides incompressibles non miscibles ( ex : eau gazeuse) |
| | *cavitatingFoam* | Transitoire, cavitation. |
| | *compressibleInterFoam* | 2 fluides compressibles, isothermes |
| | *interFoam* | 2 fluides incompressibles, isothermes |
| | *interMixingFoam* | 3 fluides incompressibles dont 2 miscibles |
| | *interPhaseChangeFoam* | 2 fluides incompressibles isothermes avec échange de phase |
| | *multiphaseInterFoam* | n fluides incompressibles |
| | *twoLiquidMixingFoam* | 2 fluides incompressibles |
| | *twoPhaseEulerFoam* | 2 fluides incompressibles dont une sous forme de gaz ou de bulles dispersés |

Tabeau 3.1: *Exemple de solveurs disponibles avec OpenFOAM.*

pour la pression, «U» pour la vitesse, «k» pour l'énergie cinétique turbulente). Le nom de ces dossiers temporels correspond au temps en secondes associé à ces valeurs (exp. «0», «1», «0.1253»). C'est dans le premier dossier temporel que l'utilisateur spécifie les conditions initiales et les conditions aux limites, afin que le solveur d'*OpenFOAM* «s'accroche» à elles pour commencer ses calculs et créer d'autres dossiers temporels avec les valeurs calculées par le solveur. On note que le format du nom des dossiers temporels peut être modifié dans le fichier *system/controlDict* [11].

#### 3.3.2.2 Dossier « system »

Le dossier «*system*» est un dossier fondamental qui contient un fichier utilisateur «*controlDict*» dans lequel celui-ci spécifie les contraintes temporelles du calcul (temps initial, temps final, intervalle de temps entre deux calculs, intervalle entre deux dossiers temporels, etc.). Le dossier «*system*» contient également deux fichiers utilisateurs «*fvSchemes*» et «*fvSol*ution» dans lesquels l'utilisateur paramètre les algorithmes de résolution utilisés. Ces trois fichiers sont indispensable à la lecture du cas sous *ParaView* ainsi que pour exécuter les commandes *OpenFOAM* et doivent être les premiers fichiers présents lors de la création d'un nouveau cas d'étude. Les fichiers «*fvSchemes*» et «*fvSolution*» sont habituellement copiés depuis le dossier «tutorials18» d'*OpenFOAM*. Bien qu'ils soient peu intuitifs pour le débutant, leur importance est capitale et il est indispensable de savoir s'en servir pour utiliser *OpenFOAM*. Ce dossier contient aussi, le cas échéant, les fichiers utilisateurs «*snappyHexMeshDict*» et «*decomposeParDict*» qui permettent d'utiliser l'algorithme de modification de maillage avec la commande «*snappyHexMesh*». Comme pour «*fvSchemes*» et «*fvSolution*», le fichier «*decomposeParDict*» peut être copié depuis un dossier de cas du dossier «*tutorials*».

#### 3.3.2.3 Dossier « constant »

Le dossier *constant* contient les fichiers utilisateurs liés aux propriétés indépendantes du temps telles que la viscosité du fluide dans « *transportProperties* », le modèle de turbulence de l'écoulement dans « turbulenceProperties » ou d'autres fichiers du même type. Ces fichiers ne sont pas indispensables à la lecture du cas sous ParaView et leur présence dépend du type d'écoulement à résoudre (non visqueux, turbulent, etc.). En outre, le dossier «*constant*» contient un sous-dossier «*polyMesh*» qui contient les fichiers machines relatifs au maillage : «boundary», «faces», «neighbour», «owner», «points», ainsi que le fichier utilisateur «*blockMeshDict*» qui permet de créer un maillage structuré de type «*blockMesh*». Le dossier «*constant*» peut également contenir un dossier «*triSurface*» qui contient la ou les géométrie(s) surfacique(s) STL dans le cas d'une modification de maillage via la commande «*snappyHexMesh*» [11].

### 3.3.3 Système de fichiers d'OpenFOAM

Avec *OpenFOAM*, l'utilisateur n'a pas affaire à une interface graphique avec des menus et des boites de dialogue comme avec un logiciel de CFD ou de maillage classique, mais à un système de fichiers. La connaissance de ce système de fichier est fondamentale pour toute personne utilisant *OpenFAOM*. Pour modifier ces fichiers, il fait appel à deux méthodes : la modification manuelle avec un éditeur de texte de fichiers compréhensibles pour un être humain, que nous appelons ici des «fichiers utilisateur» (exp. les fichiers « *blockMeshDict* » ou «*controlDict*»), ou la modification via l'exécution d'une commande dans une console Linux de fichiers appelés ici «fichiers machine» car ils ne sont dans la plupart des cas pas appréhendables par des êtres humains (exp. les fichiers «faces» ou «owner»). Le logiciel graphique *ParaView* permet seulement de visualiser les résultats de calculs en allant lire ces fichiers.

#### 3.3.3.1 Le fichier « controlDict »

Tout d'abord, le fichier «*controlDict*» est un fichier indispensable au fonctionnement d'*OpenFOAM* et doit se trouver dans le dossier «*system*» du dossier de travail. Ce fichier permet de paramétrer les pas de temps et l'écriture des dossiers temporels [11].

**Syntaxe**

Le fichier «*controlDict*» se présente comme suit :

```
ENTETE
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     SOLVEUR_UTILISE;

startFrom       TYPE;

startTime       VALEUR;

stopAt          TYPE;

endTime         VALEUR;

deltaT          VALEUR;

writeControl    TYPE;

writeInterval   VALEUR;

purgeWrite      VALEUR;

writeFormat     TYPE;

writePrecision  VALEUR;
```

```
writeCompression    TYPE;

timeFormat          TYPE;

timePrecision       VALEUR;

runTimeModifiable   TYPE;

// ************************************************************************* //
```

**Choix de l'application**

SOLVEUR_UTILISE doit être remplacé par le nom du solveur utilisé (e.g. «*simpleFoam*»).

**Temps de départ**

Le TYPE du mot clef «startFrom» est différent selon que l'on veut que le solveur «s'accroche» sur le dernier dossier temporel généré («latestTime»), sur le dossier temporel le plus ancien («firstTime»), ou sur un dossier définit plus loin dans le fichier «*controDict*» («starTime»).

Dans le troisième cas, la VALEUR de «startTime» est le nom du dossier temporel sur lequel le solveur s'accroche pour commencer les itérations.

**Temps de fin**

De même, le mot-clef «stopAt» peut être associé à des TYPES différents. Cependant, le plus commun est «endTime» qui permet d'arrêter les itérations à un temps définit plus loin dans le fichier «*controlDict*». Seulement dans le cas où l'utilisateur souhaite stopper les itérations instantanément, celui-ci peut remplacer le TYPE de «stopAt» par «writeNow».

Pour le premier cas, le temps d'arrêt des itérations doit être spécifié dans la VALEUR de «endTime».

**Temps entre deux calculs**

La VALEUR de «deltaT» doit être remplacée par la valeur de l'intervalle de temps en secondes entre deux pas de temps. Cette grandeur doit être spécifiée même pour un calcul utilisant un solveur stationnaire comme «simpleFoam». Aussi, ce intervalle de temps ne définie pas l'intervalle entre deux dossier temporels. En effet, dans la plupart des cas, il y a plusieurs pas de temps entre chaque dossier temporels.

**Temps entre deux dossiers temporels**

Le TYPE de «writeControl» est différent selon que l'on veut générer un dossier tous les n pas de temps («timeStep») ou tout les n secondes simulées («runTime»).

La VALEUR de «writeInterval» doit donc être remplacé par le nombre de pas de temps ou de secondes simulées entre deux dossiers temporels.

**Nombre maximum de dossiers temporels**

*OpenFOAM* permet également de spécifier le nombre de dossiers temporels à conserver sur le disque dur au fur et à mesure des itération. Ainsi, *OpenFOAM* peut

supprimer les dossiers temporels les plus anciens et ne garder qu'un nombre limité de dossiers les plus récents. Le nombre de ces dossier doit être spécifié dans la VALEUR de « purgeWrite ». Si l'on souhaite ne poser aucune limite dans le nombre de dossiers temporels, alors cette VALEUR doit être égale à zéro.

**Format des fichiers de sortie**

Dans *OpenFOAM*, contrairement à *Fluent* qui ne propose qu'une simple ou double précision, la précision des données de sorties est entièrement paramétrable : le nombre de décimales voulus doit être spécifié dans la VALEUR de «writePrecision». *OpenFOAM* offre la possibilité de générer les fichiers de données (exp. «p» ou «U») déjà compressés, ce qui permet de gagner de l'espace mémoire morte dans le cas de calculs volumineux. Pour activer cette option, il suffit de remplacer le TYPE de «writeCompression» par «compressed», sinon par «uncompressed». On note que les fichiers compressés peuvent être lus directement avec Paraview.

**Format des dossiers temporels**

*OpenFOAM* offre aussi la possibilité de choisir le format des noms des dossiers temporels. Ainsi, le TYPE de «timeFormat» est différent selon que l'on désire avoir des dossier dont le nom répond aux formats suivants :

| Format des dossiers temporels (exemples) | TYPE |
|---|---|
| « 0 » « -1.2 » « 15.364 » | « general » |
| « 0.000 » « -1.200 » « 15.364 » | « fixed » |
| « 0.000e00 » « 1.200e00 » « 1.536e01 » | « scientific » |

Tabeau 3.2: *Formats des dossiers temporels.*

Pour les deux derniers cas, le nombre de décimales doit être spécifié dans la VALEUR de «timePrecision».

**Prise en compte des modifications en cours d'exécution du solveur**

Enfin, un des grands atouts d'*OpenFAOM* est qu'il est possible de faire des modifications de ce fichier en cours d'itérations de façon à ce que ces modifications soit prises en compte instantanément par le solveur. Ainsi, si l'on veut allonger l'écart entre deux dossiers temporels en cours de calcul, il suffit de changer la VALEUR de «writeInterval» et de sauvegarder à nouveau le fichier «*controlDict*» sans avoir à interrompre le calcul.

Pour activer cette option, il faut remplacer le TYPE de «runTimeModifiable» par «yes», sinon par «no».

**Notre fichier « controlDict »**

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM Extend Project: Open Source CFD        |
|  \\    /   O peration     | Version:  1.6-ext                               |
|   \\  /    A nd           | Web:      www.extend-project.de                 |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
application     simpleFoam;
startFrom       latestTime;
startTime       0;
stopAt          endTime;
endTime         2000;
deltaT          1;
writeControl    timeStep;
writeInterval   100;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression compressed;
timeFormat      general;
timePrecision   6;
runTimeModifiable yes;
// ************************************************************************* //
functions
{
    #include "forceCoeffs"
}
```

#### 3.3.3.2 Conditions aux limites et initialisation

Il faut d'abord savoir que les conditions aux limites se définissent dans le dossier temporel sur lequel va «s'accrocher» le solveur (habituellement, le dossier «0»). Chaque grandeur à résoudre par le solveur doit être initialisée sur toutes les zones du cas d'étude ainsi que dans le domaine à l'intérieur de ce dossier temporel à l'aide de fichiers portant le nom des grandeurs en question :

| Type | Nom du fichier |
|---|---|
| pression | «p» |
| vitesse | «U» |
| intensité turbulente | «k» |
| dissipation turbulente | «epsilon» |

Tabeau 3.3: *Nom des fichiers de données dans le dossier «0».*

**Syntaxe**

Ces fichiers se présentent comme suit :

```
ENTETE
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      DIMENSION_DE_LA_GRANDEUR;

internalField   VALEUR_INITIALE_DE_CETTE_GRANDEUR_DANS_LE_DOMAINE;

boundaryField
{
    NOM_DE_LA_ZONE                                  ⎫
    {                                               ⎪
        type            TYPE_DE_LA_ZONE;            ⎬ Autant de fois qu'il y a de
        PARAMETRES                                  ⎪ zones dans le maillage
    }                                               ⎭
}

