

# Rapport de pré-étude

vendredi 24 octobre 2008

Projet ***Monofin***

4<sup>ième</sup> année Informatique  
INSA Rennes

YOANN CHAUDET  
PAUL GARCIA  
QUENTIN GAUTIER  
NICOLAS LE SQUER  
NICOLAS MUSSET  
XAVIER VILLOING

## Encadreurs

PATRICE LEGUESDRON  
LAURENT MONIER  
FULGENCE RAZAFIMAHERY

## Sommaire

I.	Présentation du projet .....	3
A.	Contexte .....	3
B.	Les attentes .....	3
C.	Spécifications générales .....	4
II.	Modélisation de la palme .....	5
A.	Modélisation à la souris .....	5
B.	Reconnaissance d'image .....	6
1.	Algorithme de Sobel .....	6
2.	Algorithme de Canny .....	7
3.	Algorithme des contours actifs .....	8
C.	Génération du modèle 3D .....	9
1.	Triangulation de Delaunay .....	9
2.	Heightmap .....	10
III.	Configuration de la simulation .....	12
IV.	Affichage des résultats .....	13
V.	Choix techniques .....	14
A.	Le langage de programmation .....	14
1.	Java .....	14
2.	C++ .....	14
3.	Bref comparatif .....	15
4.	Bilan .....	15
B.	L'environnement de développement .....	16
C.	La plateforme collaborative .....	16
D.	Le format STL .....	17
1.	Version ASCII .....	17
2.	Version binaire .....	18
E.	Les librairies .....	18
1.	Qt .....	18
2.	Qhull .....	20
3.	Computational Geometry Algorithms Library (CGAL) .....	21
F.	Logiciels scientifiques .....	22
1.	COMSOL .....	22
2.	MAPLE .....	22
3.	MATLAB .....	22
VI.	Estimation de la charge initiale .....	23
VII.	Conclusion .....	24
VIII.	Bibliographie .....	25
IX.	Annexes .....	26
A.	Décompte des heures .....	26

## I. Présentation du projet

### A. Contexte

Actuellement, l'étude du comportement d'une monopalme en milieu liquide est réalisée à l'aide de deux procédés. Une première méthode consiste à filmer la palme dans l'eau lorsqu'elle est animée par le mouvement d'un plongeur ou d'un robot. La seconde méthode consiste à utiliser un banc d'essai statique pour calculer les forces de résistance et d'élasticité de la palme. Toutes les données recueillies par ces deux procédés servent à l'évaluation de la palme pour les professionnels.

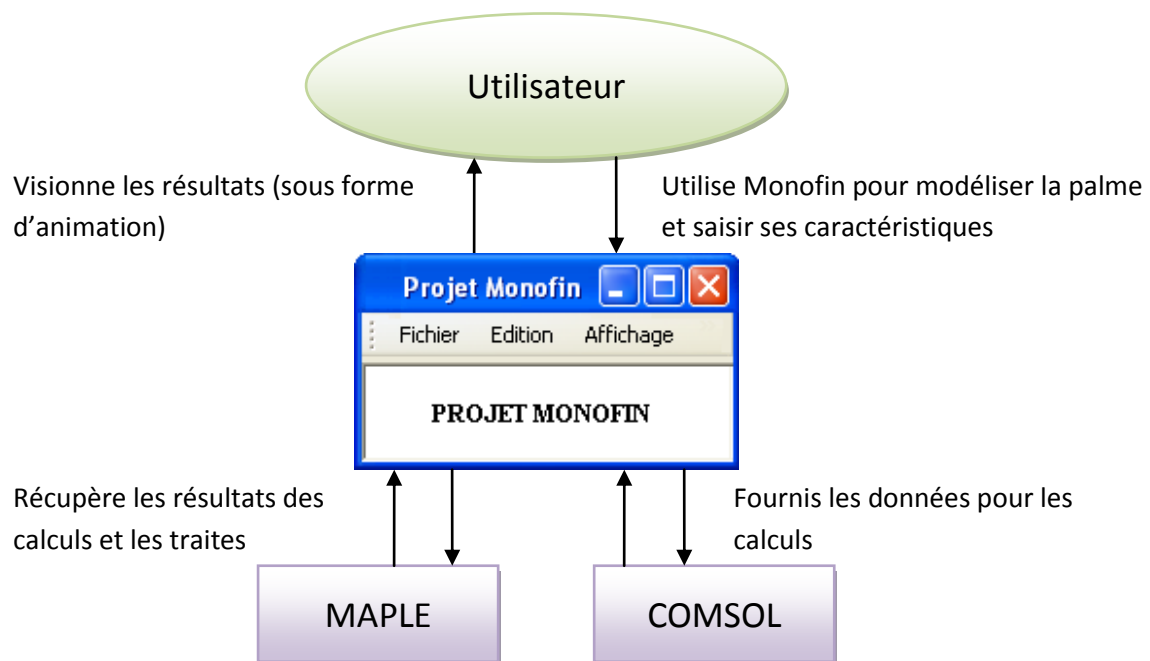
La réquisition d'une piscine et l'installation du robot pour tester les palmes forment un processus long et contraignant. C'est dans ce contexte que des professeurs de l'université de Rennes 1 et de l'INSA de Rennes ont commencé à mettre au point des modèles mathématiques dans des logiciels de simulation (et notamment COMSOL). Ces modèles permettent de simuler le comportement d'une monopalme en fonction de sa forme, son poids, son élasticité et en fonction des caractéristiques du liquide.

