

## RAPPORT DE STAGE

# Implémentation d'une interface Python pour des algorithmes de remaillage

Stagiaire : Mathieu RIGAL

Superviseur : Adrien LOSEILLE

18 septembre 2018



M. Rigal, Implémentation d'une interface Python pour les algorithmes de remaillage, Stage à l'Inria, 2018

# Remerciements

Avant toute chose j'aimerais dédier ces lignes à l'ensemble de l'équipe GAMMA3 que je remercie de m'avoir chaleureusement accueilli dans un cadre de travail très agréable.

Je tiens naturellement à exprimer ma gratitude envers Adrien Loseille pour m'avoir accepté en stage et encadré pendant deux mois et demi. Son soutien et sa disponibilité ont permis le bon déroulement de ce stage.

J'aimerais également adresser mes remerciements à Julien Vanharen pour ses suggestions et son aide.

Je souhaite aussi remercier Maria-Agustina Ronco pour son support tout au long de cette période de stage.

M. Rigal, Implémentation d'une interface Python pour les algorithmes de remaillage, Stage à l'Inria, 2018

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	L'Inria . . . . .	1
1.2	L'équipe-projet GAMMA3 . . . . .	1
1.3	Objectifs du stage et plan du rapport . . . . .	2
<b>2</b>	<b>Théorie du maillage : quelques méthodes numériques pour le cas 2D</b>	<b>3</b>
2.1	Définitions . . . . .	3
2.2	Discrétisation du contour . . . . .	4
2.3	Création d'un support linéaire . . . . .	7
2.4	Méthode frontale . . . . .	10
<b>3</b>	<b>Le langage Python</b>	<b>13</b>
3.1	Spécificités, avantages et inconvénients de Python . . . . .	13
3.2	L'API C/Python . . . . .	13
<b>4</b>	<b>Contribution</b>	<b>15</b>
4.1	La visualisation de maillages avec Vizir . . . . .	15
4.1.1	Eléments et types de maillages . . . . .	15
4.1.2	Plan de coupe et mini-mesh . . . . .	18
4.1.3	Système de coordonnées homogènes . . . . .	19
4.1.4	Le module pyviz . . . . .	20
4.2	Le remaillage avec feflo.a . . . . .	23
4.2.1	L'interface amgio pour gérer le format SU2 . . . . .	23
4.2.2	Le module pyamg . . . . .	23
<b>5</b>	<b>Conclusion</b>	<b>25</b>
5.1	Apport du stage . . . . .	25
5.2	Perspectives . . . . .	25
	<b>Bibliographie</b>	<b>27</b>

M. Rigal, Implémentation d'une interface Python pour les algorithmes de remaillage, Stage à l'Inria, 2018

# Chapitre 1

## Introduction

### 1.1 L’Inria

L’Institut National de Recherche en Informatique et en Automatique (Inria) est un établissement public français de recherche dans les domaines des sciences et technologies, et plus particulièrement des Mathématiques et de l’Informatique. Créé en 1967 dans le cadre du plan Calcul par le président Charles de Gaulle, l’IRIA — ancien acronyme de l’Inria — avait pour but premier d’affirmer l’autonomie française dans les technologies de l’information et d’étendre cet objectif à l’échelle Européenne.

Aujourd’hui sous la tutelle des ministères de l’Enseignement supérieur, de la Recherche et de l’Innovation, ainsi que de l’Économie et des Finances, l’ambition de l’Inria est désormais tournée vers « l’excellence scientifique au service du transfert technologique et de la société ». Ceci est concrétisé par la collaboration de 2 400 chercheurs et ingénieurs de 101 nationalités différentes en interaction directe avec des partenaires industriels et académiques.

L’Inria s’organise autour d’un modèle de 184 « équipes-projets » rassemblant des chercheurs aux compétences complémentaires autour d’un même projet scientifique, et répartis dans 8 centres de recherche à travers la France (Paris, Rennes, Sophia Antipolis, Grenoble, Nancy, Bordeaux, Lille et Saclay). Cette organisation permet notamment la publication de 4 400 publications par an. D’autre part, 112 logiciels étaient déposés par l’Inria en 2017, et 57 start-up Inria ont été créées depuis 2010. Le budget total s’élevait à 231 millions d’euros pour l’année 2017.

### 1.2 L’équipe-projet GAMMA3

Ce stage s’est déroulé au sein de l’équipe-projet GAMMA3 au centre Inria de Saclay (bâtiment Alan Turing, campus de l’école Polytechnique, Palaiseau). Crée en 1996, cette équipe a pour objectif d’étudier et de développer les algorithmes de génération automatique de maillages sollicités

à travers des méthodes numériques de type éléments finis, volumes finis ou encore différences finies. L'enjeu principal est l'obtention d'une boucle entièrement automatique de calculs approchant la solution du problème considéré avec une précision souhaitée. Pour parvenir à ces fins, de nombreux axes de recherches sont investis.

**Algorithmes de génération de maillages.**

- Méthode de Delaunay, méthode frontale, autres méthodes (hexaèdres) ;
- Maillage volumique, maillage surfacique, maillage plan ;
- Maillage gouverné (maillage euclidien, maillage riemannien) ;

**Estimateurs d'erreurs.** Estimateur a posteriori (*utilisation du Hessien et autres méthodes*).

**Calcul parallèle.** Partitionnement automatique a posteriori et a priori.

**Structures de données.** Lien CAO-calcul, définition géométrique des domaines, structures de données de maillages.

**Environnement de calculs.**

- Interface graphique ;
- Logiciels ;
- Analyse des résultats ;

Il est par ailleurs important de mentionner les quelques autres domaines d'applications compatibles avec les solutions proposées par l'équipe GAMMA3, à savoir la compression d'images et la saisie scanner/impression 3D.

### 1.3 Objectifs du stage et plan du rapport

Ce stage a pour objectif de proposer une librairie Python d'interfaçage des algorithmes de visualisation et de remaillage contenus dans la suite de logiciels GAMMA3. Une telle interface permettrait en effet :

- d'accroître la flexibilité des outils GAMMA3 ;
- de prototyper rapidement de nouveaux algorithmes ;
- de diffuser ces outils GAMMA3 à une audience élargie, notamment à des fins académiques ;

Ce rapport a pour vocation de présenter le travail effectué au cours de ce stage. Le chapitre 2 explicite quelques méthodes numériques permettant le maillage d'une surface plane à l'aide de triangles, tandis que le chapitre 3 évoque les spécificités du langage Python et l'*Application Programming Interface* (API) C/Python utilisée tout le long du stage. Enfin, le chapitre 4 présente les résultats ainsi que l'apport de ce stage au sein du projet GAMMA3.

## Chapitre 2

# Théorie du maillage : quelques méthodes numériques pour le cas 2D

Différentes méthodes de simulation numérique, telles que les éléments finis, les volumes finis ou encore les différences finies, requièrent une décomposition discrète de l'espace. C'est de cette notion de décomposition, connue sous le nom de *maillage*, dont il est question ici. On s'attachera à l'étude du cas simplifié des géométries 2D afin de présenter quelques méthodes d'intérêt actuellement utilisées ainsi que les différents enjeux. Il est important de garder à l'esprit que la théorie du maillage est un domaine très vaste, par conséquent ce chapitre n'essaie en aucun cas d'en présenter une description exhaustive, mais plutôt un très bref aperçu introductif.