// ********************************************************************************* //
```

**Dimension**

Les dimensions, sous *OpenFOAM*, se présentent sous la forme d'une suite de chiffres représentants les puissances des unités S.I. dans un ordre particulier :

1. Masse [kg]
2. Longueur [m]
3. temp [s]
4. Température [K]
5. Quantité [kgmoles]
6. Courant [A]
7. Intensité lumineuse [cd]

– Ensemble, ils se mettent dans l'ordre : [ kg m s K kgmol A cd ]

– Exemples :

[ 0 1 -1 0 0 0 0 ] : $m/s$ ; [ 0 2 -1 0 0 0 0 ] : $m^2/s$ ; [ 1 -1 -2 0 0 0 0 ] : $kg/m/s^2 = Pa$

**Valeur initiale du domaine**

La valeurs initiale d'une grandeur le long d'une surface ou à l'intérieur d'un volume peut être non uniforme. Cependant, dans la plupart des cas, l'initialisation se fait avec des valeurs uniformes.

C'est pourquoi VALEUR_INITIALE_DE_CETTE_GRANDEUR_DANS_LE_DOMAINE, dans la syntaxe présentée ci-dessus, est souvent remplacée par le mot-clef «uniform» suivi de la valeur de la grandeur considérée (exp. «0» pour un scalaire ou «(1 0 0)» pour un vecteur).

**Conditions aux limites**

Viennent ensuite la définition des conditions aux limites proprement dites. Celles-ci ne peuvent être, à l'instar de *Fluent*, associées qu'à des zones : NOM_DE_LA_ZONE doit donc être remplacée par le nom de la zone définie dans *blockMesh* ou créée suite à l'exécution de *snappyHexMech* sur laquelle l'utilisateur veut définir une condition initiale. Dans tous les cas, la liste des zones est disponible dans le fichier «*constant/polyMesh/boundary*».

TYPE_DE_LA_ZONE doit ensuite être remplacée par le type de la zone. Bien qu'il y ait de nombreux types de zone possibles, les types classiques et leur description sont présentés dans le tableau suivant :

| **Type** | **Description** |
|---|---|
| «fixedValue» | valeur fixée par l'utilisateur |
| «zeroGradient» | valeur calculée avec un gradient perpendiculaire à la zone nulle en son point |
| «inletOutlet» | «fixedValue» lorsque la zone est une inlet et «zeroGradient» dans le cas contraire |
| «outletInlet» | «fixedValue» lorsque la zone est une outlet et «zeroGradient» dans le cas contraire |

Tabeau 3.4: *Types de zone classiques dans OpenFOAM.*

Ces types peuvent être différents selon la grandeur. Une «velocity inlet» n'aura donc pas le même type dans le fichier «U» et dans le fichier «p».

**Note** : Pour les zones modifiant implicitement la géométrie du domaine («empty», «symmetryPlane», «cyclic» et «wedge»), leur type doit être le même que celui définit dans le *blockMesh* ou, dans tous les cas, dans le fichier «*constant/polyMesh/boundary*».

Pour certains types de zones, des PARAMETRES supplémentaires sont nécessaires. Comme il est impossible de donner une forme standard de ces paramètre ne sont explicités dans le tableau suivant que ceux des types de zones présentés ci dessus :

| Type | PARAMETRES |
|---|---|
| «fixedValue», «epsilonWallFunction», «kqRWallFunction» | «value VALEUR ;» |
| «zeroGradient» | aucun paramètre |
| «inletOutlet» | «inletValue VALEUR ;» |
| «outletInlet» | «outletValue VALEUR ;» |
| «empty», «symmetryPlane», «cyclic», wedge» | aucun paramètre |

Tabeau 3.5: *Paramètres à ajouter aux types de zones définis plus haut.*

**Note 1** : Les VALEUR sont ici dans le même format que pour la valeur initiale du domaine : «uniform 0» pour un scalaire et «uniform (1 0 0)» pour un champ vectoriel.

**Note 2** : Les VALEUR pour les types comme «epsilonWallFunction» ou «kqRWall-Function» sont habituellement «0».

Une condition aux limites (C.L) s'établit donc un peu dans chaque fichier de donnée du premier dossier temporel du cas d'étude. Pour éviter toute ambiguïté, voici des

exemples de types de zone à mettre dans les différents fichiers.

| zone / fichier | «p» | «U» | «k» | « epsilon » |
|---|---|---|---|---|
| velocity inlet | « zeroGradient » | « fixedValue » | « fixedValue » | « fixedValue » |
| pressure outlet | « fixedValue » | « zeroGradient » | « zeroGradient » | « zeroGradient » |
| wall | « zeroGradient » | « fixedValue » | « kqRWallFunction » | « epsilonWallFunction » |

Tabeau 3.6: *Exemple de définition de (C.L) classiques.*

**Nos fichiers de conditions aux limites dans le dossier «0» :**

Le fichier d'initialisation "initialConditions" :

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.6                                   |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
flowVelocity          (40 0 0);
pressure              0;
turbulentKE           0.060;
turbulentOmega        4067.8;
#inputMode            merge
// ************************************************************************* //
```

Le fichier «p» :

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM Extend Project: Open Source CFD        |
|  \\    /   O peration     | Version:  1.6-ext                               |
|   \\  /    A nd           | Web:      www.extend-project.de                 |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include        "initialConditions"
dimensions      [0 2 -2 0 0 0 0];
internalField   uniform $pressure;
boundaryField
{
    "inlet|ground|Ahmad_Body_solid"
      { type            zeroGradient;}

    outlet
    {type fixedValue;value $internalField;}

    "sides|top"
      {   type symmetryPlane;}
```

```
}
// ************************************************************************* //
```

Le fichier «U» :

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM Extend Project: Open Source CFD        |
|  \\    /   O peration     | Version:  1.6-ext                               |
|   \\  /    A nd           | Web:      www.extend-project.de                 |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include        "initialConditions"
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform $flowVelocity;
boundaryField
{
    inlet
    { type  fixedValue;
      value $internalField;
    }
    outlet
    { type inletOutlet;
      inletValue uniform (0 0 0);
      value $internalField;
    }
    "sides|top"
    { type symmetryPlane;
    }
    "ground|ABody_solid|Roues_solid"
    { type fixedValue;
      value uniform (0 0 0);
    }
}
// ************************************************************************* //
```

Le fichier «k» :

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.0.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
```

```
    object      k;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0];
internalField   uniform 0.060;
boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0.060;
    }
    outlet
    {
        type            inletOutlet;
        inletValue      uniform 0.060;
        value           uniform 0.060;
    }
    ground
    {
        type            kqRWallFunction;
        value           uniform 0.060;
    }
    sides
    {
        type            symmetryPlane;
    }
    top
    {
        type            symmetryPlane;
    }
    "ABody_solid|Roues_solid"
    {
        type            kqRWallFunction;
        value           uniform 0.060;
    }
}
// ************************************************************************* //
```

Le fichier «omega» :

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.0.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{     version     2.0;
    format      ascii;
    class       volScalarField;
    object      omega;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include        "initialConditions"
dimensions      [0 0 -1 0 0 0 0];
internalField   uniform $turbulentOmega;
boundaryField
```

```
{
    inlet
    {
        type            fixedValue;
        value           $internalField;
    }
    outlet
    {
        type            inletOutlet;
        inletValue      $internalField;
        value           $internalField;
    }
    ground
    {
        type            omegaWallFunction;
        value           $internalField;
    }
    Ahmad_Body_solid
    {   type            omegaWallFunction;
        value           $internalField;
    }
    top
   {
       type             symmetryPlane;
   }
   sides
   {
      type              symmetryPlane;
   }
}
// ************************************************************************* //
```

#### 3.3.3.3 Fichiers « fvSchemes » et « fvSolutions »

Lors de l'utilisation d'*OpenFOAM*, voici les fichiers qui sont peut être les moins intuitifs. Cependant, ils sont indispensables et conditionnent la convergence des calculs. En effet, les fichiers «fvSchemes» et «fvSolution», qui doivent impérativement être présents dans le dossier «*/system* » du dossier de travail, permettent à l'utilisateur de choisir les méthodes d'interpolation des solveurs ou de définir les critères de convergence.

Ces fichier étant trop complexes pour les éditer à la main à partir de zéro, il est recommandé d'utiliser les fichiers présents dans le dossier «tutorials», en essayant d'assurer la plus grande similitude possible entre le cas de travail et celui dont on prend les fichiers et de ne les modifier qu'en cas de besoin (divergence, incompatibilité avec le cas d'étude, etc.).

**Le fichier «fvSchemes»** Le fichier «*fvSchemes*» permet de choisir les méthodes de résolution des opérateurs mathématiques tels que les divergences, les laplaciens ou les gradients. Pour plus d'informations sur ces modèles, consulter le User Guide d'*OpenFOAM* [11].

En cas de divergence, l'utilisateur peut tout de même vérifier par lui même si dans ce fichier l'algorithme associé à la dérivée temporelle («ddtSchemes») n'est pas «steadyState» dans le cas d'un calcul avec un solveur instationnaire (exp. «*pisoFoam*»). Inversement, s'il s'agit d'un autre algorithme que «steadyState» (exp. «Euler») dans le cas d'un calcul avec un solveur stationnaire (exp. «simpleFoam»), cela peut expliquer la divergence.

**Le fichier «fvSolution»** Le fichier «*fvSolution*» permet de paramétrer les critères de convergence des différentes grandeurs. Pour une étude détaillée de ce fichier, voir le User Guide [11]. L'utilisateur peut tout de même modifier sans crainte la valeur des paramètres «tolerance» et «relTol» de chaque grandeur. En effet, «tolerance» est la valeur du résidu à partir duquel les itérations cessent dans un pas de temps. Cependant, même en n'ayant pas atteint cette valeur de résidu, les itérations peuvent s'arrêter si la différence entre le résidu final et le résidu initial est assez grande. C'est le rôle du paramètre «relTol» qui défini le rapport du résidu final sur le résidu initial en dessous duquel les itérations s'arrêteront. Une valeur nulle de «relTol» force le résidu à descendre en dessous de la valeur de «tolerance» et d'arrêter les itérations.

En cas de divergence, l'utilisateur peut vérifier si toutes les grandeurs à calculer sont bien présentes dans ce fichier. Il peut également vérifier si le dictionnaire «*SIMPLE*» est présent pour les cas utilisant des solveur stationnaires tels que «*simpleFoam*» et si le dictionnaire «*PISO*» est présent dans le cas de l'utilisation de solveurs instationnaires comme «*icoFoam*» ou «*pisoFoam*».

Dans certains cas, ce fichier peut contenir un dictionnaire définissant les facteurs de sous-relaxation qui peuvent également être modifiés par l'utilisateur.

**Vérifier le maillage** Il est possible, tout comme dans Fluent, d'avoir un compte-rendu du maillage avec la commande «*checkMesh*».

#### 3.3.3.4 Calcul des coefficients de portance et de trainée

Il est possible, avant de lancer les itérations, de demander à *OpenFOAM* de calculer les coefficients de trainée ($C_d$) et de portance ($C_l$) d'une géométrie. Pour cela, il faut ajouter au fichier «*controlDict*» les lignes suivantes :

```
functions
(
  forceCoeffs
  {
        type              forceCoeffs;

        functionObjectLibs ("libforces.so");

        outputControl     outputTime;

        patches (NOM_DES_ZONES_CONCERNEES);

        rhoName     rhoInf;
        pName p;
        Uname U;

        log true;

        rhoInf MASSE_VOLUMIQUE_INFINI_AMONT_EN_KG_PAR_METRE_CUBE;

        CofR (0 0 0);

        liftDir     COORDONNEES_DU_VECTEUR_Z;
        dragDir     COORDONNEES_DU_VECTEUR_X;
```

```
        pitchAxis (0 0 0);

        magUInf MAGNITUDE_DE_LA_VITESSE_INFINIE_AMONT;

        lRef LONGUEUR_DE_REFRENCE_EN_METRES;
        Aref SURFACE_DE_REFRENCE_EN_METRES_CARRES;
  }
);
```

### Notre fichier «forceCoeffs»

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.0.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
forces
{
    type         forceCoeffs;
    functionObjectLibs ( "libforces.so" );
    outputControl timeStep;
    outputInterval 1;
    patches      ( "ABody_solid" );
    pName        p;
    UName        U;
    rhoName      rhoInf;
    log          true;
    rhoInf       1;
    liftDir      (0 0 1);
    dragDir      (1 0 0);
    CofR         (-0.5022 0 0);
    pitchAxis    (0 1 0);
```

```
    magUInf      40.0;
    lRef         1.044;
    Aref         0.112;
}
// ************************************************************************* //
```

#### 3.3.3.5 Fichiers liés à l'écoulement

Enfin, les fichiers importants pour le bon fonctionnement d'*OpenFOAM* sont les fichiers tels que «*transportProperties*» ou «*turbulenceProperties*» qui se trouvent dans le dossier «*/constant*» du dossier de travail.

Ces fichier, souvent très concis, permettent de renseigner le solveur quand aux propriétés de l'écoulement tel que sa viscosité cinématique ou son modèle de turbulence.

Pour faire simple, nous allons ici traiter trois cas : un cas de fluide non visqueux, un cas de fluide visqueux laminaire et un cas de fluide visqueux turbulent.

**Cas de fluide non visqueux** Dans ce cas, bien qu'il n'ait pas été testé dans le cadre de ce projet, il semblerait qu'aucun fichier supplémentaire ne soit requis.

**Cas du fluide visqueux laminaire** Ce cas ne nécessite qu'un fichier : le fichier «*transportProperties*» qui ne contient que la valeur de la viscosité cinématique du fluide :