Les logiciels de simulation sont relativement complexes à utiliser. Malgré le gain de temps par rapport à l'installation d'un robot dans une piscine, le processus de simulation reste difficile. Il faut dans un premier temps modéliser la palme en trois dimensions puis bien paramétrer la simulation.

C'est ainsi qu'est né notre projet, celui-ci devant faire acte de passerelle entre les logiciels de simulation et les utilisateurs.

### B. Les attentes

Le schéma ci-dessous illustre le rôle de passerelle de notre projet.



Le projet doit répondre aux problèmes des utilisateurs, à savoir :

- Comment modéliser la palme en trois dimensions ?
- Comment paramétrer la simulation ?
- Comment récupérer les résultats de la simulation ?

Le logiciel devant entrer en interaction avec des utilisateurs non-informaticiens, il se doit de proposer une interface claire et intuitive, tout en restant puissante et le moins restrictive possible.

Pour faciliter la modélisation, l'utilisateur pourra dessiner sa palme dans un cadre prévu à cet effet (avec la possibilité d'insérer une photo en arrière-plan pour faciliter le dessin), fournir une photographie de la palme et appliquer un algorithme de détection de contours ou bien utiliser un modèle de palme prédéfini. Pour terminer, l'utilisateur saisira divers paramètres pour la simulation et le logiciel se chargera de demander à COMSOL et MAPLE d'effectuer le travail de calcul. À la fin de la simulation, le logiciel permettra à l'utilisateur de récupérer les résultats de la simulation et d'afficher sous forme d'animation le mouvement de la palme.

### C. Spécifications générales

Voici la liste des spécifications générales reprise à partir de chaque compte-rendu de réunion :

- Interface utilisant une notion d'étapes à effectuer
- Méthode de dessin par point et par symétrie
- Mettre à la disposition des utilisateurs une bibliothèque de formes de palmes prédéfinies
- MAPLE sera utilisé pour des résolutions analytiques et fournira des résultats simplifiés
- COMSOL fournira des résultats précis et notamment une animation de la palme
  - Il faudra permettre de modifier l'échelle des couleurs (en permettant d'agrandir le dégradé)
- La sortie devra être une vidéo de la palme en mouvement :
  - Animation 2D pour les résultats de MAPLE
  - Animation 3D pour les résultats de COMSOL
- Si le logiciel n'intègre pas le support de MAPLE nativement, il devra néanmoins contenir les bases permettant de rajouter ce support facilement. La priorité est donnée à l'intégration de COMSOL
- Les paramètres significatifs d'une simulation sont : la masse volumique, le coefficient de Poisson<sup>1</sup> et le module de Young<sup>2</sup>
- Permettre d'importer un modèle 3D complexe modélisé avec un modèleur 3D de type 3DS Max

Nous allons présenter ci-après le résultat de nos recherches sur les différents moyens de parvenir à nos fins.