### 2.1 Définitions

Avant d'aller plus en avant, il est important de définir quelques notions relatives au maillage.

**Définition** (Triangulation).

On appelle triangulation  $\mathcal{T}$  d'un domaine polygonal l'ensemble des triangles  $(T_i)$  vérifiant les propriétés suivantes :

1.  $\forall i$ , l'intérieur de  $T_i$  (noté  $\mathring{T}_i$ ) est non vide ;
2.  $\forall i, j$  tels que  $i \neq j$ ,  $\mathring{T}_i \cap \mathring{T}_j = \emptyset$  ;
3. toute arête d'un triangle est soit l'arête d'un autre triangle soit une arête du bord ;

Pour un domaine polygonal quelconque, on peut facilement montrer l'existence d'une triangulation (voir [1]).

On considère ici un domaine 2D noté  $\Omega$ , de frontière  $\Gamma$ . La génération automatique d'un maillage de  $\Omega$  suit généralement deux étapes : la discréétisation  $M(\Gamma)$  de la frontière  $\Gamma$  du domaine, et la construction du maillage intérieur  $M(\Omega)$  tenant compte de  $M(\Gamma)$ .

Il est dès lors souhaitable de déterminer quels critères permettent d'affirmer si un maillage est de meilleure qualité ou non qu'un autre. On introduit pour cela les termes de *finesse* et de *régularité* d'un maillage.

**Définition (Finesse).**

Soit  $K$  un élément du maillage. On nomme diamètre de  $K$  la quantité :

$$h_K = \max_{P,Q \in K} \|P - Q\|$$

On définit alors la finesse  $h$  du maillage comme :

$$h = \max_{K \in M(\Omega)} h_K$$

**Définition (Régularité).**

On appelle régularité d'un élément  $K$  la grandeur

$$\sigma_K = \frac{h_K}{\rho_K}$$

avec  $\rho_K > 0$  dépendant de la transformation  $T_K$  d'un élément de référence  $\hat{K}$  vers  $K$ . De façon générale, si  $K$  est un polygone constitué de  $n \geq 3$  sommets, on peut poser  $\rho_K = \min_{i \in [1,n]} \rho_i$  avec  $\rho_i$  le rayon du cercle inscrit dans le triangle contenant les deux arêtes issues du sommet  $i$  de  $K$ .

On définit alors la régularité du maillage par :

$$\sigma = \max_{K \in M(\Omega)} \sigma_K$$

On dira d'un maillage  $M_1(\Omega)$  qu'il est de meilleure qualité que le maillage  $M_2(\Omega)$  si sa finesse et sa régularité sont inférieures à celles de  $M_2(\Omega)$ . À titre d'exemple, pour une finesse donnée un maillage triangulaire aura une meilleure qualité si ses triangles sont équilatéraux ou quasi équilatéraux, et une moins bonne qualité si ses triangles sont aplatis. En effet, pour un triangle donné le rayon du cercle inscrit est plus élevé si ce triangle est équilatéral plutôt qu'aplatit.

## 2.2 Discréétisation du contour

Différentes méthodes permettent la discréétisation d'une courbe définie au moyen d'une paramétrisation  $t \mapsto \gamma(t) \in \mathbf{R}^2$  pour  $t$  dans  $[t_0, t_f]$ . En notant

## 2.2 Discrétisation du contour

$s(t)$  l'abscisse curviligne définit par :

$$s(t) = \int_{t_0}^t \|\dot{\gamma}(\tau)\|_2 d\tau \quad \forall t \in [t_0, t_f] \quad (2.1)$$

ces méthodes reviennent à extraire une famille croissante de paramètres  $(t_i)_{1 \leq i \leq n}$ , avec  $n$  le nombre de points souhaité.

En outre, il existe plusieurs catégories de discrétisation :

- discrétisation uniforme (les points sont répartis de façon équidistante le long du contour) ;
- discrétisation isotrope (les points sont séparés par une distance qui varie en fonction de la position) ;
- discrétisation anisotrope (les points sont séparés par une distance qui varie en fonction de la position et de l'orientation) ;

**Cas uniforme :** La méthode de discrétisation en  $n$  points suit les étapes suivantes :

1. Calculer la longueur totale  $L = s(t_f) = \int_{t_0}^{t_f} \|\dot{\gamma}(t)\|_2 dt$  ;
2. Déterminer les paramètres  $t_i$  tels que  $s(t_i) = \int_{t_0}^{t_i} \|\dot{\gamma}(t)\|_2 dt = i \frac{L}{n}$  ;

La première étape est résolue numériquement par une méthode de quadrature, tandis que la seconde étape revient à résoudre l'équation non linéaire  $f(r) = 0$  avec  $f(r) = s(r) - r \frac{L}{n}$ . Une solution de cette équation peut être approchée par une méthode de point fixe par exemple.

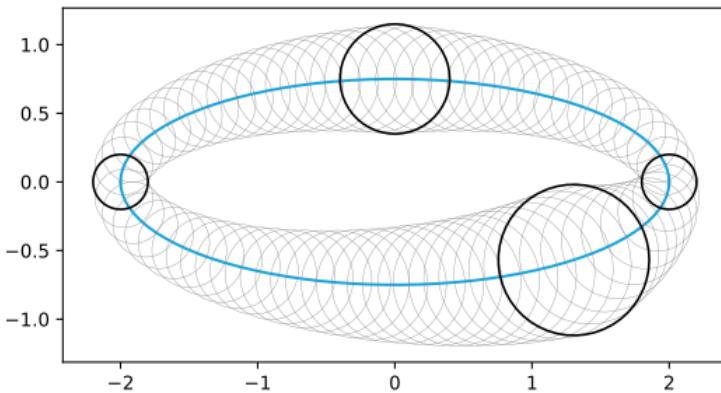


FIGURE 2.1 – Interpolation linéaire des contraintes isotropes le long d'une ellipse (contour bleu). Les seules contraintes définies sont représentées par les cercles noirs. Les cercles gris représentent les contraintes estimées par interpolation linéaire.

**Cas isotrope :** On introduit une fonction de taille  $h(P)$  donnant la longueur idéale de toute arrête issue de  $P$ , de telle sorte que tout point relié à  $P$  doit

nécessairement être contenu dans le disque  $\mathcal{D}(P, h(P))$ . En réalité, la fonction  $h$  n'est connue qu'en un nombre fini de points  $(P_i)_i$  associés aux abscisses curvilignes  $(s_i)_i$ . Il est toutefois possible d'en obtenir une description continue par interpolation linéaire ou géométrique entre deux points du contour (voir fig. 2.1) :

$$\begin{cases} h_{l,i}(s) = h(P_i) + r_i(s) (h(P_{i+1}) - h(P_i)) \\ h_{g,i}(s) = h(P_i) \left( \frac{h(P_{i+1})}{h(P_i)} \right)^{r_i(s)} \end{cases} \quad \text{avec } r_i(s) = \frac{s - s_i}{s_{i+1} - s_i}$$

On définit ensuite la longueur normalisée  $l_h(P, Q)$  séparant deux points  $P$  et  $Q$  :

$$l_h(P, Q) = \int_{s_P}^{s_Q} \frac{1}{h^*(s)} ds, \quad s_P \text{ et } s_Q \text{ abscisses curvilignes de } P \text{ et } Q \quad (2.2)$$

Dans la définition (2.2), la fonction  $h^*$  est par exemple définie à l'aide de l'interpolation linéaire  $h_l$  :

$$h^*(s) = \sum_{i=1}^n \mathbf{1}_{[s_i, s_{i+1}[}(s) h_{l,i}(s) \quad \forall s \in [s(t_0), s(t_f)] \quad (2.3)$$

La discréttisation isotrope du contour peut alors être formulée dans les termes suivants : décomposer  $\Gamma$  en arcs de longueurs normalisées aussi proches que possible de 1. Dès lors, on a l'algorithme suivant :

1. Calculer la longueur totale  $L = s(t_f)$  et déterminer  $n$  tel que  $\frac{L}{n}$  est aussi proche que possible de 1 ;
2. Déterminer les paramètres  $(t_i)_{0 \leq i \leq n}$  tels que  $l_h(\gamma(t_0), \gamma(t_i)) = i \frac{L}{n}$  ;

Les mêmes méthodes numériques que pour le cas uniforme peuvent être employées ici.