```
ENTETE
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

nu              nu [ 0 2 -1 0 0 0 0 ]       VALEUR;

// ************************************************************************* //
```

**Cas de fluide visqueux turbulent** Ce cas nécessite trois fichiers :

- Toujours le fichier « *transportProperties* » mais augmenté d'une ligne renseignant sur le modèle de viscosité utilisé :

```
ENTETE
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
transportModel  MODELE;

nu              nu [ 0 2 -1 0 0 0 0 ]       VALEUR;

// ************************************************************************* //
```

le **MODELE** le plus commun étant bien entendu «Newtonian». Pour les autres modèles, se référer à la documentation officielle [11].

- Un fichier «*turbulenceProperties*» renseignant sur le choix de l'utilisateur de travailler en RANS ou en LES :

```
ENTETE
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType  MODELE;
```

MODELE valant «RASModel» ou «LESModel».

– Un fichier «*RASProperties*» ou «*LESProperties*» renseignant sur le modèle de turbulence. Par exemple :

```
ENTETE
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

RASModel            MODELE;

turbulence          on;

printCoeffs         on;

// ************************************************************************* //
```

**MODELE** pouvant valoir « kEpsilon », « kOmega », « RNGEpsilon » ou autres visibles dans la documentation officielle [11].

### Notre fichier « transportProperties »

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM Extend Project: Open Source CFD        |
|  \\    /   O peration     | Version:  1.6-ext                               |
|   \\  /    A nd           | Web:      www.extend-project.de                 |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
transportModel  Newtonian;
nu              nu [0 2 -1 0 0 0 0] 1.5e-05;
// ************************************************************************* //
```

### Notre fichier « turbulenceProperties »

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM Extend Project: Open Source CFD        |
|  \\    /   O peration     | Version:  1.6-ext                               |
|   \\  /    A nd           | Web:      www.extend-project.de                 |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
```

```
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
simulationType  RASModel;
// ************************************************************************* //
```

**Notre fichier « RASProperties »**

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM Extend Project: Open Source CFD        |
|  \\    /   O peration     | Version:  1.6-ext                               |
|   \\  /    A nd           | Web:      www.extend-project.de                 |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      RASProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
RASModel            kOmegaSST;
turbulence          on;
printCoeffs         on;
// ************************************************************************* //
```

La figure (Fig.3.2) nous résume les conditions aux limites de notre problème.

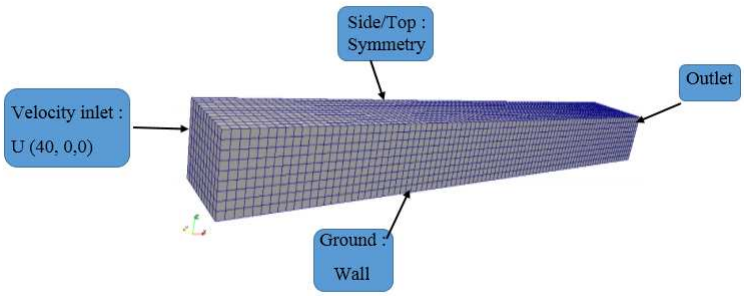



FIGURE 3.2: *Conditions aux limites.*

Maintenant, et après avoir remplis tous ces fichiers, nous sommes prêt à lancer le calcul avec la commande suivante :

– ~/ABody$ **pyFoamPlotRunner.py simpleFoam**

Le solveur utilisé est *simpleFoam* précédé d'une commande qui permet de visualiser les résidus.

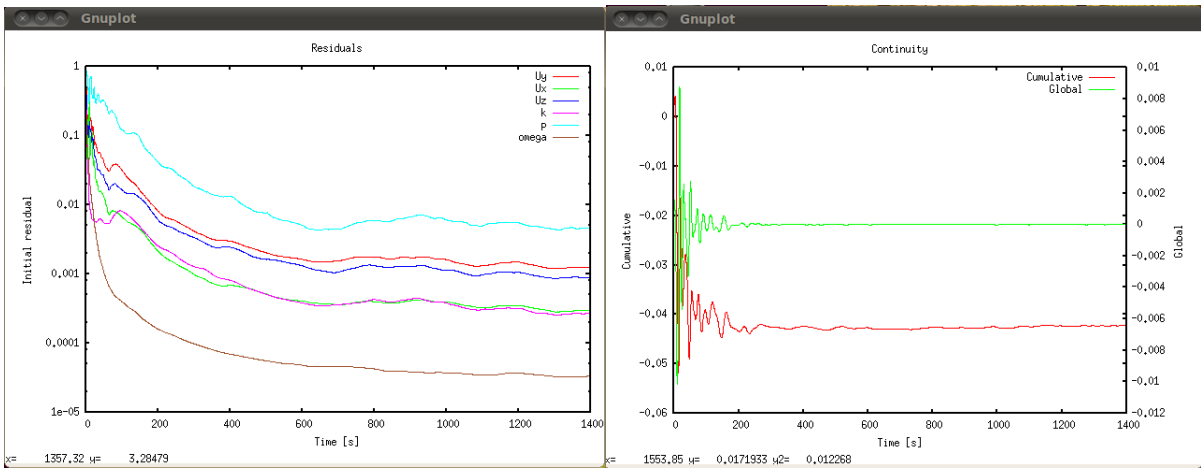



FIGURE 3.3: *Résidus visualisés avec pyFoamPlotRunner.*

Le temps de calcul dépond du nombre de noeuds que nous avons utilisé (environ 1h00 pour 1 Million). Une fois le calcul terminé, on lance les commandes ci-dessous en fonction des besoins pour calculer les paramètres de l'écoulement. Il existe beaucoup d'autres commandes qui sont en fait des outils dévelopés avec *OpenFOAM*.

1. ~/ABody$ **foamCalc components U -latestTime**
2. ~/ABody$ **foamCalc mag U -latestTime**
3. ~/ABody$ **vorticity -noZero**
4. ~/ABody$ **yPlusRAS -latestTime**
5. ~/ABody$ **sample -latestTime**
6. ~/ABody$ **foamToVTK -latestTime**

Le répertoire de travail sera alors enrichi de la manière suivante :

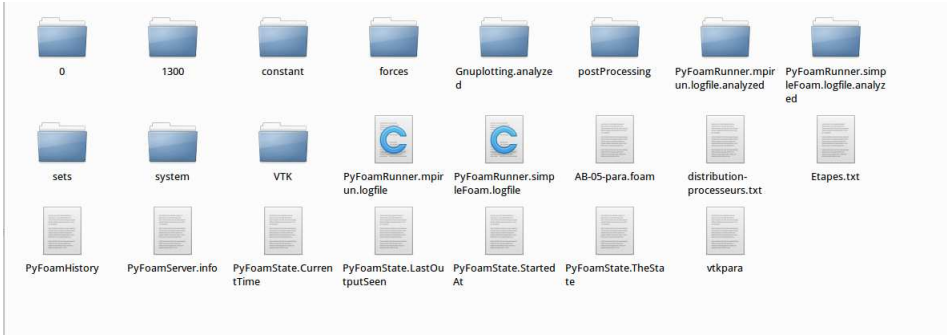



FIGURE 3.4: *Répertoire de travail après calculs.*

## 3.4 Conclusion

Dans cette partie, nous avons essayé de donner le maximum de détails possibles conçernant le code *OpenFOAM* ainsi que les outils qu'il utilise pour mener à bien une simulation numérique. A première vue, le lecteur habitué avec d'autres codes commerciaux tels que *Fluent*, *Ansys*, *StarCCM+*, *Comsol* ou autres, est complètement dérouté. Mais avec la pratique sur plusieurs exemples à travers les tutoriaux livrés avec, on s'habitue de plus en plus et il devient beaucoup plus intéressant que les autres codes. Il est intéressant de noter que chacun peut créer son propre solveur, en fonction de ses besoins, en se basant sur ceux qui existent déjà.

Nous allons, dans le prochain chapitre, montrer comment utiliser les outils Open-Source pour visualiser nos résultats.

# CHAPITRE 4

# RESULTATS

## 4.1 Introduction

Dans ce dernier chapitre, nous allons voir les outils OpenSource que nous offre la distribution CAELinux et qui sont destinés à visualiser les résultats de nos simulations.

Afin de ne pas allourdir inutilement le document, nous avons préféré mettre certains détails en annexe et omis volontairement d'autres conçernant par exemple l'interprétation de certains résultas vu que l'objectif de notre travail est surtout l'utilisation des logiciels OpenSource.

### 4.1.1 ParaView

*ParaView* est un logiciel libre de visualisation (2D et 3D) scientifique de données. Il est fondé sur la bibliothèque VTK et publié sous licence BSD. Il est développé principalement par le *Sandia National Laboratories*, le *Los Alamos National Laboratory* et la société *Kitware Inc.* ParaView dispose d'une architecture client-serveur qui permet de traiter des données à distance ; typiquement, pour un volume de données important, tout ou partie du traitement est effectué par un serveur, le poste de travail servant uniquement à l'affichage. En visualisation volumique, il réduit le niveau de détail des objets éloignés afin de maintenir une bonne vitesse d'affichage [13]. On trouve en annexe C les détails d'utilisation de cet outils très performant qui a été conçu principalement pour traîter de grandes quantités de données et pour un travail parallèle.

### 4.1.2 QtiPlot

*QtiPlot* est aussi une plate-forme de visualisation scientifique 2D et 3D. Il est idéal pour les étudiants et propose une alternative aux logiciels propriétaires extrêmement

coûteux comme *Origine*, *SigmaPlot*, *SPSS*, *Regressi* ou *Igor Pro*. *QtiPlot* est utilisé pour l'enseignement ainsi que pour l'analyse de données complexes et la visualisation dans les entreprises, les lycées, les universités et les instituts de recherche du monde entier. On trouvera dans la référence [14] une courte liste d'ouvrages scientifiques citant *QtiPlot*. Notons que nous utilisé ce logiciel uniquement pour les résultats 2D.

## 4.2 Ecoulement autour du corps Ahmed

Nous allons exploiter les résultats que nous avons trouvé durant les simulations que nous avions effectué sous OpenFOAM. On trouve dans la littérature un grand nombre de travaux relatifs à l'écoulement autour du *corps Ahmed* avec différents angles obliques. Nous avons choisi uniquement le cas le plus utilisé avec un angle de 25° et nous avons trouvé des résultats très comparables à ceux trouvés par différents auteurs [5-10, 15].

Nous avons effectué un grand nombre de simulations et nous allons nous limiter uniquement aux résultats les plus significatifs.

### 4.2.1 Répartition de vitesse

La figure (Fig.4.1) ci-dessous nous montre les profils de vitesse derrière le corps Ahmed en fonction de la distance (en mm).

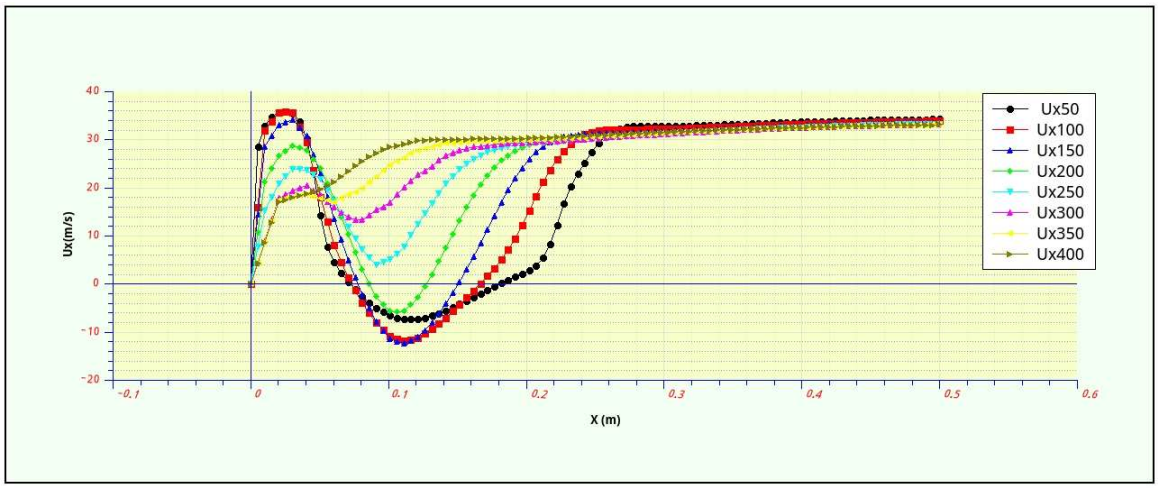



FIGURE 4.1: *Profils de vitesse derrière le corps Ahmed.*

Ci-dessous (Fig.4.2) la répartition de vitesse derrière le corps Ahmed à différentes distances.

