Université Paris Diderot

I.N.R.A

Encadrant : Philippe Andrey
Professeur référant : Yan Jurski

Rapport de stage : Implémentation de nouveaux descripteurs de formes pour l'analyse d'images 3D de cellules végétales

Emile Rolley

Juin 2019

Table des matières

1 Introduction

Durant un mois, j'ai effectué un stage à Institut Jean-Pierre Bourgin (IJPB) au sein de l'équipe de modélisation et d'imagerie numérique. L'IJPB fait partie des plus grands laboratoires de recherche européens en biologie végétale. L'équipe dans laquelle j'ai travaillé a pour objectifs de concevoir et implémenter différentes méthodes et outils pour analyser, traiter et modéliser des données d'imagerie végétale. J'ai pu notamment utiliser la librairie `freedlib` (C++) sur laquelle repose `Free-D`, un logiciel de reconstruction et de modélisation d'images 3D développé par l'équipe.

Ma mission a été d'enrichir cette librairie, notamment en implémentant dans un premier temps une mesure de circularité, puis une mesure de compacité applicable sur des formes 3D. J'ai ensuite travaillé sur un projet plus spécifique qui consistait à concevoir et à implémenter une classe permettant l'analyse et la modélisation d'interfaces des objets d'images 3D labélisées.

2 Descripteurs de formes

En analyse d'images, les descripteurs sont utilisés sur une image une fois que celle-ci ai été prétraitée et segmentée. Le prétraitement permet de restaurer ou d'améliorer la qualité de l'image à fin quelle restitue aussi fidèlement que possible le signal d'origine. Une fois l'image prétraitée, la segmentation permet de définir les formes qui composent l'image. Ces formes sont généralement labélisées et peuvent être ainsi quantifiées grâce aux descripteurs à travers différentes grandeurs telles que la position, la taille et la forme.

2.1 Une nouvelle mesure de circularité

Mon premier projet a été l'implémentation d'une nouvelle mesure de circularité basée sur un moment invariant de Hu. En effet, la manière la plus courante de mesurer la circularité d'une forme F est de calculer `Aire_de_F / Perimetre_de_F`^{2}, cependant, ce rapport est très sensible aux déformations du fait que le périmètre croît bien plus vite que l'aire dès que la forme s'éloigne d'un disque. Dans l'article [2], J.žunić propose une nouvelle mesure de circularité basée sur un moment invariant de Hu ne dépendant pas du périmètre ni de l'aire. Cette nouvelle mesure est censée être plus robuste. Dans [2], il démontre que pour une forme F plane, ce moment invariant de Hu noté $\mathcal{I}_{Hu}(F)$ atteint un minimum de $1/2\pi$ lorsque F est un disque.

Le moment $\mathcal{I}_{Hu}(F)$ est défini par :

$$\mathcal{I}_{Hu}(F) = \frac{\mu_{2,0}(F) + \mu_{0,2}(F)}{\mu_{0,0}(F)^2},$$

où $\mu_{p,q}(F)$ est le *moment centré* d'ordre (p,q) de F définit par :

$$\mu_{p,q}(F) = \int\int_F \left(x - \frac{m_{1,0}(F)}{m_{0,0}(F)}\right)^p \left(y - \frac{m_{0,1}(F)}{m_{0,0}(F)}\right)^q \mathrm{d}x\,\mathrm{d}y\,,$$

où $m_{p,q}(F)$ est lui-même le *moment* d'ordre (p,q) de F défini par :

$$m_{p,q}(F) = \int\int_F x^p y^q \,\mathrm{d}x\,\mathrm{d}y.$$

La nouvelle mesure de circularité normalisée est donc :

$$\mathcal{C}(F) = \frac{\mu_{0,0}(F)^2}{2\pi(\mu_{2,0}(F) + \mu_{0,2}(F))}.$$

On a $\mathcal{C}(F) \in (0, 1]$ pour toute forme F et $\mathcal{C}(F) = 1 \Leftrightarrow F$ est un disque.