**Cas anisotrope :** On introduit un tenseur métrique  $\mathcal{M}$  qui à un point  $P$  associe une matrice symétrique définie positive de taille 2. Tout comme dans le cas isotrope, on introduit une longueur normalisée associée à  $\mathcal{M}$  :

$$l_{\mathcal{M}}(\gamma(t_1), \gamma(t_2)) = \int_{t_1}^{t_2} \sqrt{\dot{\gamma}(t)^T \mathcal{M}(\gamma(t)) \dot{\gamma}(t)} dt \quad (2.4)$$

Pour tout point  $Q$  relié à  $P$ , on impose la contrainte d'anisotropie  $l_{\mathcal{M}}(P, Q) = 1$ . Une interprétation géométrique est donnée dans le cas simplifié où la métrique  $\mathcal{M}$  est constante :

$$\mathcal{M} = \begin{pmatrix} a & b \\ b & c \end{pmatrix}, \quad a > 0, \quad c > 0, \quad ac - b^2 > 0$$

## 2.3 Crédation d'un support linéaire

Dans le repère centré en  $P$ , on a  $(x_P \ y_P) = (0 \ 0)$  et :

$$\begin{aligned} l_{\mathcal{M}}(P, Q)^2 = 1 &\iff \begin{pmatrix} x_Q & y_Q \end{pmatrix} \begin{pmatrix} a & b \\ b & c \end{pmatrix} \begin{pmatrix} x_Q \\ y_Q \end{pmatrix} = 1 \\ &\iff ax_Q^2 + 2bx_Qy_Q + cy^2 = 1 \end{aligned}$$

Ainsi, le point  $Q$  appartient idéalement à une ellipse centrée en  $P$  et définie par les coefficients de  $\mathcal{M}$ . L'algorithme de discréétisation de  $\Gamma$  dans le cas anisotrope est le même que pour le cas isotrope, en remplaçant  $l_h$  par  $l_{\mathcal{M}}$ .

## 2.3 Crédation d'un support linéaire

Toutes les méthodes numériques présentées ci-dessus ont le défaut d'être coûteuses en temps de calcul, d'autant que la discréétisation de  $\Gamma$  peut requérir un nombre relativement élevé de points. Pour cette raison, l'équipe GAMMA3 a développé une méthode alternative plus efficace basée sur la création d'un support linéaire par morceaux comme expliqué dans [2]. Ce support, noté par la suite  $S(\Gamma)$ , est une ligne polygonale ayant pour but d'approximer au mieux le contour  $\Gamma$  tout en utilisant un minimum de points. L'aspect linéaire par morceaux de  $S(\Gamma)$  pourra être utilisé à bon escient afin d'accélérer les méthodes de quadrature et de point fixe mentionnées jusqu'ici, avec un impact négligeable sur la fidélité du maillage.

L'idée principale est d'exhiber une famille croissante de paramètres  $(t_i)_i$  vérifiant la propriété suivante :

### Propriété (Linéarité du support).

*Lorsque  $t$  varie de  $t_i$  à  $t_{i+1}$ , le point  $\gamma(t)$  de  $\Gamma$  peut être approché, grâce à une interpolation, par  $\gamma'(t) = \gamma(t_i) + r_i(t) (\gamma(t_{i+1}) - \gamma(t_i))$  où  $r_i(t) = \frac{t-t_i}{t_{i+1}-t_i}$ .*

L'algorithme 1 permet de construire une telle suite  $(t_i)_i$  de manière récursive tout en majorant par  $\varepsilon > 0$  l'écart entre  $S(\Gamma)$  et  $\Gamma$ . Dans un premier temps, l'idée est de fournir un découpage préliminaire de  $\Gamma$  en  $N$  portions  $(\Gamma_n)_{1 \leq n \leq N}$ . Pour un  $\Gamma_i$  donné, on considère le support  $S_0^i(\Gamma_i)$  défini comme le segment reliant les deux extrémités de  $\Gamma_i$ . On cherche alors le paramètre  $t_{\max}$  maximisant la distance  $\text{dist}(\gamma(t), \gamma'(t))$ . Si cette distance est acceptable,  $S_0^i(\Gamma_i)$  est une assez bonne approximation de  $\Gamma_i$ . Dans le cas contraire, on introduit un nouveau support  $S_1^i(\Gamma_i)$  obtenu en insérant le point  $\gamma(t_{\max})$  dans  $S_0^i(\Gamma_i)$ . On itère alors de manière récursive de par et d'autre de ce point jusqu'à l'obtention d'un support  $S_{k(i)}^i(\Gamma_i)$  acceptable. En effectuant cette opération sur chacun des  $(\Gamma_n)_{1 \leq n \leq N}$ , on prend finalement  $S(\Gamma) = \bigcup_{1 \leq n \leq N} S_{k(n)}^n(\Gamma_n)$ .

---

**Algorithm 1** Search of points to be inserted

---

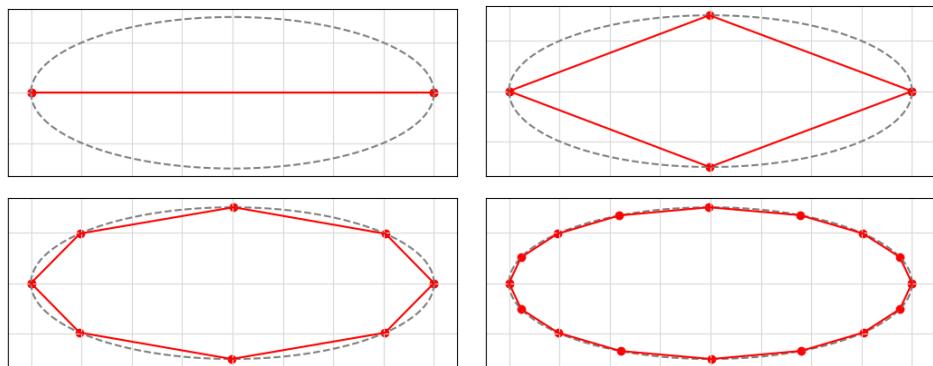
```

1 : function SUBDIVIDE( $I, p$ )
2 : Input :  $I = [t_0, t_f]$  is the interval that we want to subdivide,  $p$  is the
   number of sampling points that will be used to approximate the maximum
   distance between  $\gamma$  and  $\gamma'$  on  $I$ 
3 : Output : linear support  $S(\Gamma)$ 
4 :    $d_{\max} \leftarrow \text{dist}(\gamma(t_0), \gamma'(t_0))$ 
5 :    $t_{\max} \leftarrow t_0$ 
6 :   for  $i \leftarrow 1$  to  $p$  do
7 :      $t \leftarrow t_0 + r(i)(t_f - t_0)$ 
8 :      $d \leftarrow \text{dist}(\gamma(t), \gamma'(t))$ 
9 :     if  $d > d_{\max}$  then
10:       $d_{\max} \leftarrow d$ 
11:       $t_{\max} \leftarrow t$ 
12:    end if
13:   end for
14:   if  $d_{\max} > \varepsilon$  then
15:     add  $\gamma(t_{\max})$  to the final support  $S(\Gamma)$ 
16:     SUBDIVIDE( $t_0, t_{\max}$ )
17:     SUBDIVIDE( $t_{\max}, t_f$ )
18:   end if
19: end function