La figure suivante (Fig.4.3) nous montre la répartition de vitesse sous le corps Ahmed à différentes distances selon z.

Les lignes de courant sont présentées sur la figure (Fig.4.4).

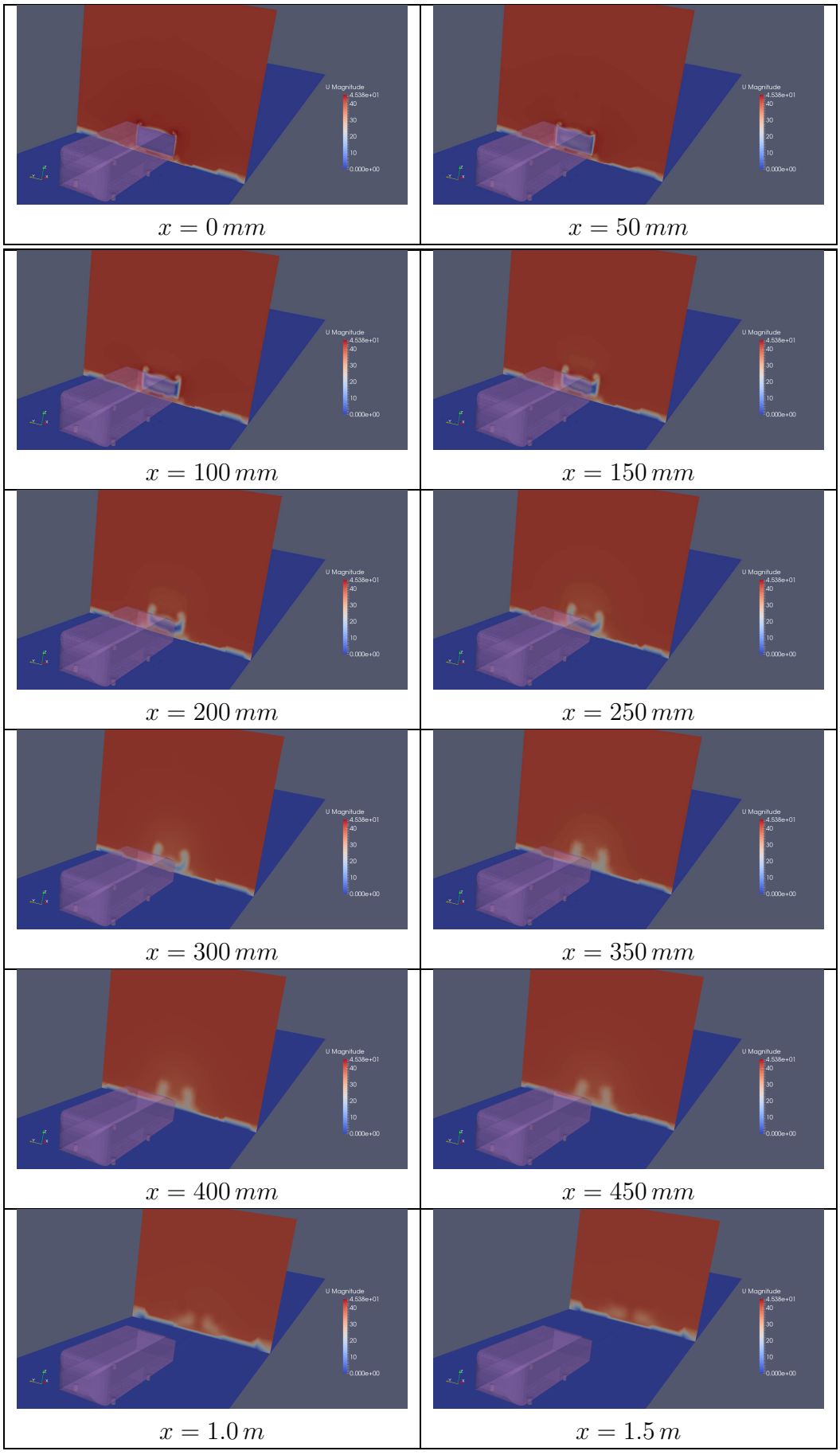



FIGURE 4.2: *Répartition de vitesse derrière le corps Ahmed.*

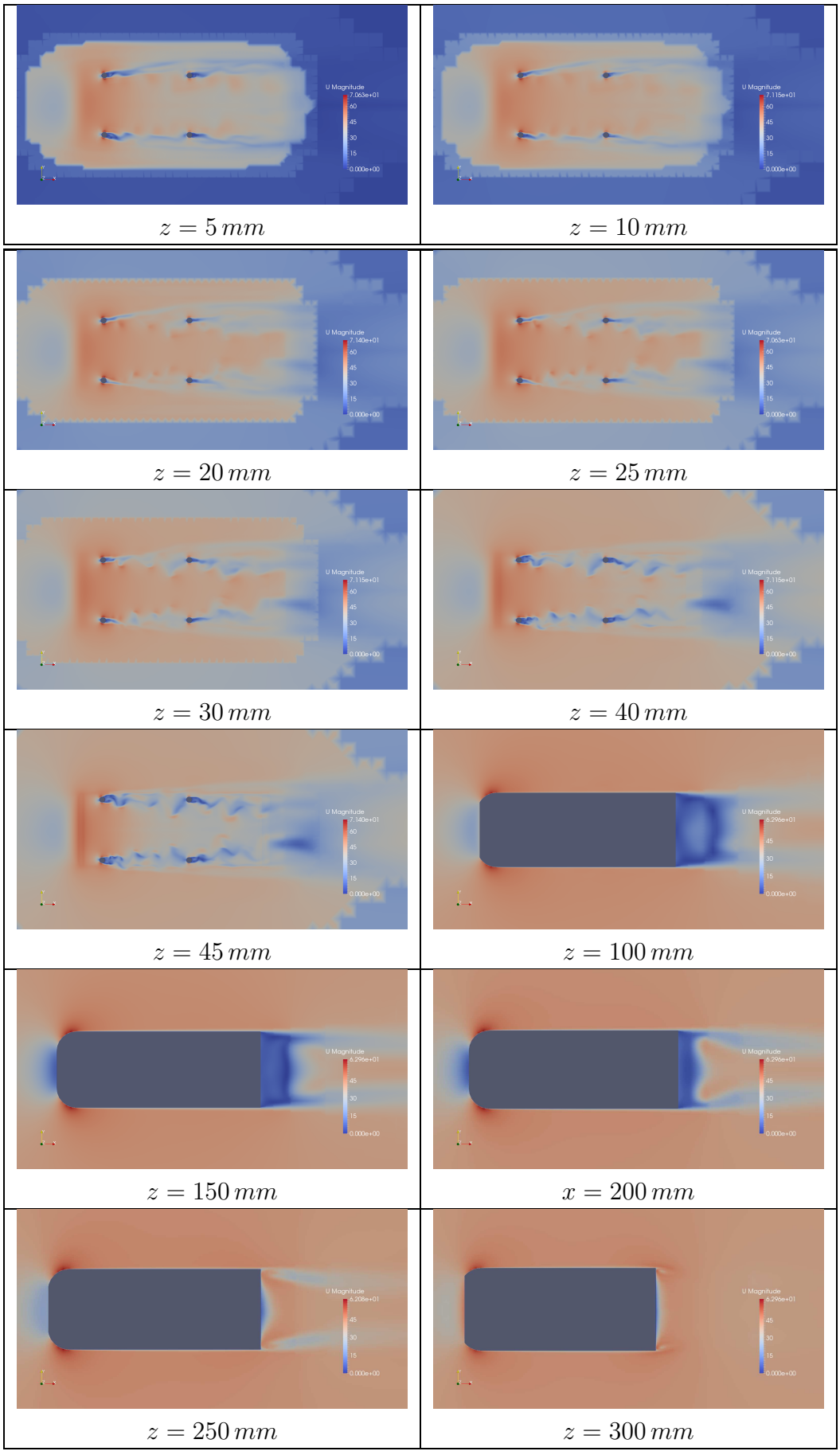



FIGURE 4.3: *Répartition de vitesse sous le corps Ahmed.*

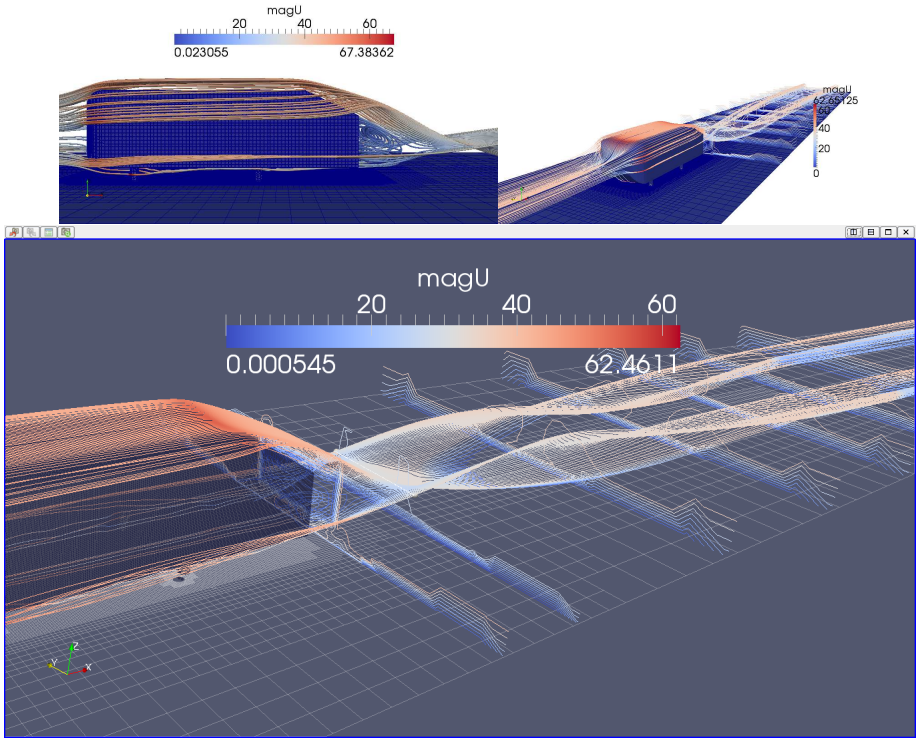



FIGURE 4.4: *Lignes de courant.*

### 4.2.2 Répartition de pression

Nous présentons ci-dessous (Fig.4.5) la répartition de pression autour du corps Ahmed.

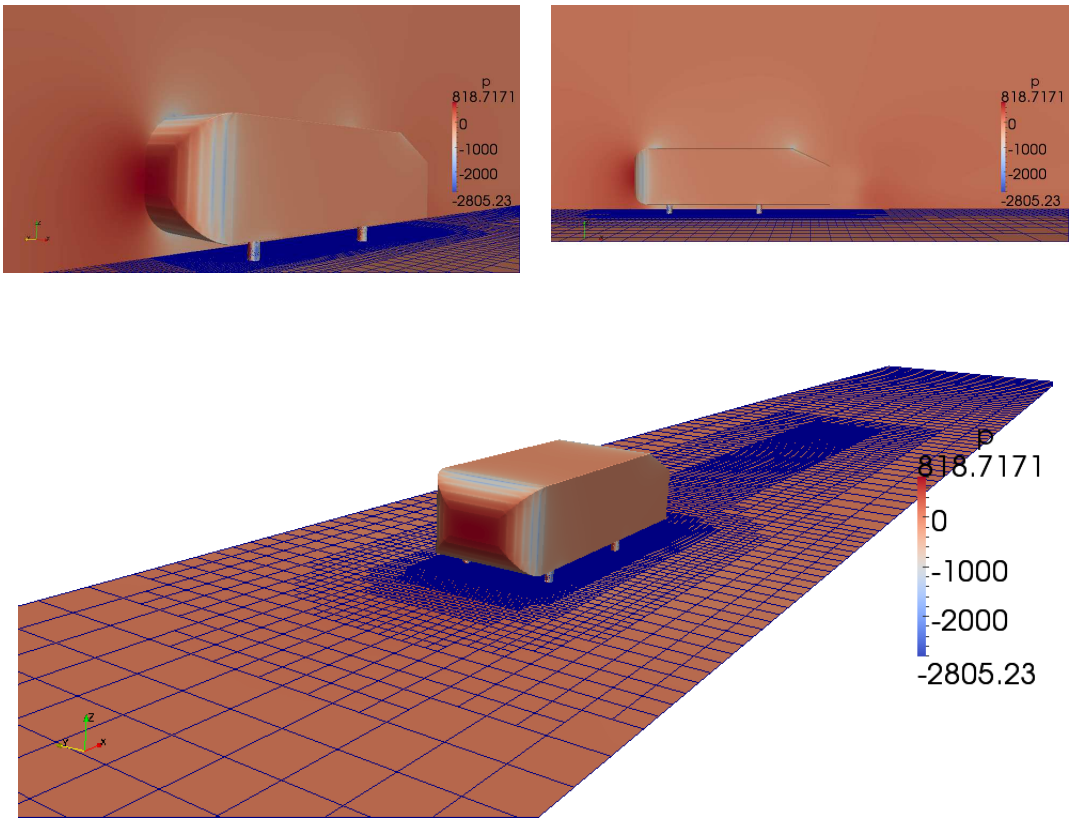



FIGURE 4.5: *Répartition de pression autour du corps Ahmed.*

La figure (Fig.4.6) montre aussi la répartition de la pression sur les roues.