A partir de cette formule j'ai pu concevoir une classe permettant le calcul de cette mesure ainsi que des différents moments. La difficulté a été de créer une classe permettant à elle seule le calcul des moments et ce pour tous les types d'images supportées par la classe `Image2` de `freedlib`. En effet, une image numérique contient un nombre finis de points, appelés pixels, ils sont situés sur une grille régulière et à chaque pixel est associé une valeur représentant une couleur ou un niveau de gris. C'est cette valeur qui suivant la méthode d'acquisition et de traitement va être codé de différentes manières. La classe `Image2` ne peut lire qu'un seul format d'image : le TIFF. Cependant, les valeures des pixels de l'image peuvent être stockées dans différents type : `unsigned char`, `unsigned short`, `unsigned int`, `short`, `int`, `float`, `double`. Pour traiter cette diversité, la classe `Image2` est une interface entre l'utilisateur et les classes *templatées* sur les différents types vu précédemment.

Pour ce faire, j'ai réutilisé une des architecture déjà utilisée dans `freedlib` pour construire la classe `Moment` (voir Fig. 1). Les méthodes publiques de la classe `Moment` font office d'interface, elles vérifient seulement si l'image fournie est à la bonne dimension et délèguent les calculs à l'instance de la classe `Implementation` *templatée* sur le type des valeurs de l'image (`ValueType`).

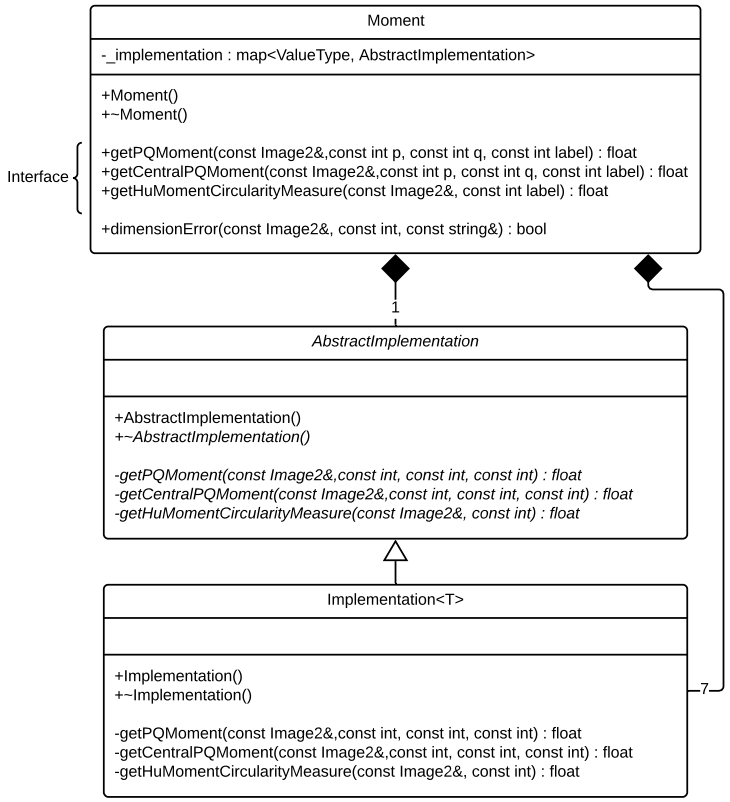

Figure 1: Organisation de la classe Moment

2.2 Tests de la nouvelle mesure de circularité

Après avoir implémenté la classe `Moment`, j'ai comparé les résultats de $\mathcal{C}(F)$ à ceux de la mesure de circularité d'`ImageJ` (logiciel de traitement et d'analyse d'images) sur un jeu de 99 images de disques ('Circular masks' d'`ImageJ`) de rayon croissant (voir Fig. 2).