```

---

On notera que cet algorithme présente l'avantage d'être facile à paralléliser. Un exemple de support linéaire obtenu est montré en figure 2.1. On considère dans ce cas l'ellipse d'équation  $(1/2 x)^2 + (4/3 y)^2 = 1$ , dont la courbe est tracée en pointillés. Les segments rouges constituent le support. On remarque que localement aux zones caractérisées par un faible rayon de courbure, la densité des points est plus élevée, ce qui permet de mieux décrire les variations de  $t \mapsto \gamma(t)$ .



### 2.3 Création d'un support linéaire

9

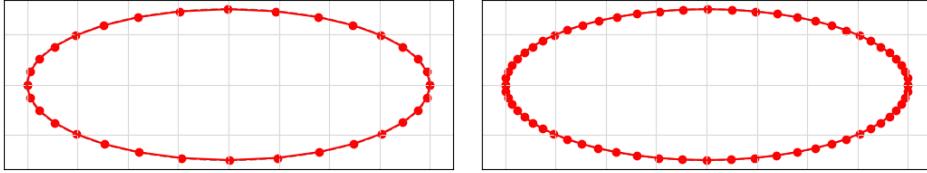


FIGURE 2.1 – Support linéaire d'une ellipse

Une fois le support linéaire obtenu, il est très aisément de le mailler de manière uniforme. L'algorithme ci-après permet d'y parvenir. La figure 2.2 montre le résultat obtenu dans le cas de l'ellipse.

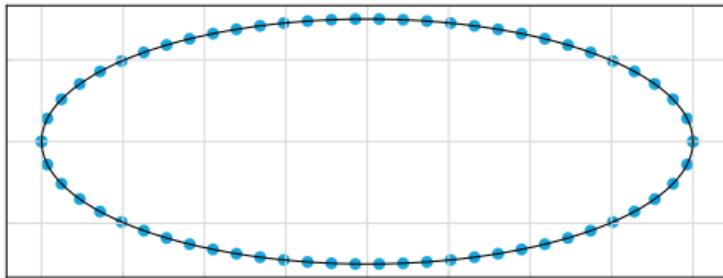


FIGURE 2.2 – Maillage uniforme généré à l'aide des algorithmes 1 et 2. Le trait noir représente le contour exact de l'ellipse. L'erreur maximale tolérée pour la création du support est de  $10^{-6}$  avec 198 points d'échantillonnage pour le calcul de l'erreur.

---

#### Algorithm 2 Generation of an uniform mesh for the support $S(\Gamma)$

---

```

1 : function UNIFORM_MESH( $S(\Gamma)$ ,  $h_{\max}$ )
2 : Input :  $S(\Gamma)$  is the linear support containing the successive segments
    $S_0, \dots, S_N$  with  $S_i$  defined by its two extreme points  $(s_i^1, s_i^2)$ ,  $h_{\max} > 0$  is
   the maximum distance between two consecutive points of the mesh
3 : Output : list  $\mathcal{P}$  of successive points defining the mesh
4 :      $L \leftarrow$  length of  $S(\Gamma)$ ,  $h \leftarrow L \times \left[ \frac{L}{h_{\max}} \right]^{-1}$ 
5 :     Add the point  $s_0^1$  to the list  $\mathcal{P}$  of mesh points
6 :      $d \leftarrow h$ ,  $P_{\text{current}} \leftarrow s_0^1$ 
7 :     for  $i \leftarrow 0$  to  $N$  do
8 :         while  $\text{dist}(P_{\text{current}}, s_i^2) \geq d$  do
9 :              $P_{\text{current}} \leftarrow P_{\text{current}} + d \times (s_i^2 - s_i^1) (\|s_i^2 - s_i^1\|_2)^{-1}$ 
10 :            Add  $P_{\text{current}}$  to  $\mathcal{P}$ 
11 :             $d \leftarrow h$ 

```

---

---

```

12 :     end while
13 :      $d \leftarrow d - \|s_i^2 - P_{\text{current}}\|_2$ 
14 :      $P_{\text{current}} \leftarrow s_i^2$ 
15 :   end for
16 :   return  $\mathcal{P}$ 
17 : end function

```

---

## 2.4 Méthode frontale

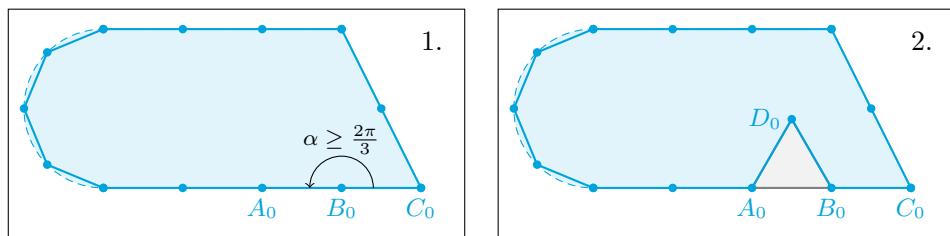
Il existe trois grandes classes d'algorithmes de maillage : quadtree/octree, la triangulation de Delaunay et la méthode frontale (voir [1] [3] pour plus de détails sur ces méthodes). Nous discutons ici de cette dernière méthode, qui nécessite comme prérequis de disposer d'un maillage du contour du domaine, ce dont il était justement question dans les deux sections précédentes.

L'approche retenue est la suivante : on considère le front initial définit comme le maillage du bord (noeuds et arêtes). Pour chaque couple d'arêtes consécutives  $[AB]$ ,  $[BC]$  de ce front, on considère l'angle  $\alpha$  les séparant :

- si  $\alpha < \frac{\pi}{2}$  les deux segments appartiendront au même triangle  $ABC$  ;
- si  $\frac{\pi}{2} \leq \alpha < \frac{2\pi}{3}$ , un point  $D$  sera inséré à l'intérieur du domaine de sorte à former les triangles  $ABD$  et  $BCD$  ;
- si  $\frac{2\pi}{3} \leq \alpha$ , un seul point  $D$  sera inséré à l'intérieur du domaine de sorte à former le triangle  $ABD$  ;

Dans ces deux derniers cas, on vérifiera si le point  $D$  possède un voisin contenu dans un rayon  $r$  prédéfini. Le cas échéant,  $D$  n'est pas inséré et est remplacé par le voisin maximisant la qualité du triangle formé. Une fois qu'un ou deux triangles ont été formés, le front est mis à jour en ajoutant les arêtes nouvellement créées, et en retirant les arêtes  $[AB]$  et  $[BC]$  (sauf si  $\frac{2\pi}{3} \leq \alpha$ , dans ce cas seule l'arête  $[AB]$  est enlevée). Une autre nécessité est de vérifier que l'insertion d'un nouveau point n'engendre pas d'intersection avec le front courant. Dans le cas contraire, le nouveau point n'est pas inséré, et est remplacé par un point voisin maximisant la qualité du ou des triangles formés.