---

<sup>1</sup> « Le coefficient de Poisson permet de caractériser la contraction de la matière perpendiculairement à la direction de l'effort appliqué ».

<sup>2</sup> Le module de Young est « le rapport entre la contrainte de traction appliquée à un matériau et la déformation qui en résulte ».

## II. Modélisation de la palme

Afin de pouvoir lancer une simulation des résultats par COMSOL, il est nécessaire de fournir un modèle de la palme en trois dimensions. Cependant, comme il n'est pas prévu que l'utilisateur dispose systématiquement d'un modèle précis et au bon format, il est plus efficace de lui permettre d'entrer des photos ou de dessiner lui-même une forme. Dans les deux cas, l'objectif est d'obtenir une ou plusieurs images en interne et de s'en servir pour reconstruire la palme. De plus, il est important de fournir un format compatible avec COMSOL (cf. Le format STL page 17).

### A. Modélisation à la souris

L'objectif est de permettre à l'utilisateur de dessiner une forme qui puisse être relativement complexe de la façon la plus libre possible et intuitivement.

Une première approche qui fut rapidement abandonnée était de permettre un dessin libre via la souris ou une tablette graphique, mais il sembla rapidement clair que cette solution ne pourrait pas aboutir à une modélisation précise de la monopalme. Il était donc nécessaire d'utiliser des formes géométriques et d'agir sur un minimum de points pour réaliser le contour, comme des *splines*, par exemple.

Deux autres aspects doivent également être pris en compte. Le relief d'une palme n'est pas égal, selon que l'on s'éloigne de la position des pieds. L'interface de dessin doit donc disposer d'un espace spécial pour représenter un profil en coupe de la palme, et gérer ainsi son épaisseur. De plus il est possible de tirer partie de la symétrie de la palme et donc de faciliter le travail de l'utilisateur en ne lui faisant travailler que sur une moitié.

La solution qui est donc retenue est de permettre dans un premier temps à l'utilisateur de sélectionner une forme prédéfinie de palme dans une liste qui lui servira de patron, ou de rentrer sa propre image qui sera également appliqué en arrière-plan de la fenêtre de dessin. Un plan de symétrie pourra alors être placé par-dessus le patron, automatiquement dans le cas d'une forme prédéfinie, et géré par l'utilisateur dans le cas d'une image. En ce qui concerne le profil, le même procédé sera utilisé, dans une fenêtre de dessin sous-jacente. L'utilisateur pourra ensuite placer un certain nombre de formes afin de réaliser le contour de la palme (arcs de cercles, ellipses, lignes brisées, *splines*, ...). Il pourra agir sur n'importe quelle moitié du dessin. Chaque élément modifié sur une moitié le sera également sur l'autre, ce qui offrira en permanence une vue d'ensemble de la palme. Un certain nombre de fonctions d'aides seront disponibles, afin d'annuler, de reprendre, de sauvegarder ou de charger un dessin, de vérifier la fermeture d'un contour ou d'avoir un aperçu de la forme finale. Enfin, après validation par l'utilisateur et vérification du logiciel, le dessin réalisé sera récupéré comme s'il s'agissait d'une photo, et le logiciel poursuivra son traitement en interne.

L'image ainsi récupérée pourra, avec le profil, générer un modèle 3D précis et complexe, sans que l'utilisateur n'ait eu à réaliser des tâches complexes. De plus, il pourra réaliser plusieurs variations autour d'une même forme de départ, afin de déterminer laquelle est la plus efficace. La réalisation de la forme en 3D sera donc parfaitement transparente.

## B. Reconnaissance d'image

Une seconde approche pour modéliser une palme en 3D serait de partir d'une image existante de monopalme (vue de dessus) et par le biais d'un algorithme de reconnaissance de contour, extraire la forme de cette monopalme. Cette forme pourra être exploitée de différentes manières selon le désir de l'utilisateur. Si la forme extraite semble satisfaisante, l'utilisateur pourra passer à l'étape suivante en créant l'objet en 3D. Si l'utilisateur désire modifier cette forme, celle-ci sera vectorisée et pourra être modifiée selon la méthode de la modélisation à la souris.