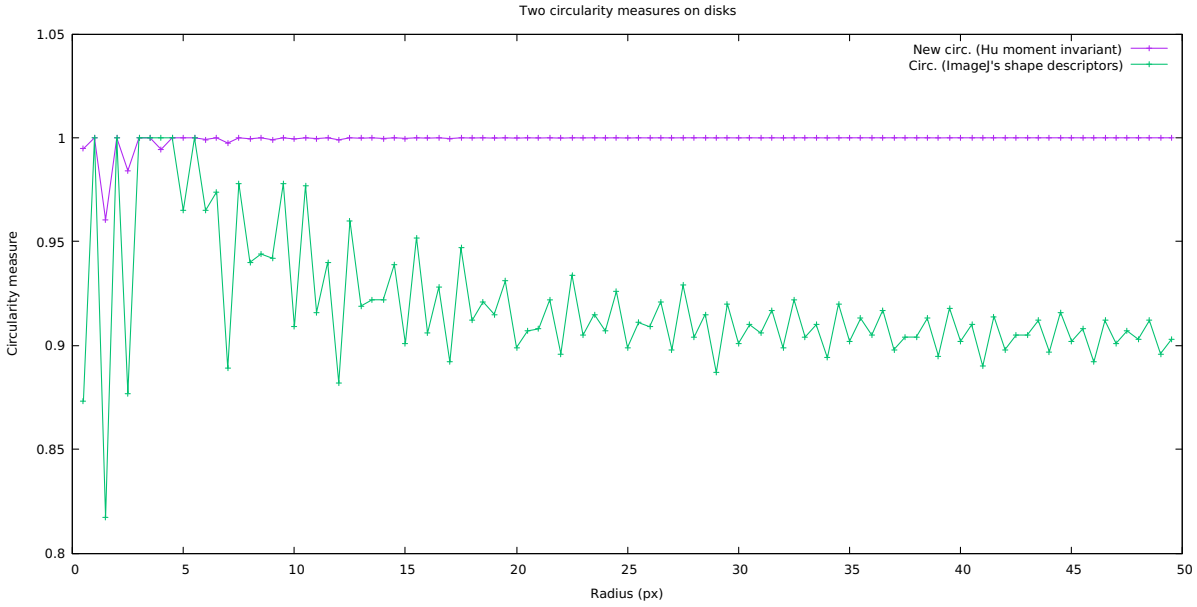

Figure 2: Graphique des deux mesures de circularités pour 99 images de disque de rayon croissant.

On remarque que $\mathcal{C}(F)$ se stabilise pour des disques de rayons supérieur à 5 pixels alors que celle d'ImageJ ne l'est qu'à partir d'un rayon supérieur à 20 pixels. Cette fluctuation des mesures avant un certain seuil s'explique par la discrétisation de l'image. En effet, un disque d'un rayon d'un pixel est en réalité qu'une croix. Cependant, dès que le rayon est supérieur à 5, la discrétisation n'influence pratiquement pas $\mathcal{C}(F)$.

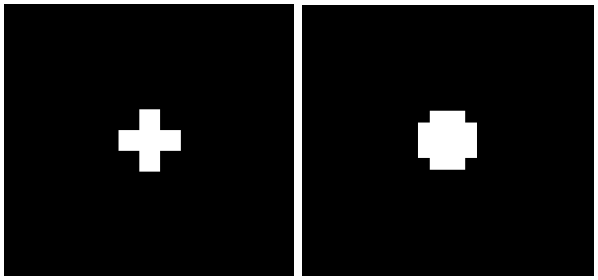

Figure 3: Exemples de discrétisation d'un disque de rayon 1 et 5 pixels.

Création d'un jeu de données

Pour aller plus loin dans l'évaluation de la mesure, j'ai implémenté une nouvelle classe `DrawShape` permettant de générer des images test consistant en un disque avec des déformations sinusoïdales à sa périphérie. Grâce à cette classe j'ai pu créer un jeu de données comprenant 40 images de fréquence (voir Fig. 5) ou d'amplitude (voir Fig. 6) variable.

Figure 4: Image de disques déformés sinusoïdalement. Avec de gauche à droite : une amplitude de 7 pour une fréquence de 22 ; une amplitude de 10 pour une fréquence de 5 ; une amplitude de 38 pour une fréquence de 5.

La méthode permettant de dessiner les disques est la suivante : `void DrawShape::drawSinusoidalDisk(int X, int Y, float radius, float freq, float range)`. Soit d la distance entre le centre (X,Y) du disque et un pixel p de l'image :
p est considéré comme faisant partie du disque, si et seulement si, $d \leq \sin(\theta \cdot freq) \cdot range + radius$.