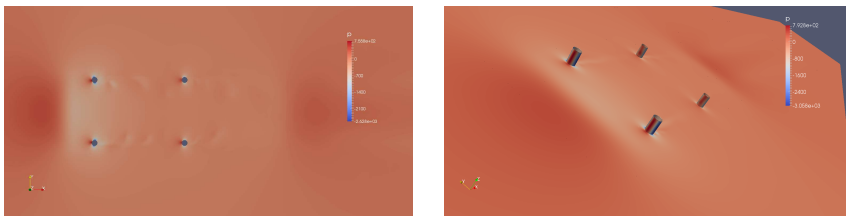


FIGURE 4.6: *Répartition de pression autour du corps Ahmed.*

### 4.2.3 Répartition de la turbulence

La figure (Fig.4.7) ci-dessous nous montre la répartition de l'énergie cinétique turbulente derrière le corps Ahmed en fonction de la distance.

La figure (Fig.4.8) nous montre la répartition de l'énergie cinétique turbulente sous le corps Ahmed à différentes hauteurs.

L'écoulement tourbillonaire derrière le corps Ahmed est montré sur la figure (Fig.4.9).

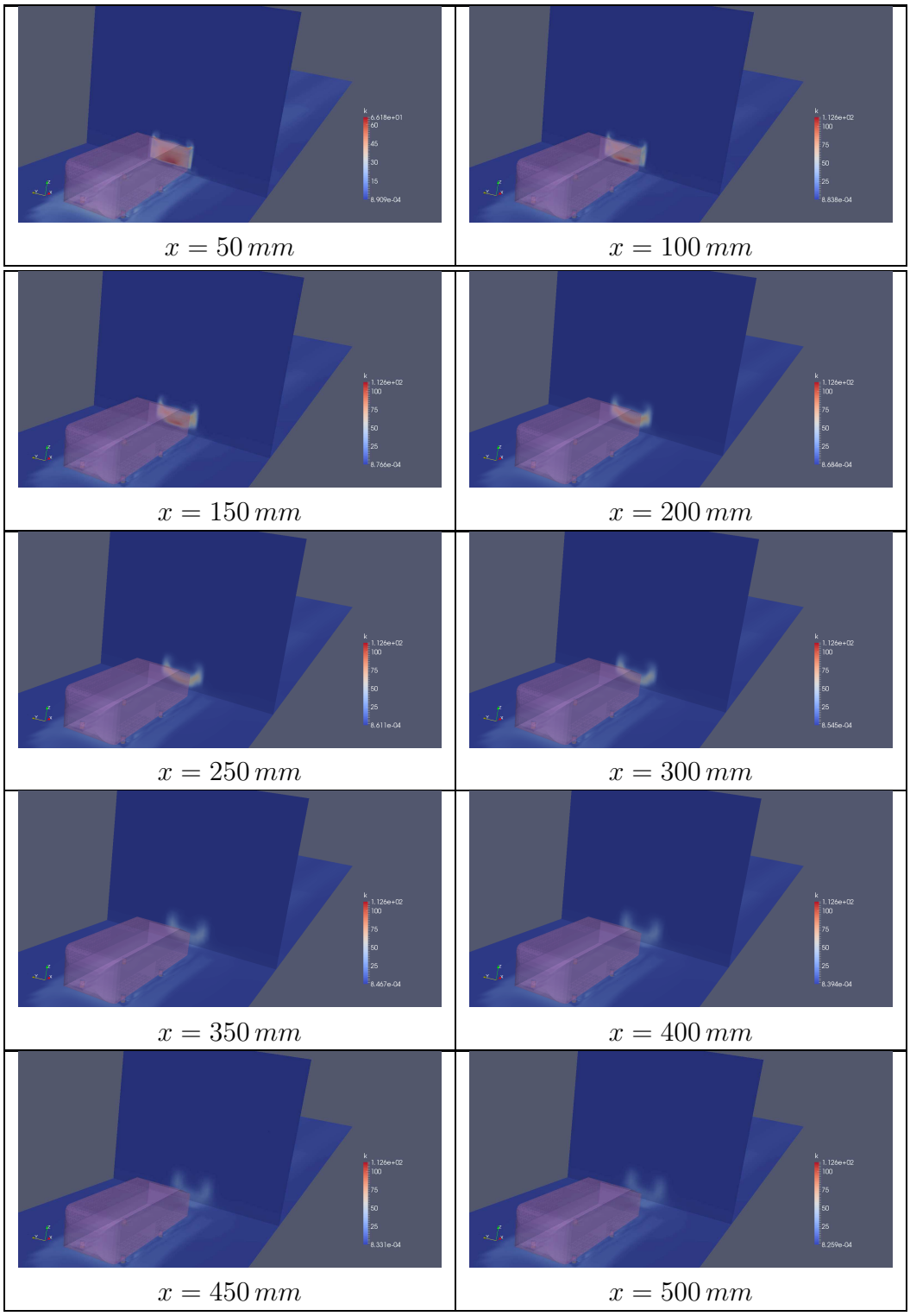



FIGURE 4.7: *Répartition de l'énergie cinétique turbulente derrière le corps Ahmed.*

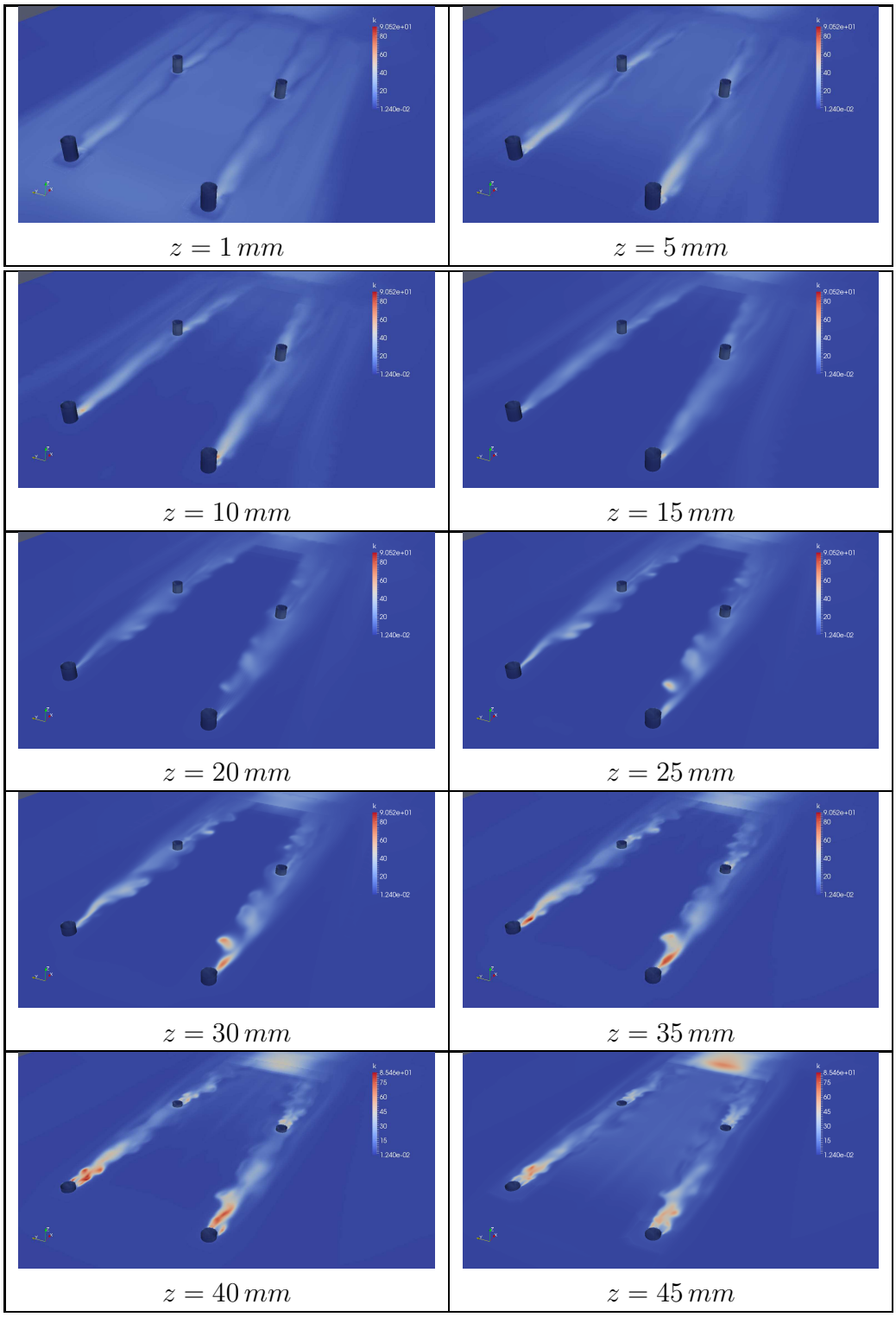



FIGURE 4.8: *Répartition de l'énergie cinétique turbulente sous le corps Ahmed.*

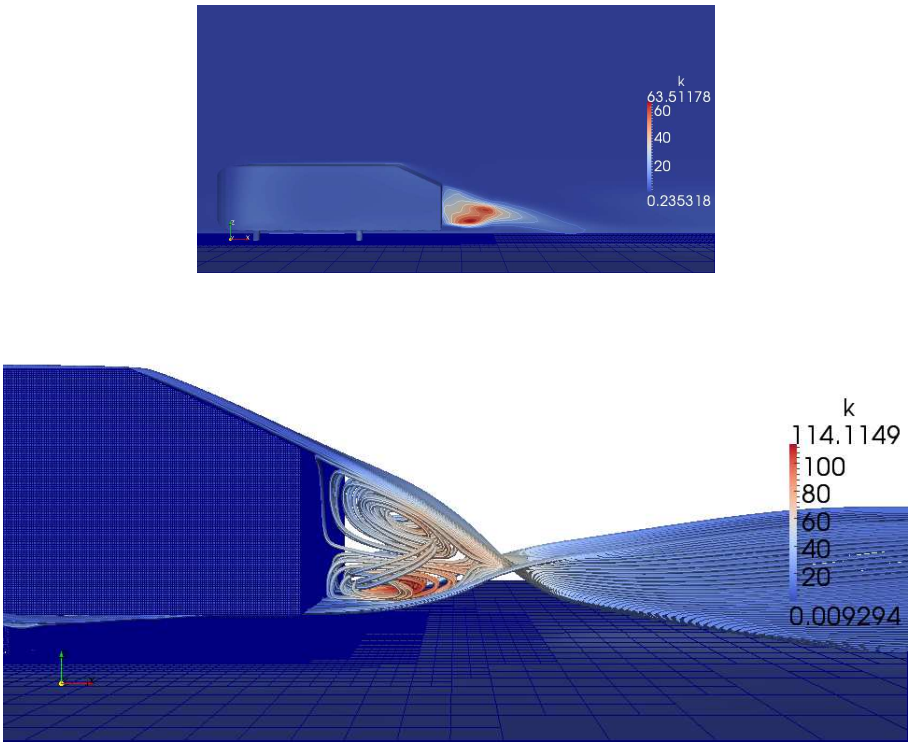



FIGURE 4.9: *Tourbillon développé derrière le corps Ahmed.*

### 4.2.4 Coefficient de traînée

L'objectif principal de notre travail est la détermination du coefficient de traînée $C_d$. Nous avons tracé l'évolution de ce paramètre primordial en aérodynamique en fonction du nombre d'itération (Fig.4.10). En consulant le fichier de calcul dans le répertoire */force/500/forceCoeffs.dat*, nous avons les valeurs :

$C_d = 0.2946$ et $C_l = 0.3681$.