Il reste donc à trouver un algorithme de reconnaissance de contour approprié au projet. L'utilisateur devra fournir une image de bonne qualité (sans bruit) et avec un contraste suffisant pour pouvoir différencier la monopalme de l'arrière plan, cet arrière plan devra être dans le meilleur des cas unicolore. En partant de ces hypothèses, l'algorithme utilisé pourra être relativement simple et de faible complexité. De plus il est préférable d'utiliser un algorithme déjà existant et au mieux déjà implémenté dans le langage qui sera utilisé dans le projet. Ainsi, dans le cadre de nos recherches, trois solutions ont été trouvées et étudiées : l'algorithme de Sobel, l'algorithme de Canny et enfin l'algorithme des contours actifs.

### 1. Algorithme de Sobel

L'algorithme de Sobel est l'un des opérateurs les plus simples utilisés en traitement d'image et donne des résultats très concluant. Il parcourt l'image donnée en entrée et calcule le gradient de l'intensité de chaque pixel de l'image. Ce gradient indique la direction de la plus forte variation du clair au sombre et le taux de changement dans cette direction. Ainsi l'algorithme peut détecter des changements soudains d'intensité dans l'image qui sont probablement due à des bords.

Pour pouvoir calculer le gradient d'intensité d'un pixel, il suffit de connaître l'intensité des 8 pixels l'entourant. Ces 9 pixels peuvent être représentés dans une matrice 3x3. On effectue un produit de convolution avec des matrices qui permettent d'obtenir des approximations des dérivées horizontales et verticales du pixel central (les matrices utilisées pour le produit de convolution sont spécifiques au traitement d'image et dans ce cas à la détection de contour). A partir des deux dérivées obtenues, il est possible de calculer une approximation de la norme du gradient de chaque pixel de l'image. Ainsi les pixels sont remplacés par ce gradient d'intensité et il en ressort une image avec une approximation des contours. Par exemple l'image B est obtenue à partir de l'image A à l'aide de l'algorithme.



L'avantage de cet algorithme est qu'il est très simple à implémenter et possède une faible complexité (proportionnel au nombre de pixels de l'image).

Cependant il est nécessaire que l'image donnée en entrée n'ait pas de pixel parasite (bruit) qui serait détecté comme contour par l'algorithme et donc amplifié. L'algorithme ne permet pas non plus de détecter des faux contours et d'en faire abstraction.

Ces derniers critères ne jouent pas nécessairement en défaveur de cette solution, si les utilisateurs possèdent des images de qualité correcte, cet algorithme semble tout à fait satisfaisant pour la détection de contour

## **2. Algorithme de Canny**

Le filtre de Canny est basé sur l'algorithme de Sobel, il dispose d'étapes supplémentaires qui permettent de travailler sur des images plus complexes. Il permet, en plus de l'algorithme de Sobel, de faire abstraction d'image bruitée et de détecter les faux contours. L'algorithme de Canny fonctionne en trois étapes :

### ***Etape 1 : Réduction du bruit de l'image***

Le but de cette opération est d'éliminer un maximum de pixels parasites qui pourraient être détectés comme contour. Un filtre Gaussien 2D est utilisé pour lisser l'image.

### ***Etape 2 : Calcul du gradient d'intensité***

C'est à cette étape que l'algorithme de Sobel est repris.

### ***Etape 3 : Suppressions des non-maxima***

Une fois l'étape 2 effectuée, l'image est constituée de gradients d'intensité. Cependant, certaines zones peuvent être détectées comme des contours même si le gradient d'intensité n'est pas très élevé. Le but de cette étape est de détecter les pixels possédant un gradient d'intensité très élevé, qui sont alors reconnus comme contours.

Une méthode de seuillage par hystérésis est utilisée pour cette étape. En définissant un seuil bas et un seuil haut, il est possible de détecter les pixels considérés comme des contours et d'éliminer les autres.

L'avantage de cette méthode est qu'elle limite la détection de faux contours et offre une plus grande précision que l'algorithme de Sobel. Cependant, elle est plus difficile à implémenter et ne s'avérera pas forcément nécessaire. Sa complexité reste proportionnelle au nombre de pixels de l'image.