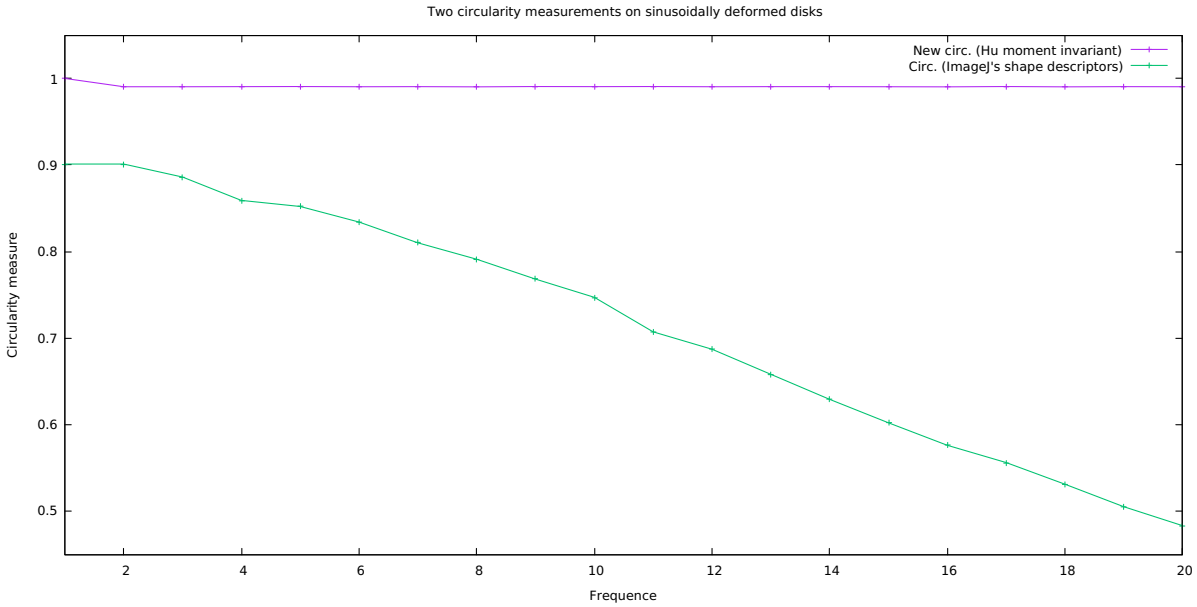

Figure 5: Graphique des deux mesures de circularités pour 20 disques déformés sinusoïdalement avec $range = 7$, $radius = 100$ et $freq \in [0,20]$

La Fig.3 montre la robustesse de $\mathcal{C}(F)$ qui sera de ce fait beaucoup moins sensible aux bruits qui pourraient parasiter une image. Cela est confirmé par la Fig.4. Cependant, $\mathcal{C}(F)$ atteint une valeur de 0.8 pour une amplitude de 38 dans la Fig.4 ce qui pourrait paraître un peut trop élevé par rapport à ce que l'on pourrait attendre en voyant la Fig. 4.

Avec les résultats précédents on peut légitimement se poser la question : $\mathcal{C}(F)$ est-t-elle réellement une mesure de circularité ou bien une mesure d'isotropie ? En effet, plus la forme F est répartie de façon homogène autour de son centre plus $\mathcal{C}(F)$ sera élevée, et ce, même si la forme est très éloignée du disque. Ce n'est pas pour autant que la mesure est inintéressante. Les descripteurs de formes ne sont jamais utilisés seuls, en