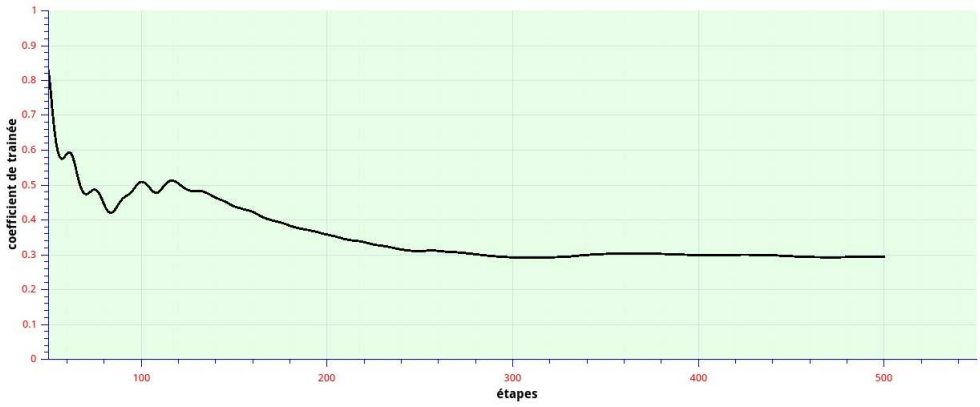



FIGURE 4.10: *Évolution du coefficient de traînée.*

## 4.3 Conclusion

Nous constatons à travers la qualité des graphiques présentés dans les résultats, la puissance des logiciels de visualisation *OpenSource* livrés gratuitement avec la distribution CAELinux. Nous pouvons dire que *ParaView* est aussi puissant et plus intuitif que *Tecplot*. De même *QtiPlot* est plus facile à utiliser que son homologue *Origine* sous Windows.

Les résultats que nous avons trouvés sont en bon accord avec la majorité des résultats trouvés par d'autres chercheurs [5-10, 15].

# CONCLUSION GÉNÉRALE

Notre expérience dans l'utilisation des logiciels OpenSource pour résoudre un problème de CFD nous a ammené faire les constatations suivantes :

**Avantages**

- Gratuit (pas de limitations dues aux licences) ;
- accès aux sources (pas une "boîte noire") ;
- un outil supplémentaire pour les benchmarks code-to-code ;
- bénéficie de mises à jour régulières ;
- de nombreux solveurs "clé en main" ;
- facilité pour programmer des équations ;
- une communauté réactive (forum, congrès, université d'été...etc) ;

**Inconvénients**

- Commandes unix et C++ ;
- Temps de prise en main ;
- La documentation : soit manquante soit en anglais ;
- Pas d'interface graphique dans le pluspart des cas ;

Malgré ces inconvénients, nous avons eu un grand plaisir à travailler d'une manière générale sous l'environnement Linux et, particulièrement, avec la ligne de commande. Les codes livrés avec la distribution *CAELinux* que nous avons utilisé sont très puissant et les résultats que nous avons trouvé conçernant l'écoulement autour du modèle Ahmed sont très satisfaisant.

# BIBLIOGRAPHIE


[1] Essai publié dans le livre Open Sources — Voices from the Open Source Revolution, ISBN 1-56592-582-3, janvier 1999, édité par Chris DiBona, Sam Ockman, et Mark Stone chez O'Reilly & Associates. Une version en français est en cours d'adaptation par et pour les Éditions O'Reilly. Bruce Perens détient le copyright sur ce texte et l'a publié sous les termes de la licence publique générale de GNU, version 2 ou ultérieure.

[2] S. R. Ahmed, G. Ramm, and G. Faltin. *Some salient features of the time averaged ground vehicle wake.* Technical Report TP-840300, Society of Automotive Engineers, Warrendale, Pa., 1984.

[3] http ://fr.wikipedia.org/wiki/Linux

[4] http ://caelinux.com

[5] C.-H. Bruneau, P. Gilliéron, and I. Mortazavi. *Flow manipulation around the ahmed body with a rear window using passive strategies.* Comptes Rendus Mécanique, 335(4) :213-218, 2007.

[6] G. Franck, N. Nigro, M. Storti, and J. D'Elia. *Numerical simulation of the ahmed vehicle model near-wake.* Technical report, Instituto de Desarrollo Tecnologico para la Industria Quimica, Argentina, 2007.

[7] E. Guilmineau. *Computational study of flow around a simplified car body.* Journal of Wind Engineering and Industrial Aerodynamics, 96(6-7) :1207-1217, 2008.

[8] R. J. A. Howard and M. Pourquie. *Large eddy simulation of an ahmed reference model.* Journal of Turbulence, 3 :012, 2002.

[9] C. Hinterberger, M. Garcia-Villalba, and W. Rodi. *Large eddy simulation of flow around the ahmed body.* In R. McCallen, F. Browand, and J. Ross, editors, Lecture Notes in Applied and Computational Mechanics, The Aerodynamics of Heavy Vehicles : Trucks, Buses, and Trains, pages 77-87. Springer Verlag, 2004.

[10] H. Lienhart, C. Stoots, and S. Becker. *Flow and turbulence structures in the wake of a simplified car model (ahmed model).* Technical report, Lehrstuhl fur Strômungsmechanik (LSTM), Universitàt Erlangen-Nùrnberg, Cauerstr. 4, 91058 Erlangen, Germany, 2000.

[11] Site d'OpenFOAM / User Guide utility : http ://www.openfoam.com/docs/user

[12] http ://fr.wikipedia.org/wiki/OpenFOAM#cite_ref-2

[13] http ://fr.wikipedia.org/wiki/ParaView

[14] http ://www.qtiplot.com/

[15] Stefan Bordei, Florin Popescu. *Aerodynamic results for a notchback race car.* The annals of DUNAREA DE JOS. University of GALATI FASCILE V, Technologies in Machine Building, ISSN 1221-4566, 2011.

[16] M. Baudouin DEBAIN et M. William TOUGERON "Présentation, essai et validation du logiciel open-source ", Projet de Fin d'Etudes ,Institut Polytechnique des Sciences Avancées ,Année 2010-2011

[17] T. Grahs "Generic external aerodynamic simulation training ",Inst. Scientific Computing, TU Braunschweig 2012

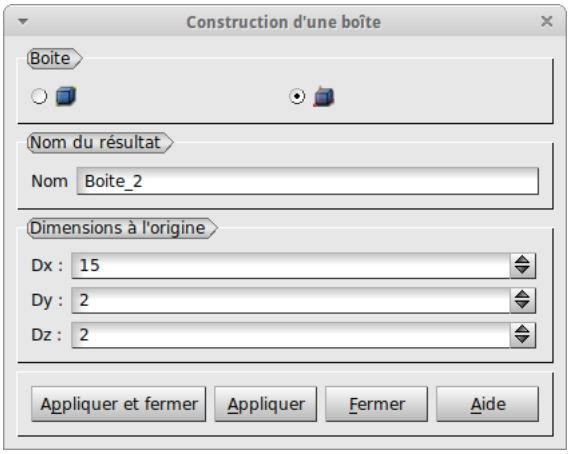
# Annexes

# ANNEXE A

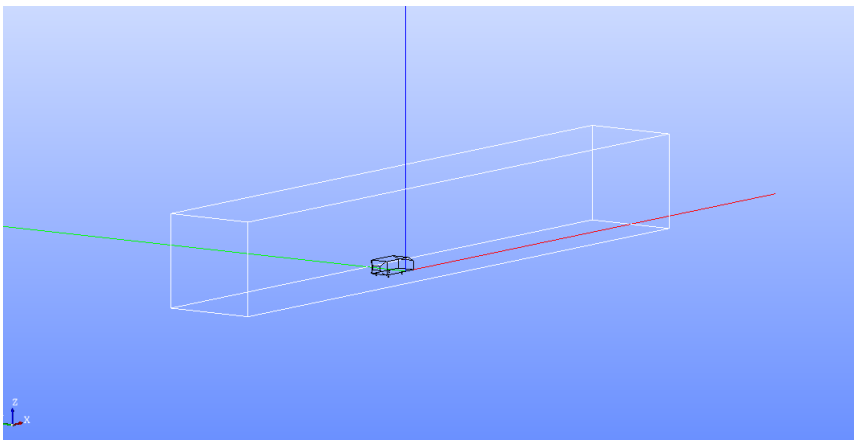
# MAILLAGE PAR SALOME

## A.1 Création du tunnel

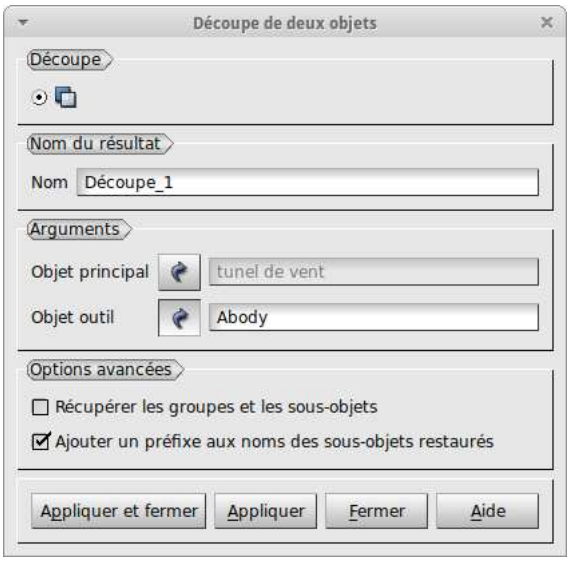
– Appuyez sur le bouton 



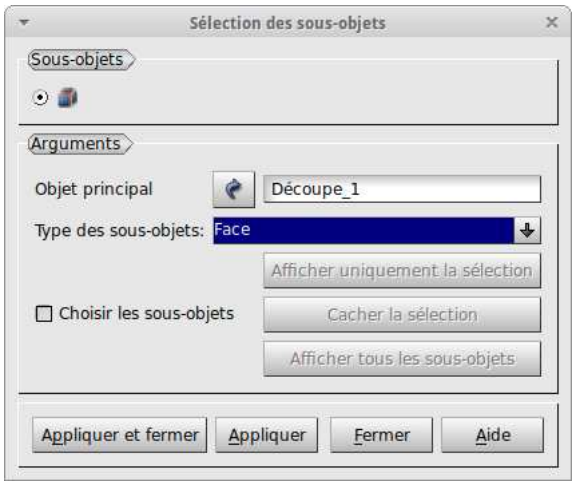

– Puis sur  et régler le position du tunnel pour obtenir



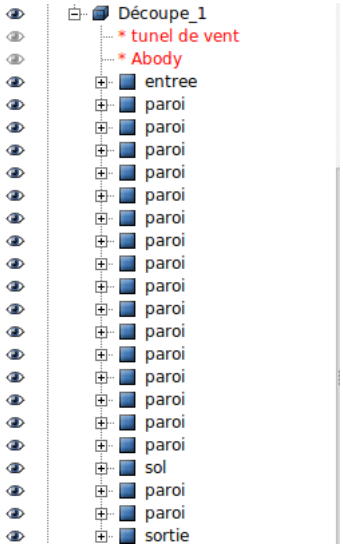
– Puis sur  pour découper




– Appuyez sur *Appliquer et fermer,* ensuite appuyez sur  pour sélectionner des sous objets

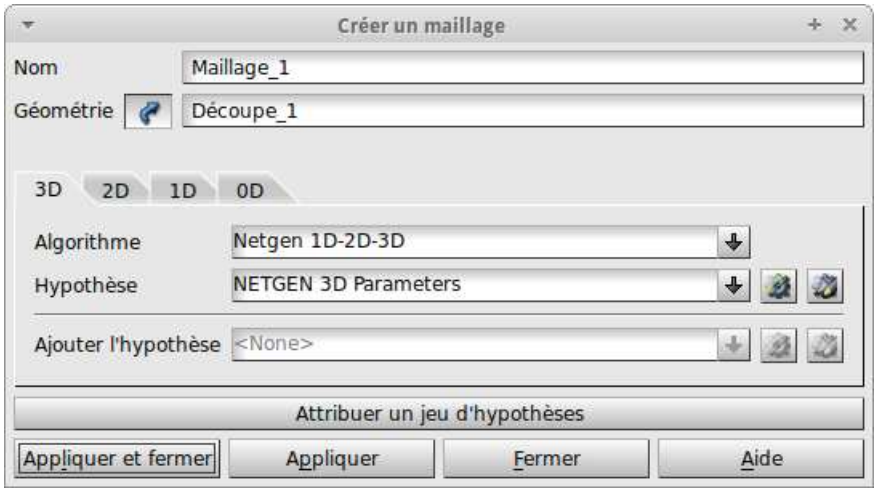



– Appuyez sur *Appliquer et fermer,* on obtient l'arborescence :