Au final, cette solution pourra être retenue s'il s'avère qu'il est trop difficile de détecter les contours uniquement avec l'algorithme de Sobel. Cet algorithme, plus poussé, pourra alors le remplacer.

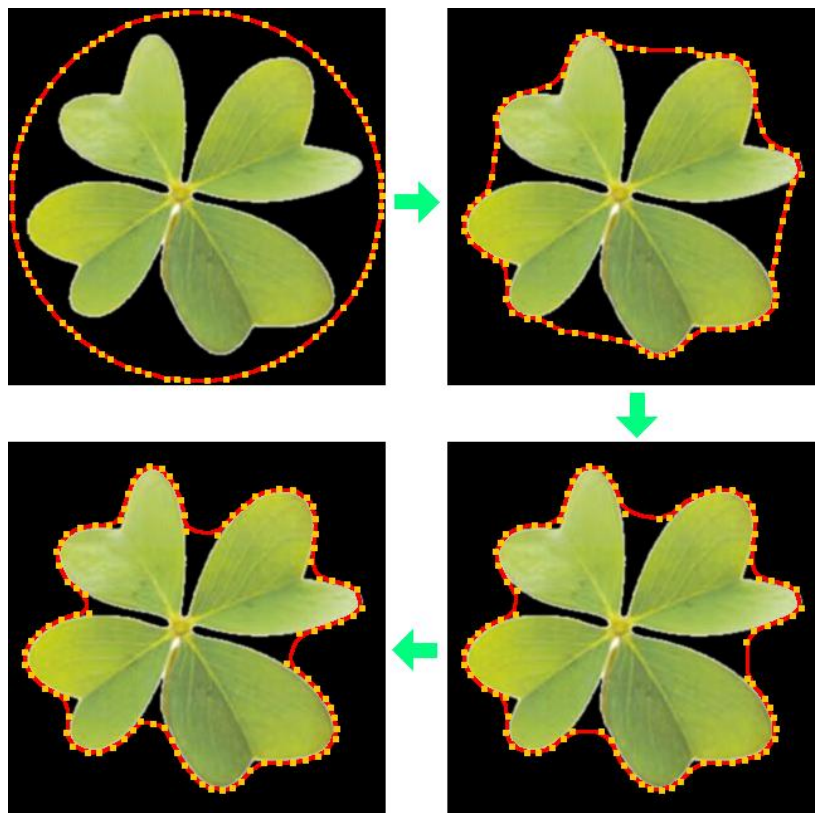
### 3. Algorithme des contours actifs

L'algorithme des contours actifs est un algorithme globalement assez simple permettant de déterminer le contour d'un objet sur un support de type photo. Il est utilisé notamment en vision artificielle. Il s'agit d'un ensemble de points partant du bord de l'image et qui, itération après itération, se rapproche du centre. Le mouvement est conditionné par le calcul de deux énergies. L'énergie interne et l'énergie externe. Le calcul des énergies interne et externe sont deux heuristiques définies à l'avance et selon les besoins. En voici deux exemples :

L'énergie interne s'occupe de notions comme la courbure du contour ou la régularité d'espacement des points. Elle cherche à empêcher qu'un point ne s'éloigne trop du contour dessiné par les autres points. Elle est minimale pour un cercle dont tous les points sont régulièrement espacés.

L'énergie externe quant à elle s'intéresse à la variation du gradient de couleurs. Un point qui passe d'une zone avec un gradient donné, à une zone avec un gradient qui varie fortement, va faire augmenter l'énergie externe.

Le but est de trouver l'état minimisant l'énergie globale de la courbe. Exemple de déroulement de l'algorithme :



L'avantage de cet algorithme est sa relative simplicité. Cependant il est nécessaire que le contour à détecter soit bien différencié du fond qui doit être très proche d'un ton uni.

Il existe un certain nombre de bibliothèques Java et C++ implémentant des algorithmes de contours actifs (Active contour ou *Snakes* en anglais), car ces types d'algorithmes se sont développés



avec la reconnaissance de visages ou même pour implanter des algorithmes de vision pour les robots.