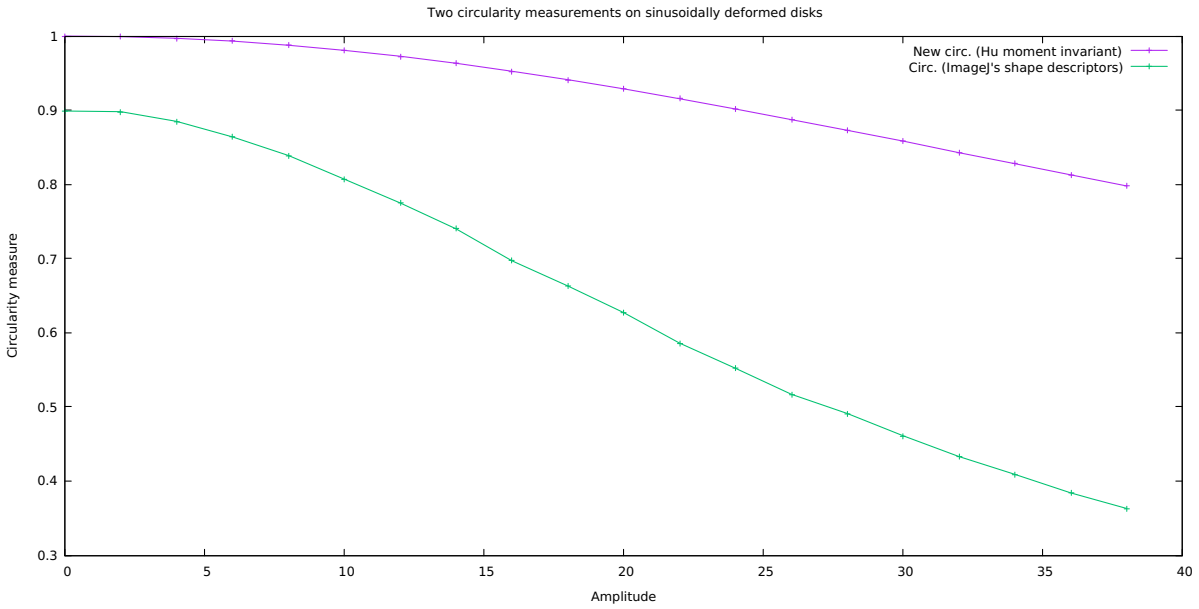

Figure 6: Graphique des deux mesures de circularités pour 20 disques déformés sinusoïdalement avec $freq = 5$, $radius = 100$ et $range \in [0, 38]$

effet, plus on applique de descripteurs différents à une forme, plus la représentation que l'on se fait de cette forme sera précise.

2.3 Une nouvelle mesure de compacité

En 2015, J. Žunić publie un nouvel article [1] dans lequel il propose un analogue 3D au premier moment invariant de Hu

$$\mathcal{I}(F) = \frac{\mu_{2,0,0}(F) + \mu_{0,2,0}(F) + \mu_{0,0,2}(F)}{\mu_{0,0,0}(F)^{5/3}}$$

pour une forme 3D donnée F. De la même manière que pour la mesure de circularité, il en déduit une mesure de compacité

$$\mathcal{K}(F) = \frac{3^{5/3}}{5(4\pi)^{2/3}} \cdot \frac{1}{\mathcal{I}(F)}.$$

J'ai donc enrichi la classe `Moment` avec les méthodes permettant de calculer les *moments* et les *moments centrés* d'ordre (p,q,r) ainsi que $\mathcal{K}(F)$.

Vérification de la nouvelle mesure de compacité

Pour tester les méthodes ajoutées, j'ai utilisé le même jeu de données qui est utilisé dans [1], à savoir le *McGill 3D Shape Benchmark data set*. J'ai donc pu comparer les mesures faites avec la méthode `float Moment::get3DCompactnessMeasure(const Image2, const int)` et les résultats de [1]. Je l'ai fait sur une dizaine de formes, les résultats concordant j'en ai déduit que que ma méthode calculait la bonne valeur $\mathcal{K}(F)$. Je n'ai pas fait plus de tests, en effet, ceux effectués dans [1] sont assez complets et je n'ai pas jugé pertinent de les refaire pour trouver les mêmes résultats.

2.4 Conclusion

Au bout de deux semaines, j'ai réussi à implémenter et tester la classe `Moment` ce qui m'a permis de me familiariser avec l'environnement de travail et avec `freedlib`. J'ai pu également faire une présentation (LaTeX/ `beamer`) à l'équipe sur le travail effectué jusqu'ici.

Dans [1], une mesure "d'ellipsoïdité" est également présenté dont $\mathcal{K}(F)$ en est une mesure particulière (celle de la sphère). Cependant, je n'ai pas réussi à implémenter la méthode permettant de la calculer, j'ai donc préféré commencer le deuxième projet pour avoir une chance de le terminer dans le temps restant.

3 Traitement des interfaces des cellules d'embryons de plantes, à partir d'images 3D labélisées

Une grande partie des recherches de l'équipe de modélisation et d'imagerie numérique consiste à étudier la morphogenèse et le développement des organes chez les plantes, notamment l'organisation cellulaire lors de l'embryogenèse.