Ce procédé est itéré jusqu'à ce que le front soit vide : l'ensemble des arêtes qui ont été retirées forment alors le maillage du domaine (voir fig. 2.2).



## 2.4 Méthode frontale

11

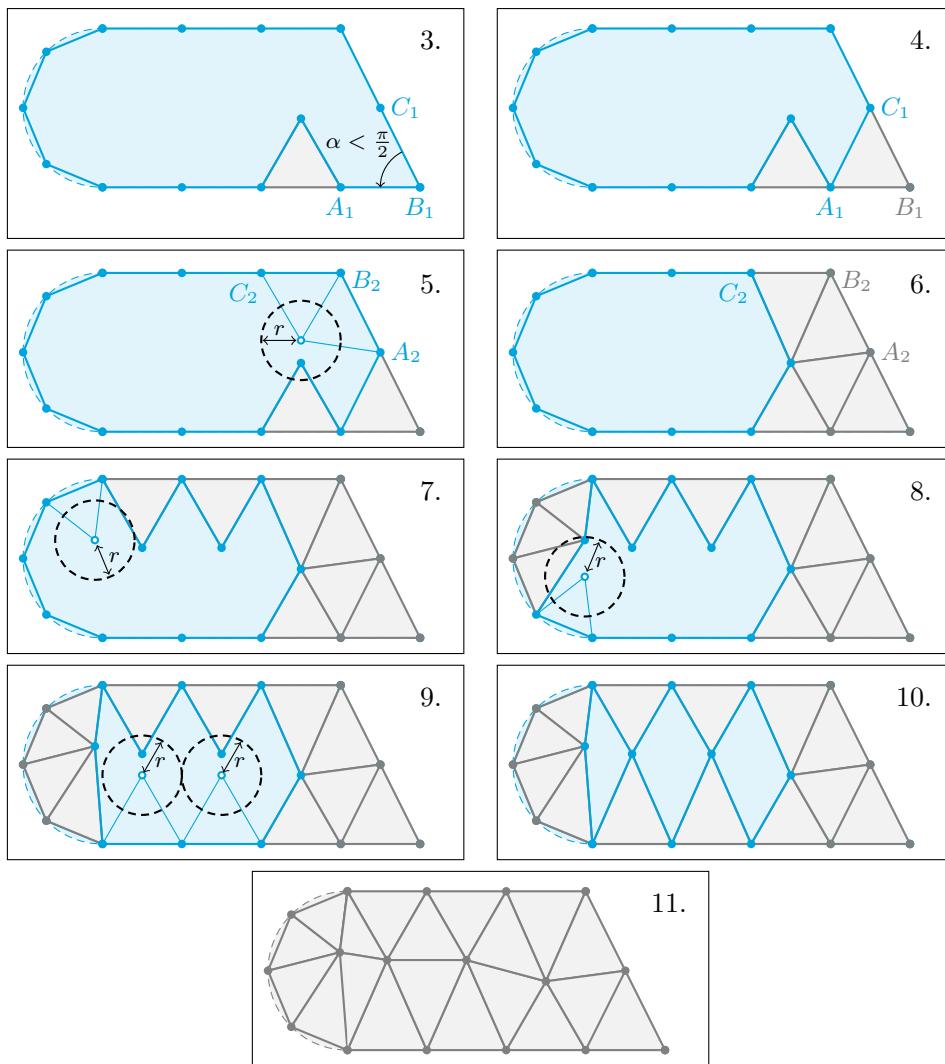


FIGURE 2.2 – Illustration de l'avancée de front

M. Rigal, Implémentation d'une interface Python pour les algorithmes de remaillage, Stage à l'Inria, 2018

## Chapitre 3

# Le langage Python

### 3.1 Spécificités, avantages et inconvénients de Python

### 3.2 L'API C/Python

L'objectif principal de ce stage était d'implémenter une librairie Python faisant appel à des codes déjà développés en C/C++. Pour cela, l'API C/Python a été utilisée afin d'obtenir un programme C/C++ interagissant directement avec l'interpréteur Python. C'est de cette manière qu'a été développée la librairie numpy parmi de nombreux exemples.

L'utilité de cette méthode est de pouvoir bénéficier de performances optimales en implémentant les parties critiques de la librairie en C/C++. Les parties moins critiques et pénibles à coder peuvent être écrites en Python. L'utilisateur de la librairie bénéficie alors du "meilleur des deux mondes" : la simplicité de faire un script Python et une vitesse proche de celle d'un programme C/C++.

M. Rigal, Implémentation d'une interface Python pour les algorithmes de remaillage, Stage à l'Inria, 2018

# Chapitre 4

# Contribution

Le travail fourni au cours de stage se base sur des codes déjà existants et relativement conséquents. Il s'agit par ailleurs d'un des points techniques rencontrés : la compréhension et l'appropriation d'un code, puis l'ajout de nouvelles fonctionnalités sans altérer le bon fonctionnement du programme. Avant de détailler ma contribution à ces codes, une description de ceux-ci sera faite et quelques détails d'implémentation intéressants seront donnés.

## 4.1 La visualisation de maillages avec **Vizir**

**Vizir** est un logiciel de visualisation et de modification de maillage. Il s'agit du point de convergence de toutes les librairies dynamiques Inria (générateur de maillage 3D, remailleur 3D, solveurs...) en facilitant leur interaction depuis le traitement de la géométrie jusqu'à la résolution numérique du problème. Pour atteindre cet objectif, **Vizir** offre les fonctionnalités suivantes :

- interfaçage avec plusieurs librairies Inria ;
- visualisation de maillage courbes (éléments d'ordres élevés, superposition de la solution) ;
- visualisation de maillages hybrides (constitués de différents types d'éléments : éléments 2D et 3D) ;
- inspection de l'intérieur du maillage par création de plans de coupe, gestion de la transparence, réduction de la taille des éléments, affichage d'une zone partielle du maillage (« mini-mesh ») ;

Actuellement en cours de développement, **Vizir** se base sur les librairies Qt et **openGL**. Le format de fichier pris en charge pour le stockage des maillages et des solutions est **libmeshhb**, également développé par l'équipe GAMMA3.

### 4.1.1 Eléments et types de maillages

**Vizir** prend en charge plusieurs types de maillages décrits dans les lignes qui suivent. Les surfaces peuvent être maillées à l'aide de triangles et de

quadrilatères, tandis que les volumes peuvent être maillés par des tétraèdres, des hexaèdres, des prismes ou encore des pyramides. Chacun de ces éléments peut être représenté par interpolation à l'ordre 1 ou 2 (mis à part les prismes et les pyramides pour qui seule l'interpolation à l'ordre 1 est gérée). Les images ci-dessous donnent une idée des possibilités offertes par Vizir.