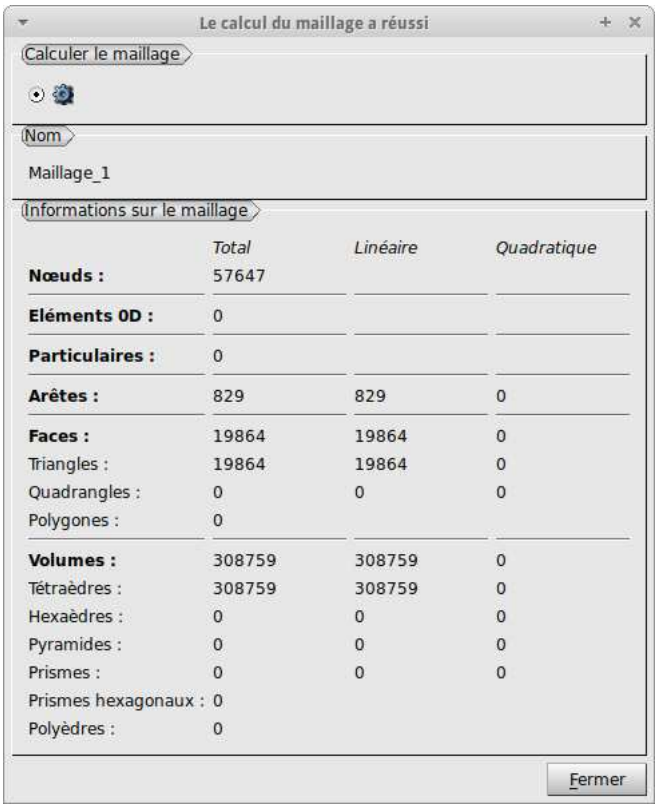



## A.2 Maillage

- Appuyez sur le bouton 
- Cette commande ouvre le dialogue

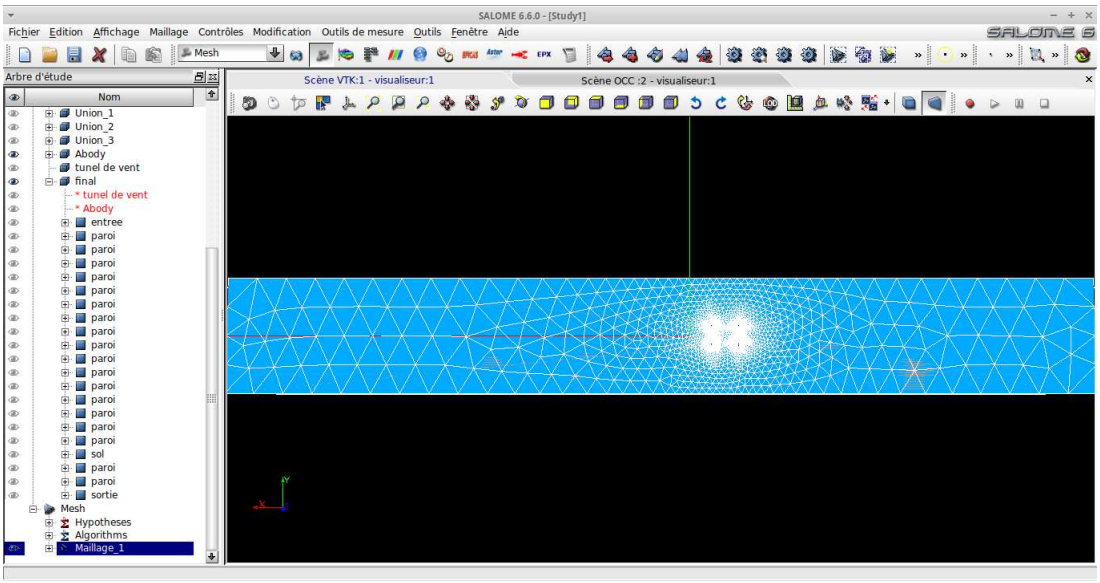



- On choisis la méthode de maillage dans *Algorithme* puis on règle les paramètres du maillage avec le bouton à droite de *Hypothèse* et enfin sur *Appliquer et fermer*.
- On appuis sur le bouton  pour lancer l'exécution du maillage ;



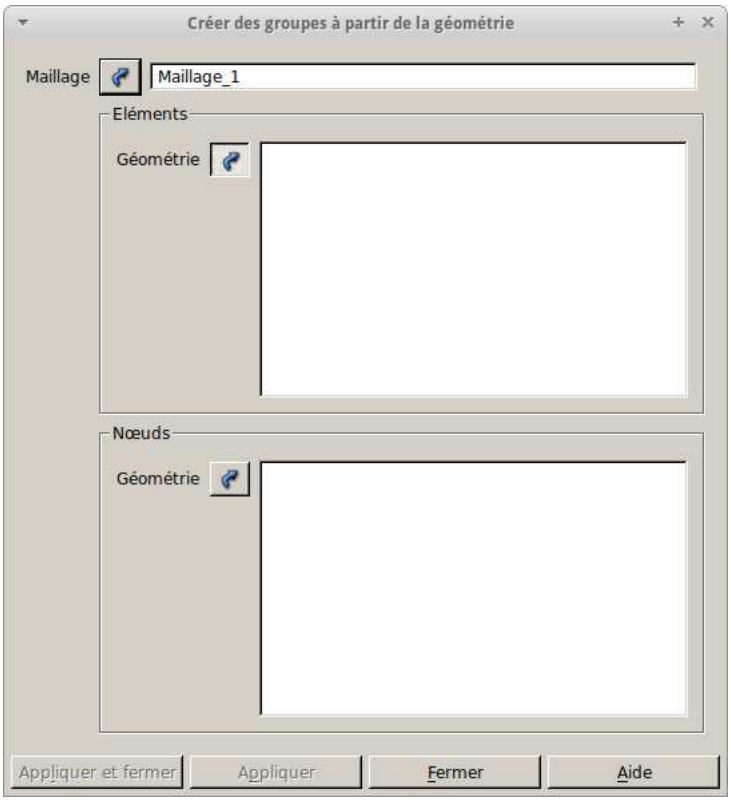

| | Total | Linéaire | Quadratique |
|---|---|---|---|
| **Nœuds :** | 57647 | | |
| **Eléments 0D :** | 0 | | |
| **Particulaires :** | 0 | | |
| **Arêtes :** | 829 | 829 | 0 |
| **Faces :** | 19864 | 19864 | 0 |
| Triangles : | 19864 | 19864 | 0 |
| Quadrangles : | 0 | 0 | 0 |
| Polygones : | 0 | | |
| **Volumes :** | 308759 | 308759 | 0 |
| Tétraèdres : | 308759 | 308759 | 0 |
| Hexaèdres : | 0 | 0 | 0 |
| Pyramides : | 0 | 0 | 0 |
| Prismes : | 0 | 0 | 0 |
| Prismes hexagonaux : | 0 | | |
| Polyèdres : | 0 | | |

– On obtien :

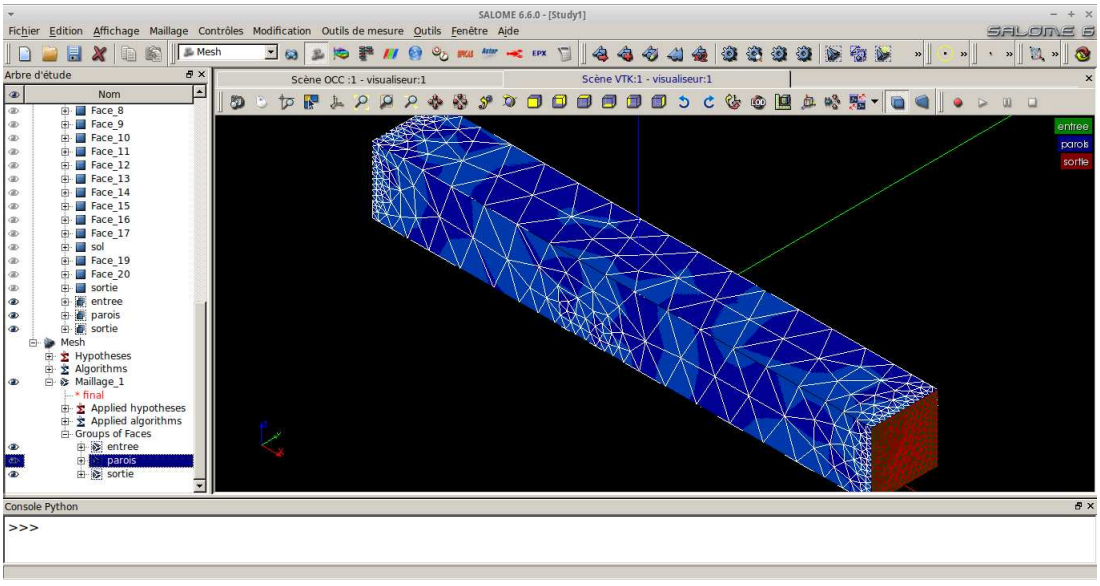



# A.3 Conditions aux limites

Avant de faire entrer les (C.L), on doit d'abord grouper les faces **entrée**, **sortie** et **parois** situés sur l'arborescence à gauche ;



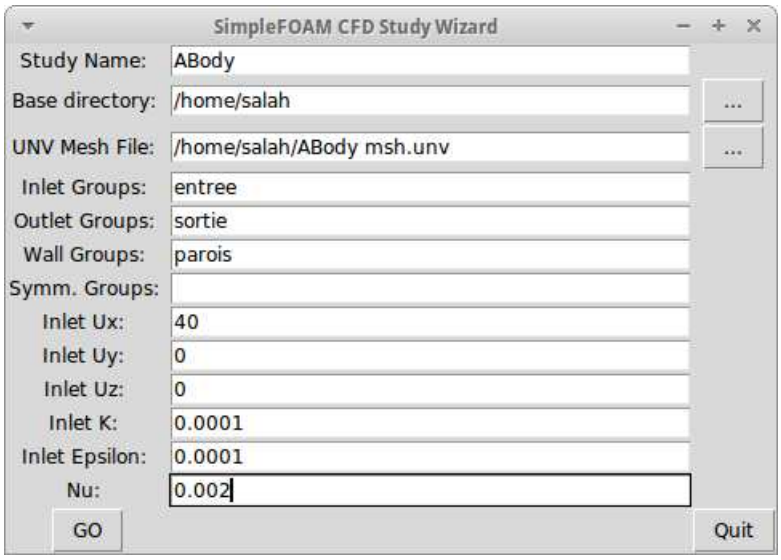

– A la fin de cette opération on obtient ;



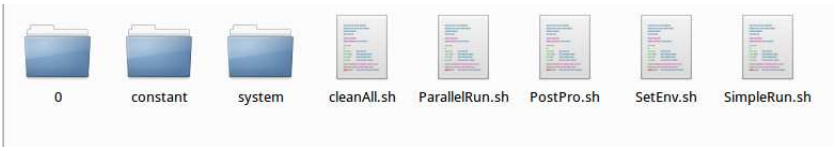

– Finalement, on exporte le maillage au format UNV en cliquant sur *maillage* à gauche puis avec le bouton droit de la souris.

## A.4 Simulation par OpenFOAM Wizard

– Ouvrir *OpenFOAM Wizard* qui est disponible dans le menu de *CAELinux* puis entrez les données en faisant attention aux (C.L) qui doivent être écrite de la même manière que dans *Salome Mesh*.




– Dans le répertoire de base on trouvera alors de nouveaux répertoires et fichiers ;




– Lancer les calculs par la commande **SimpleRun.sh** en utilisant un terminal disponible dans le menu *Accessoires*.