Il est donc intéressant d'étudier la forme des cellules et comment elle peut être influencée par ses voisines. C'est dans ce but qu'il m'a été confié de concevoir et implémenter une classe permettant de calculer, stocker et représenter les interfaces ainsi que les faces des cellules d'une image 3D segmentée d'embryon, comme Fig. 7.

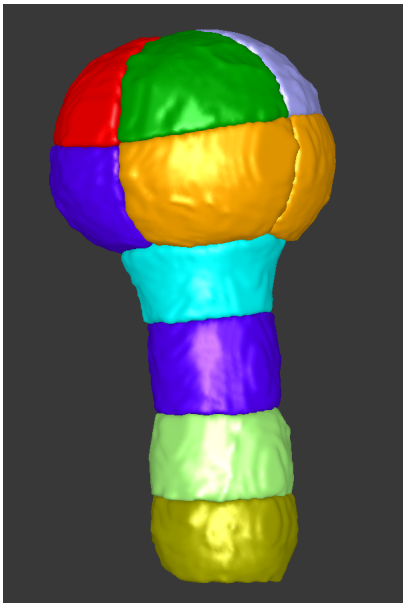

Figure 7: 12 cellules d'un embryon et 4 de son suspenseur visualisées à l'aide de `sviewer` (version autonome du module de rendu 3D de`Free-D`).

3.1 Organisation des classes

J'ai créé une première classe `Interface` qui représente une interface entre deux régions (cellules) de labels différents et contient l'ensemble des coordonnées des voxels (équivalent d'un pixel en 3D) la composant ; une seconde classe `Facet` représentant une face d'une cellule, soit un ensemble d'`Interface`. (voir Fig. 8)

En reprenant la même architecture que la classe `Moment`, j'ai créé la classe `InterfaceAnalysis` (voir Fig. 9) qui détermine et stocke les interfaces ainsi que les faces des régions d'une image labélisée.

3.2 Calcul des faces

L'objectif de ce deuxième projet est d'arriver à créer une classe permettant de déterminer et enregistrer les faces d'une cellule. Cependant, une face étant un ensemble d'interfaces, il faut d'abord déterminer l'ensemble de ces interfaces. Pour cela, j'ai réutilisé une méthode de la classe `RegionAnalysis2` de `freedlib` permettant de calculer les interfaces de toutes les régions d'une image, je l'ai simplement modifiée pour qu'elle fonctionne avec les deux nouvelles classes `Interface` et `Facet`.

Une fois les interfaces calculées, pour déterminer si deux interfaces voisines font parties d'une même face on a choisi de comparer l'angle entre leur vecteurs normaux à un seuil paramétré par l'utilisateur. Pour savoir si deux interfaces sont voisines, j'ai utilisé la classe `RegionAnalysis2` qui calcule et stocke un graphe d'adjacence sous forme d'une matrice d'entiers. Le vecteur normal $\vec{n}$ d'une interface est la moyenne de tous les vecteurs normaux des voxels la constituant. En revanche, le vecteur normal $\vec{n}_F$ d'une face est la moyenne pondérée des $\vec{n}_i$ des i interfaces la constituant :

$$\vec{n}_F = \frac{1}{N_F} \left(\sum_{i \in interfacesOfF} N_i \cdot \vec{n}_i \right)$$

où N_i est l'effectif en voxels de i.

Après le calcul des interfaces et des vecteurs normaux, il ne restait plus qu'à construire les faces selon l'Algorithme 1.

Algorithm 1 computeRegionFacet

Require: $interfaces$
  for all $interfaces$ **do**
    $facet \leftarrow interface$
    $facets \leftarrow facet$
  end for
  while $TRUE$ **do**
    if $facets.size() > 1$ **then**
      $facet1, facet2 \leftarrow getFacetWithMinimumAngles()$
      if $getAngle(facet1, facet2) \leq threshold$ **then**
        $merge(facet1, facet2)$
      else
        $return$
      end if
    else
      $return$
    end if
  end while

Cependant, après quelques tests pour trouver le meilleur seuil, je me suis aperçu que certaines interfaces de quelques voxels seulement formaient des faces à elles toutes seules, comme par exemple la facette couleur parme de la Fig. 10.