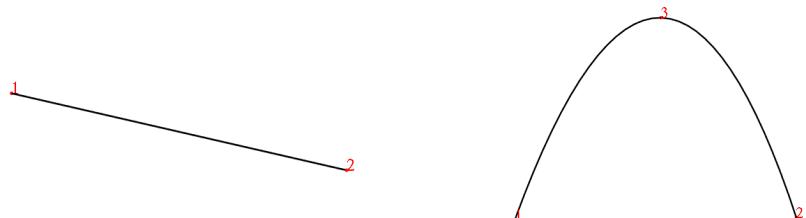


FIGURE 4.1 – Segments  $P^1$  et  $P^2$

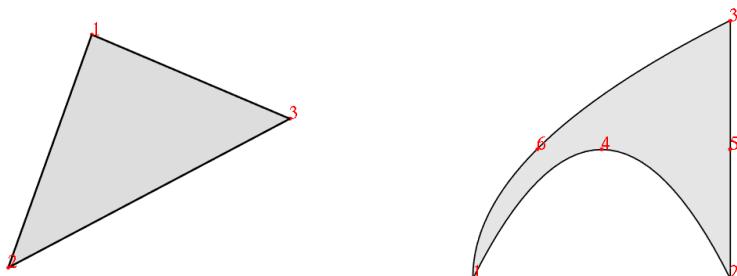


FIGURE 4.2 – Triangles  $P^1$  et  $P^2$

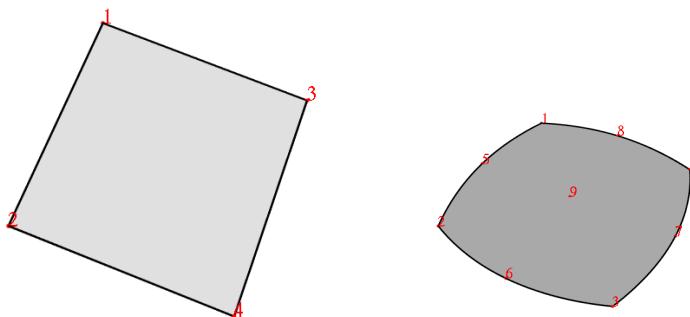


FIGURE 4.3 – Quadrilatères  $Q^1$  et  $Q^2$

#### 4.1 La visualisation de maillages avec **Vizir**

17

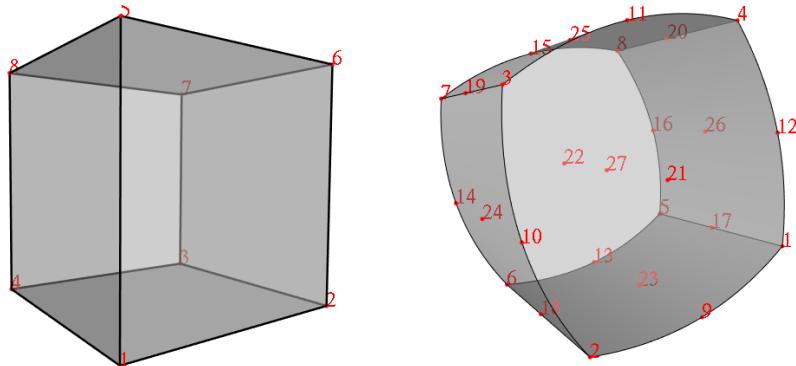


FIGURE 4.4 – Hexaèdres  $Q^1$  et  $Q^2$

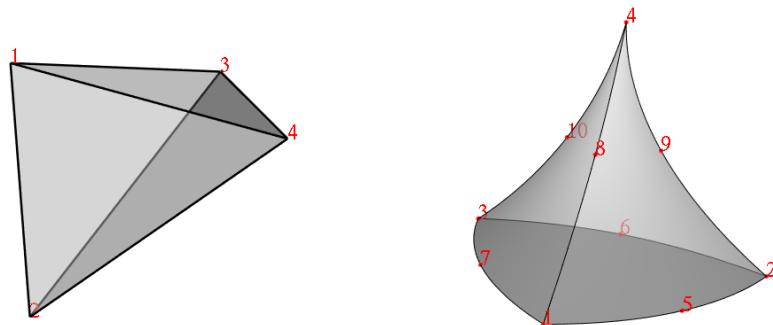


FIGURE 4.5 – Tétraèdres  $P^1$  et  $P^2$

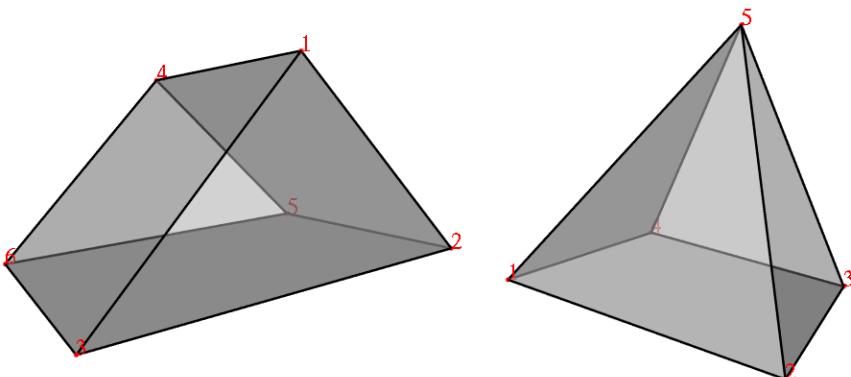


FIGURE 4.6 – Prisme et pyramide

D'autre part, Vizir prend en charge des maillages dits hybrides, au sens où ces maillages sont constitués à la fois d'éléments 2D et 3D (voir figure 4.7).

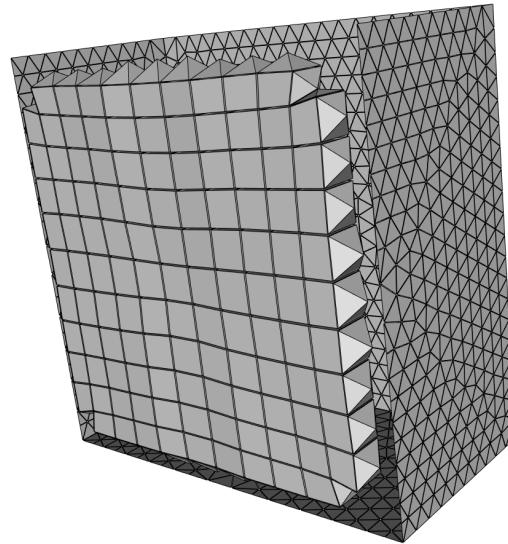


FIGURE 4.7 – Maillage hybride dont le volume intérieur est constitué de pyramides, d'hexaèdres et de tétraèdres, et dont les surfaces extérieures sont constituées de triangles

#### 4.1.2 Plan de coupe et mini-mesh

Pour visualiser l'intérieur du maillage, il est possible de définir un plan de coupe à la souris ou en définissant les coefficients  $(A_x, A_y, A_z, D)$  intervenant dans l'équation du plan souhaité :  $A_x x + A_y y + A_z z = D$  (voir figure 4.8). Tous les éléments du maillage se trouvant du côté du plan de coupe pointé par le vecteur directeur  $(A_x, A_y, A_z)^T$  sont alors masqués.