# ANNEXE B

# DICTIONNAIRE DE MAILLAGE *SNAPPYHEXMESHDIC*

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM Extend Project: Open Source CFD        |
|  \\    /   O peration     | Version:  1.6-ext                               |
|   \\  /    A nd           | Web:      www.extend-project.de                 |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      snappyHexMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
castellatedMesh true;
snap            true;
addLayers       true;
//=====================
geometry
//=====================
{
     ABody.stl
    {
        type triSurfaceMesh;
        name ABody;
    }
     Roues.stl
    {
        type triSurfaceMesh;
        name Roues;
    }

   ABody_refinementBox
   { type searchableBox;
```

```
      min (-1.560 -0.25 0.0); max (0.516 0.25 0.5);
    }
    Ground_refinement
    { type searchableBox;
      min (-1.560 -0.25 0.0); max (0.516 0.25 0.05);
    }
    Wake_refinement1
    { type searchableBox;
      min (0.516 -0.3 0.0); max (2.0 0.3 0.5);
    }
    Wake_refinement2
    { type searchableBox;
      min (2.0 -0.3 0.0); max (3.508 0.3 0.4);
    }
};
//====================
castellatedMeshControls
//====================
{
     maxLocalCells 1000000;
     maxGlobalCells 2000000;
     minRefinementCells 10;
     maxLoadUnbalance 0.10;
     nCellsBetweenLevels 5;

     features
     (
     );
     refinementSurfaces
     {
         ABody
         {
              level (5 6);
         }
         Roues
        {
              level (6 7);
         }
         ground
       {
             level (3 4);
       }
     }
     resolveFeatureAngle 130;
     refinementRegions
     {
      ABody_refinementBox
         {
              mode inside;
              levels ((1E15 6));
         }
     Ground_refinement
         {
       mode inside;
            levels ((1E15 4));
     }
     Wake_refinement1
         {
       mode inside;
```

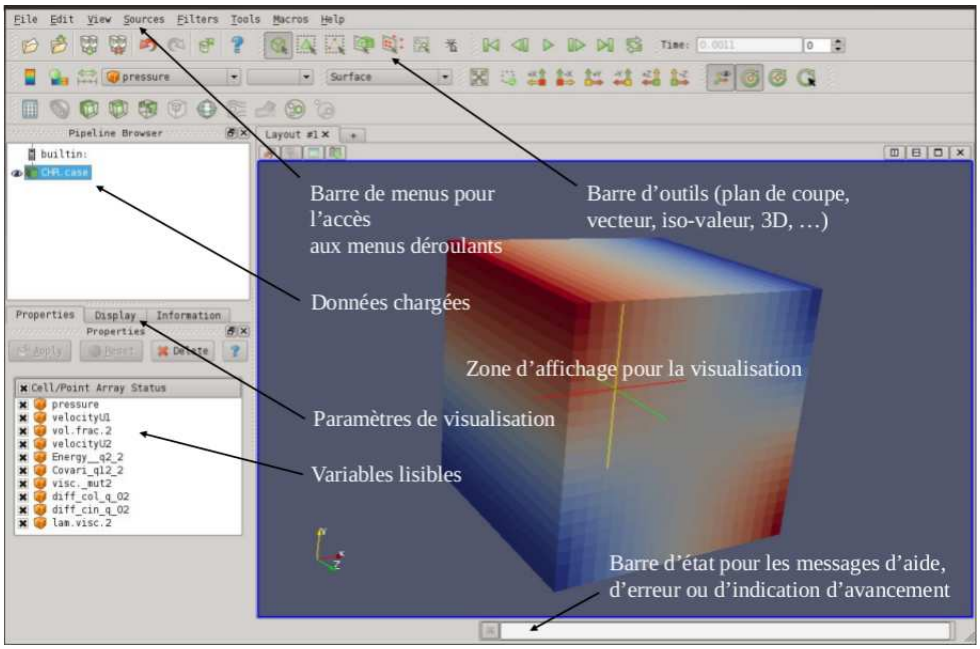
```
            levels ((1E15 4));
        }
            Wake_refinement2
            {
          mode inside;
            levels ((1E15 3));
        }
        }
        locationInMesh (3 0.0 0.8);
        allowFreeStandingZoneFaces true;
}
//====================
snapControls
//====================
{
        nSmoothPatch 3;
        tolerance 4.0;
        nSolveIter 30;
        nRelaxIter 5;
}
//====================
addLayersControls
//====================
{
        relativeSizes false;
        layers
        {
            ABody_solid
            {
                nSurfaceLayers 6;
            }
            ground
            {
                nSurfaceLayers 3;
            }
        }
        expansionRatio 1.2;
        finalLayerThickness 0.0025;
        minThickness 0.00125;
        nGrow 1;
        featureAngle 150;
        nRelaxIter 3;
        nSmoothSurfaceNormals 1;
        nSmoothNormals 3;
        nSmoothThickness 10;
        maxFaceThicknessRatio 0.5;
        maxThicknessToMedialRatio 0.3;
        minMedianAxisAngle 130;
        nBufferCellsNoExtrude 0;
        nLayerIter 50;
}
//====================
meshQualityControls
//====================
{
        maxNonOrtho 65;
        maxBoundarySkewness 20;
        maxInternalSkewness 4;
        maxConcave 80;
```

```
    minFlatness 0.5;
    minVol 1e-13;
    minTetQuality -1;
    minArea -1;
    minTwist 0.02;
    minDeterminant 0.001;
    minFaceWeight 0.02;
    minVolRatio 0.01;
    minTriangleTwist -1;
    nSmoothScale 4;
    errorReduction 0.75;
}
// Advanced
debug 0;
mergeTolerance 1E-6;
// ************************************************* //
```

# ANNEXE C

# POST-TRAITEMENT AVEC PARAFOAM

La commande *paraFoam* dans le terminal est un moyen simple de voir rapidement les résultats dans *ParaView*. Par contre, la meilleur façon d'exploiter pleinnement les résultats est d'exporter tous les résultats au format VTK ensuite d'appeler ParaView à partir du menu principal qui présenté ci-dessous.




## C.1 Paramètres de visualisation

**L'onglet Properties**

Permet de connaître les variables contenues dans un jeu de données et de choisir celles que l'on veut post-traiter.

**L'onglet Display**

Permet de paramétrer l'affichage :

- choix de la variable ;
- choix de la palette de couleurs ;
- dimensions ;
- type de représentation ;
- transparence.

**L'onglet Information**

Permet d'obtenir des informations sur :

- le type des données ;
- le nombre de cellules du maillage ;
- la taille en mémoire ;
- le nom des variables, leur type et leur min et max ;
- les dimensions du maillage ;
- les différentes sauvegardes en temps.

## C.2 Filtres

Les filtres/Menu alphabétique permettent de gérer les données de différentes façons. Certains filtres peuvent être sélectionnés par les icônes :

**Calculatrice Filtre**

• Avec le filtre de la calculatrice  vous pouvez créer de nouveaux champs scalaires ou vectoriels non existant, en utilisant des fonctions mathématiques simples.

• Les nouveaux champs peuvent être utilisés comme champs d'origine.

• Créer un nouveau champ scalaire qui est seulement la composante selon x de la vitesse : en cliquant sur Scalaires et en choisissant U_x. Spécifiez Résultat "Nom du tableau" par exemple U_x, et cliquez sur Appliquer. Un objet "calculatrice 1" apparaît dans le Navigateur de pipeline, et la géométrie est colorée par U_x. Le nouvel objet est basé sur l'objet qui a été sélectionné lors de la génération de "calculatrice 1".

• Utilisez l'icône «œil» en face des objets pour afficher / masquer chaque objet.

• Supprimer l'objet calculatrice 1 en cliquant dessus dans la PipelineBrowser puis en cliquant sur supprimer dans Inspecteur Objet / Propriétés.

**Contour, clip et Filtres Slice**

• Le filtre Contour  vous permet de faire un tracé de contour ou iso-surface, si l'objet parent est en 2D ou 3D, respectivement. Une gamme de valeur peut être

spécifiée pour plusieurs valeurs en même temps. Assurez-vous de ne pas être dans le temps 0 !

• Le filtre Clip  permet de couper vos résultats à l'aide avion / boîte / sphère / scalaire.

• Le filtre de Slice  vous permet d'effectuer avion / boîte / sphère traverse vos résultats. Il est possible de spécifier le plan de différentes manières. Plusieurs coupes peuvent être faites simultanément en utilisant Slice valeurs de décalage. Chaque tranche va générer un nouvel objet dans le navigateur pipeline, qui peut être de couleur comme tout autre objet

**Seuil, Glyph et centre de la cellule Filtres**

• Le filtre de seuil  vous permet de montrer les cellules avec des valeurs de solution scalaires dans une plage spécifique.

• Le filtre des glyphes  vous permet de faire des tracés de vecteurs. Il peut également être utilisé pour visualiser des particules lagrangiennes.

- Glyphes sont générées par défaut au niveau des points de la grille. Si vous voulez les vecteurs doivent être situés dans le centre des cellules, vous devez d'abord faire un nouvel objet des centres cellulaires en utilisant les Centres cellulaires filtre qui peut être trouvé dans les filtres / alphabétique.

- Marquez l'objet centré cellulaire, et filtrer avec les valeurs par défaut du filtre des glyphes. Il est possible de modifier la représentation des vecteurs de la propriétés tag.

**Diffuser Tracer et Warp Filtres**

• Le filtre flux Tracer  vous permet de faire des tracés rationalisés, présentés avec un point, un nuage sphérique, ou de sources en ligne.

• La chaîne Par filtre Vector  vous permet de déformer la géométrie selon la valeur locale d'un vecteur. Cela est surtout utile dans la mécanique des solides. Il ya aussi une chaîne par le filtre Scalar disponible dans Filtres / Alphabétique, où vous pouvez trouver de nombreux filtres supplémentaires.

**Plot Over filtre Line**

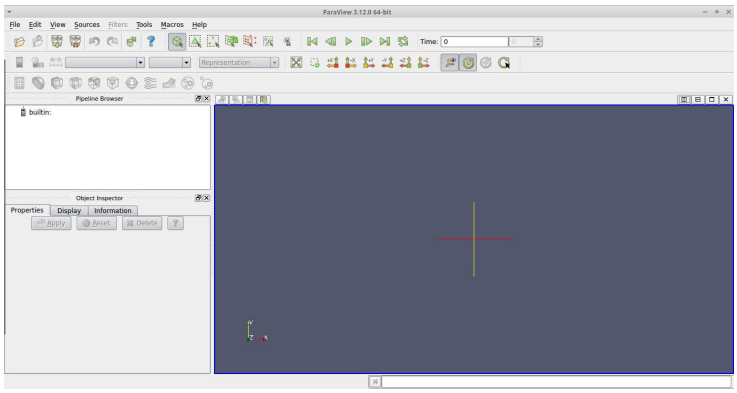
• En Filtres / alphabétique, vous trouverez une longue liste de filtres.

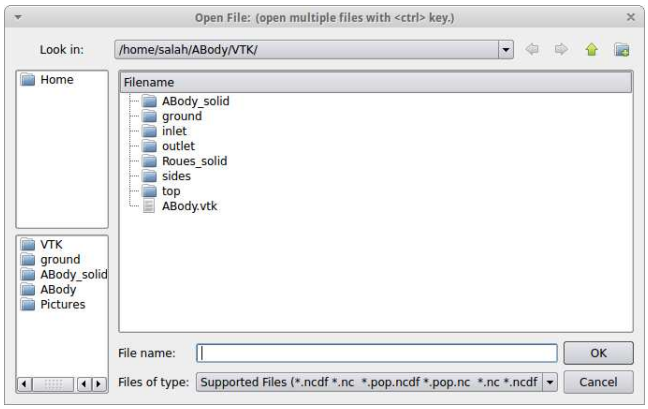
• Le Plot Over filtre line vous permet de faire des parcelles détaillées des variables le long d'une ligne.

• Dans l'onglet Affichage, vous pouvez choisir les variables à tracer, et choisissez couleurs de ligne et styles de ligne.

## C.3 Lecture d'un fichier

- Menu File, sous-menu Open
- Sélectionner le fichier à lire, puis OK
- Dans le panneau gauche, fenêtre Pipeline browser,sélectionner la ou les éléments à visualiser (**ABody_solid**, **ground** et **ABody.vtk**)
- Apply






**Changer l'affichage des données**

La première chose que fait l'utilisateur à ce stade est généralement de changer l'apparence des données affichées. Pour ce faire, il est possible d'utiliser les outils de la barre des tâches afin de changer la coloration («Solid Color» pour une couleur unie, «p» pour le champ de pression, «U» pour celui de vitesse, etc.) ou le mode de représentation («Surface» pour surfacique, «Wireframe» pour filaire, etc.) :