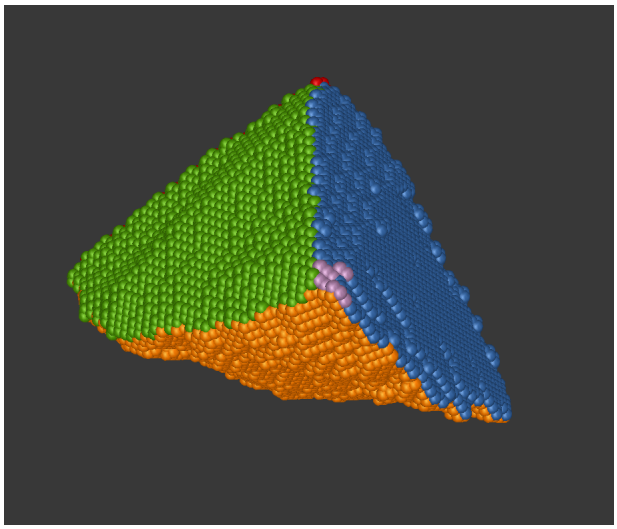

Figure 10: Faces d'une cellule pour un seuil de 38 degrés visualisées avec `sviewer`.

Pour éviter de devoir augmenter le seuil et risquer de rassembler des faces distinctes en une seule, j'ai choisi d'ajouter un deuxième seuil. Après avoir calculé les faces selon l'algo.1, la méthode `computeRegionFacets` vérifie que l'aire de la surface de chacune des faces est supérieur à un pourcentage de l'aire de la surface de la cellule, sinon tant que la face est trop petite, elle est ajoutée à celle dont l'angle entre leurs vecteurs normaux est le plus petit.

Grâce à ce deuxième paramètre, la méthode n'est plus sensible aux légères perturbations qui peuvent avoir lieu au moment de la prise de vue ou bien lors de la segmentation de l'image, comme le montre la Fig. 11, en effet, la facette parme a été ajoutée à la face bleu.

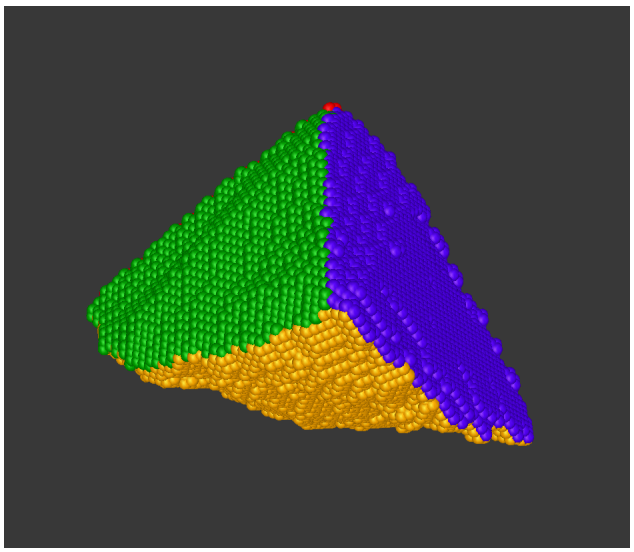

Figure 11: Faces d'une cellule pour un seuil de 38 degrés pour l'angle et de 4% pour l'aire minimum d'une face visualisées avec `sviewer`.

3.3 Conclusion

Je n'ai pas eu le temps de faire des tests plus poussés sur plusieurs images d'embryon. Malgré cela, je pense que la classe `InterfaceAnalysis` est tout à fait utilisable. Il y a quand même quelques optimisations possibles notamment sur les utilisations de pointeurs au lieu de références et `InterfaceAnalysis` pourrait être divisées en deux classes `InterfaceAnalysis` et `FacetAnalysis` pour plus de lisibilité et modularité.

4 Conclusion

Avec ces deux projets, j'ai pu découvrir un nouveau langage de programmation mais également de nombreux outils tel que l'IDE `QtCreator` et son système de débogage, LaTeX et `ImageJ`. L'utilisation de `freedlib`, m'a permis de me familiariser avec la programmation orientée objet.

Au delà des compétences techniques, j'ai surtout pu découvrir le métier de chercheur et plus précisément, la force de la pluridisciplinarité et la place que peut occuper l'informatique dans un centre de recherche en biologie végétale.