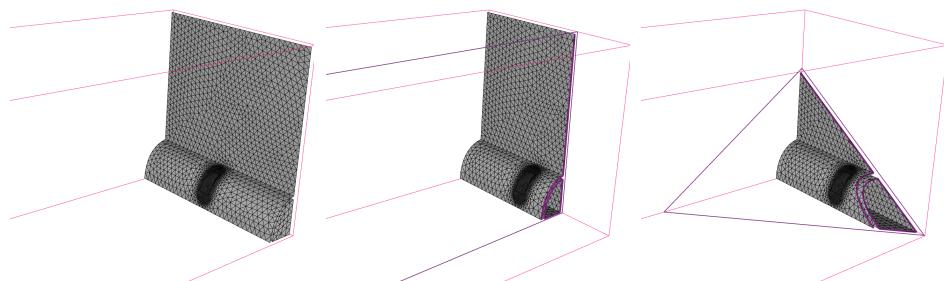


FIGURE 4.8 – Définition d'un plan de coupe. A gauche, le plan de coupe est désactivé, au milieu il est activé, et à droite il est orienté différemment.

Il est également possible d'afficher l'ensemble des éléments intersectés par le plan de coupe, et (facultativement) de les projeter de façon orthogonale sur ce plan (il est alors question de *capping*).

#### 4.1 La visualisation de maillages avec **Vizir**

19

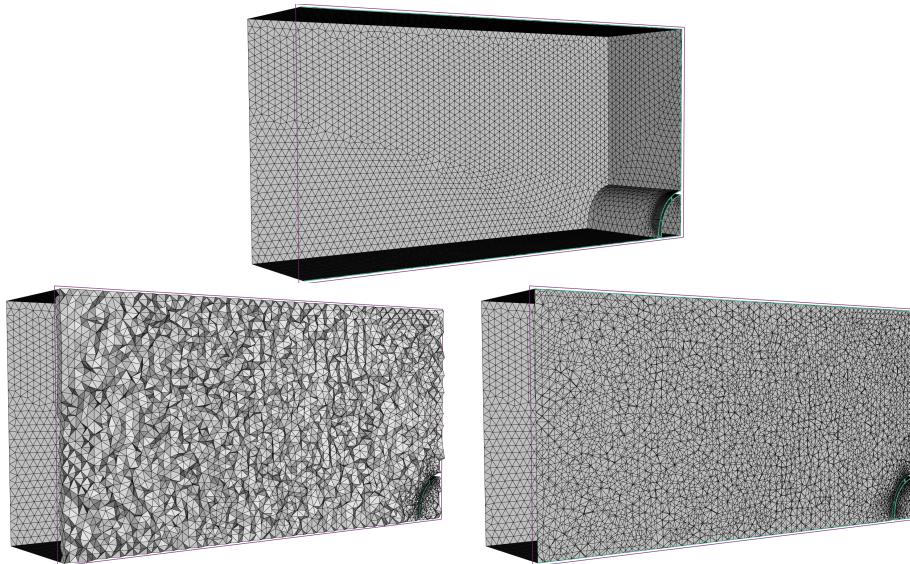


FIGURE 4.9 – Affichage des éléments intersectés par le plan de coupe. En haut, aucun élément n'est affiché, à gauche les éléments intersectés sont affichés et à droite ils sont projetés sur le plan de coupe avant d'être affichés.

Enfin, Vizir permet d'afficher une sous-région du maillage (« mini-mesh »). Pour cela, l'utilisateur choisit un élément du maillage (par exemple un tétraèdre), et tous les autres éléments sont masqués. Ensuite, l'utilisateur peut étendre ce sous-maillage en faisant réapparaître les voisins directs de cet élément initial. En itérant de la sorte de proche en proche, le mini-mesh souhaité est obtenu (voir figure 4.10).

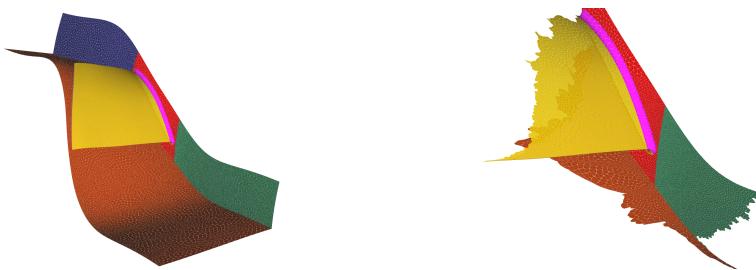


FIGURE 4.10 – Maillage d'un rotor (la pale est en jaune). A gauche : maillage initial, à droite : mini-mesh du même rotor.

##### 4.1.3 Système de coordonnées homogènes

Le but d'un système de coordonnées homogènes est de définir des transformations entre deux référentiels par le biais d'une multiplication matricielle. Utilisé dans Vizir, ce système répond en outre à plusieurs besoins :

- décrire la position et l'orientation d'un objet ;

- translater un objet ;
- effectuer une rotation d'un objet par rapport à un axe ;

Un point et un vecteur sont tous deux représentés comme des éléments de  $\mathcal{M}_{4,1}(\mathbf{R})$ , les trois premières composantes correspondant aux coordonnées dans le repère choisi, la dernière composante étant égale à 1 ou 0 selon que l'objet est un point ou un vecteur :

$$p = \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}, \quad \vec{q} = \begin{pmatrix} q_x \\ q_y \\ q_z \\ 0 \end{pmatrix}$$

Une translation selon  $\vec{v} \in \mathcal{M}_{3,1}(\mathbf{R})$  et une rotation de  $\theta \in [-\pi, \pi]$  autour de  $\vec{r} \in \mathcal{M}_{3,1}(\mathbf{R})$  (de matrice  $R_{\vec{r}}(\theta) \in \mathcal{M}_3(\mathbf{R})$ ) s'écrit alors :

$$p' = T p = \begin{bmatrix} R_{\vec{r}}(\theta) & \vec{v} \\ 0_{\mathcal{M}_{1,3}(\mathbf{R})} & 1 \end{bmatrix} p$$

Si l'on remplace le point  $p$  par un vecteur, on voit que la translation  $\vec{v}$  ne s'applique plus, seule la rotation est effective. Ceci est cohérent avec le fait qu'un vecteur est invariant par translation.

L'interaction entre l'utilisateur et Vizir se déroule comme suit :

1. La fenêtre Qt détecte les événements de la souris et du clavier (entre autres : clic gauche + déplacement  $\leftrightarrow$  rotation, clic milieu + déplacement  $\leftrightarrow$  translation, touche z  $\leftrightarrow$  zoom, touche Maj + z  $\leftrightarrow$  dé-zoom).
2. La matrice de transformation  $T$  correspondante est assemblée à l'aide d'OpenGL.
3. Les nouvelles coordonnées pour chacun des points sont obtenues en multipliant la matrice de transformation avec les anciennes coordonnées.
4. Une nouvelle image est générée par OpenGL et est affichée dans la fenêtre Qt.

#### 4.1.4 Le module `pyviz`

Le module Python `pyviz` développé au cours de ce stage est un *wrapper* Python de Vizir. Il permet de manipuler et visualiser des maillages dans l'interface graphique de Vizir depuis un script Python. Le détail des fonctionnalités intégrées à `pyviz` est donné ci-après.