Références

[1] J. Žunić, K. Hirota, D. Dukić, and M. Ali Aktaş, *On a 3d analogue of the first hu moment invariant and a family of shape ellipsoidness measures*, Machine Vision and Applications, 27 (2015).

[2] J. Žunić, K. Hirota, and P. Rosin, *A hu moment invariant as a shape circularity measure*, Pattern Recognition, 43 (2010), pp. 47–57.

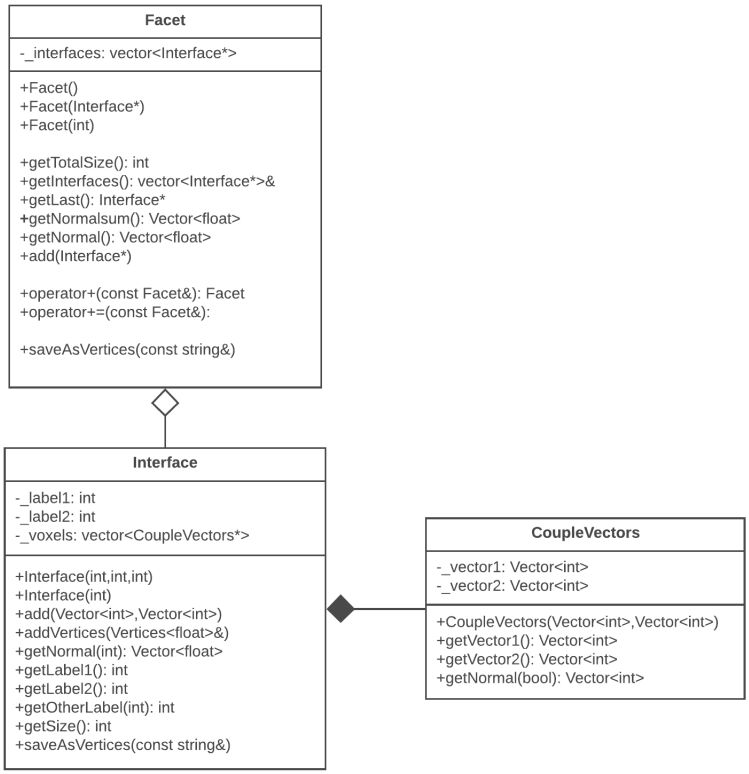

Figure 8: Organisation des classes Facet et Interface

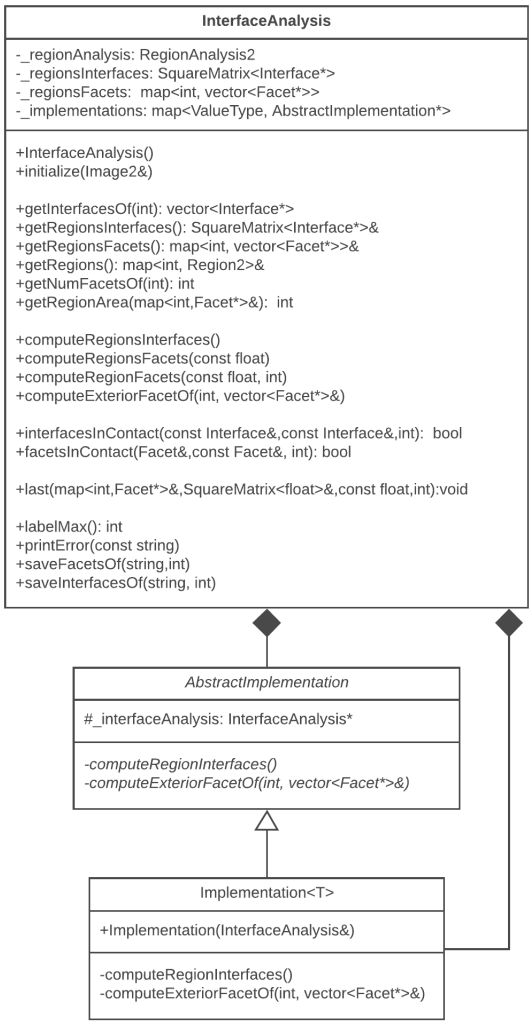

Figure 9: Organisation de la classe InterfaceAnalysis