Le module `pyviz` fournit les méthodes `read_mesh()` et `read_sol()` pour gérer les fichiers `.mesh(b)` et `.sol(b)` respectant les spécifications de la librairie `libmeshb`. Une fois qu'un tel fichier a été ouvert, `pyviz` stocke la représentation de ce maillage ou de cette solution dans un dictionnaire. Un exemple est donné ci-dessous :

## 4.1 La visualisation de maillages avec Vizir

21

```
1 import pyviz as vz
2 msh = vz.read_mesh('filename.mesh(b)')
3 sol = vz.read_sol('filename.sol(b)')
```

Pour les maillages 2D (resp. 3D), les coordonnées des points peuvent alors être trouvées dans la liste `msh['xy']` (resp. `msh['xyz']`). La clé '`Vertices`' est utile pour accéder aux références de chacun des points :

```
1 # Points toward msh['xy'] (resp. msh['xyz'])
2 msh['Vertices'][0]
3 # List of references associated to each vertex
4 msh['Vertices'][1]
```

Il est d'autre part possible de créer un dictionnaire de maillage sans avoir à lire de fichier. C'est notamment le cas si l'on désire générer le maillage automatiquement ou manuellement depuis le script Python. Avant de procéder il est nécessaire d'appeler la fonction `init_mesh()` afin de lier les deux listes `msh['Vertices'][0]` et `msh['xy']` ou `msh['xyz']` :

```
1 mesh = vz.init_mesh(dim=2) # The default dimension is 2
```

Pour ajouter un ensemble de points au dictionnaire du maillage, la méthode `add_vertices()` peut être employée comme ci-dessous. Par défaut tous les éléments de la liste `vertices_list` sont vérifiés afin d'anticiper certaines erreurs. Pour de meilleures performances, ce comportement peut être désactivé avec l'argument optionnel `check_items=False`.

```
1 pyviz.add_vertices(
2     mesh_dict, vertices_list, ref, check_items=False
3 )
```

Pour visualiser un maillage et sa solution (facultative) dans vizir, la ligne suivante suffit :

```
1 vz.show_mesh(msh, solution=sol)
```

Il y a également la possibilité d'enregistrer un dictionnaire représentant un maillage ou une solution dans un fichier `.mesh(b)` or `.sol(b)` grâce aux méthodes `write_mesh()` et `write_sol()` :

```
1 vz.write_mesh(msh, 'out.mesh(b)')
2 vz.write_sol(sol, 'out.sol(b)')
```

Enfin, `pyviz` peut gérer plusieurs options pour modifier l'affichage. Pour ce faire, il suffit de créer un dictionnaire d'options décrivant la configuration voulue. Un exemple peut être trouvé ci-dessous :

```
1 opt = {
2     # Enable the isolines (default is False)
3     'Isolines': True,
4
5     # Enable the isosurfaces (default is False)
6     'Isosurfaces': True,
7
8     # Set the tessellation quality , min=1, max=32
9     'Tesselation': 32,
10
11    # Customize the solution colormap. An other choice is
12    # to set this key to a predefined colormap such as :
13    # 'COOLWARM', 'MAGMA', 'INFERNO', 'PLASMA', 'VIRIDIS',
14    # 'JET', 'CUBEHELIX_BLUE', 'CUBEHELIX_BLUE_REV'
15    'Colormap': [
16        [.5, .1, .1],
17        [.5, .3, .3],
18        [.5, .5, .5],
19        [.5, .7, .7],
20        [.5, .9, .9],
21    ],
22
23    # Center the mesh arround the point (Px,Py,Pz) in the
24    # [Ox = (1,0,0) ; Oy = (0,1,0) ; Oz = (0,0,1)] basis
25    'Position': [Px, Py, Pz],
26
27    # Rotate the mesh prior to each axis:
28    # * Ax degrees prior to Ox = (1,0,0);
29    # * Ay degrees prior to Oy = (0,1,0);
30    # * Az degrees prior to Oz = (0,0,1);
31    'Angle': [Ax, Ay, Az],
32
33    # Enable the clip plane satisfying Ax + By + Cz = D
34    'ClipPlane': [A, B, C, D],
35
36    # Enable capping when a clip plane is activated
37    # (default is False)
38    'Capping': True,
39
40    # Enable the box (default is False)
41    'ShowBox': True,
42
43    # Enable the axis (default is False)
44    'ShowAxis': True,
45
46    # Enable negative background coloring
47    # (default is False)
48    'NegativeBackground': True,
49 }
```

## 4.2 Le remaillage avec **feflo.a**

23

```
50 vz.show_mesh(msh, solution=sol, options=opt)
```

L'intérêt du module `pyviz` est réel : il permet par exemple, en amont d'une présentation, de préparer une ou plusieurs prises de vue, ce qui peut constituer un gain de temps tout en facilitant la démonstration. Une autre possibilité est celle du prototypage de nouveaux algorithmes. Ceux-ci pourront être implémentés et testés en Python sur des maillages simples, avant d'afficher le résultat avec `pyviz`.

## 4.2 Le remaillage avec **feflo.a**

Le programme `feflo.a` est un remailleur permettant l'adaptation de maillages anisotropes et la génération de maillages de couche limite pour les géométries 2D, surfaciques et 3D.

Afin de fonctionner, `feflo.a` nécessite un maillage initial et un solveur avec lequel interagir. Etant donnée une première solution fournie par le solveur pour le maillage initial, `feflo.a` établit une nouvelle métrique  $\mathcal{M}$  dépendant de la position (voir section 2.2). Une fois cette métrique établie, un nouveau maillage (anisotrope) est généré, puis le solveur résout à nouveau le problème pour fournir une seconde solution. En itérant de la sorte un certain nombre de fois, on obtient un maillage raffiné au niveau des zones où le gradient de la solution est élevé. De cette manière, les variations de la solution sont mieux captées lors de l'itération finale, sans pour autant avoir à raffiner tout le domaine. On peut ainsi espérer obtenir une convergence plus rapide vers la solution du problème.

### 4.2.1 L'interface **amgio** pour gérer le format **SU2**

Comme indiqué dans la section précédente, `feflo.a` requiert une interaction avec un solveur. Lors de ce stage, le solveur **SU2** a été utilisé. **SU2** est un logiciel libre dont une partie du développement est effectué à Standford, aux Etats-Unis, et possède son propre format de maillage `.su2` différent de `libmeshb`. La première étape a donc été le développement d'un module Python appelé `amgio` (*Adaptative Mesh Generation I/O*) permettant de convertir ce format **SU2** vers `libmeshb` et inversement.

### 4.2.2 Le module **pyamg**

M. Rigal, Implémentation d'une interface Python pour les algorithmes de remaillage, Stage à l'Inria, 2018

## **Chapitre 5**

# **Conclusion**

### **5.1 Apport du stage**

### **5.2 Perspectives**

M. Rigal, Implémentation d'une interface Python pour les algorithmes de remaillage, Stage à l'Inria, 2018

# Bibliographie

- [1] FREY PASCAL-JEAN ; GEORGE PAUL-LOUIS. *Le maillage facile*. Hermès Science publications Paris, 2003, p. 175.
- [2] PATRICK LAUG. *Contribution à la génération automatique de maillages de qualité pour la simulation numérique*. <https://tel.archives-ouvertes.fr/tel-00012000>. 2006.
- [3] P.L. GEORGE. *Génération des maillages pour la simulation par Eléments Finis de problèmes physiques*. <https://hal.archives-ouvertes.fr/jpa-00246222>. 1990.