### C. Génération du modèle 3D

La génération du modèle 3D est une étape importante dans le processus de modélisation de la palme. En effet, COMSOL effectuera sa simulation à partir d'une représentation 3D de la palme : un *mesh*. Le terme *mesh* vient de l'anglais « maillage » et en 3D, représente un objet constitué de polygones et en général plus spécifiquement de triangles.

#### 1. Triangulation de Delaunay

La triangulation de Delaunay a été la première méthode envisagée pour modéliser une palme en trois dimensions.

Soit  $T(P)$  une triangulation pour un ensemble de points  $P$ . On dit que  $T(P)$  est une triangulation de Delaunay s'il n'existe aucun point appartenant à  $P$  qui soit contenu à l'intérieur d'un cercle circonscrit à un quelconque triangle de  $T(P)$ .



La définition de la triangulation de Delaunay s'applique également dans un espace à trois dimensions en remplaçant les cercles circonscrits par des sphères circonscrites (et les triangles par des tétraèdres). La triangulation de Delaunay assure un certain nombre de propriétés intéressantes sur les triangulations si bien que de nombreux algorithmes ont été développés pour les calculer. Exemples de propriétés :


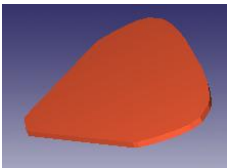
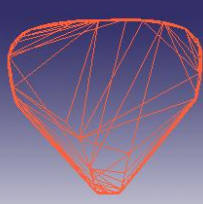

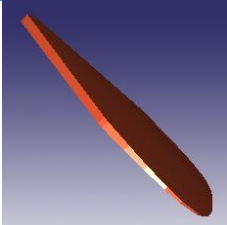
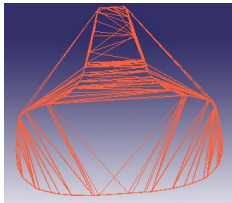

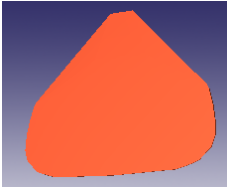
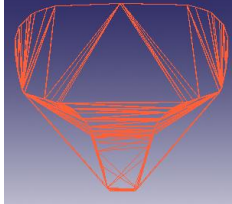
- L'union de tous les simplexes<sup>3</sup> de la triangulation de Delaunay  $T(P)$  est l'enveloppe convexe de  $P$ .
- Dans le plan, une triangulation de Delaunay  $T(P)$  maximise l'angle minimal parmi tous les triangles. Il n'existe pas d'autres triangulations de l'ensemble  $P$  telles que l'angle minimal pour l'ensemble soit plus grand que celui de la triangulation de Delaunay  $T(P)$ .

Dans la pratique, les triangulations de Delaunay forment des maillages composés de peu de triangles et grands de surcroît. Pour se donner une idée des possibilités de la modélisation d'une monopalme à partir d'une triangulation de Delaunay, nous avons réalisé un prototype utilisant la librairie Qhull (cf. Qhull page 20). Qhull implémente en particulier un algorithme capable de calculer une triangulation de Delaunay dans  $n$  dimensions à partir d'un ensemble de points.

Le prototype en lui-même est très simple, il s'agit d'un programme en ligne de commande qui accepte en entrée une image et retourne en sortie un fichier au format STL (cf. Le format STL page 17). L'image est en niveau de gris, les pixels blancs ne sont pas « traités », les autres définissent en fonction de leur luminosité un point en trois dimensions (plus un point est noir, plus il possèdera une grande valeur pour sa composante Z). L'ensemble des points obtenu est envoyé à la librairie Qhull qui nous retourne la triangulation de Delaunay. Les images ci-dessous représentent des résultats obtenus grâce à la triangulation de Delaunay.

---

<sup>3</sup> Le simplexe est l'analogue au triangle dans un espace à  $n$  dimensions

Image source	Monopalme modélisée	Maillage
		
		
		

Comme on peut le constater (c'est encore plus flagrant sur la dernière ligne du tableau), les palmes générées ne respectent pas correctement l'image source. En effet, la première propriété d'une triangulation de Delaunay est d'engendrer l'enveloppe convexe de tous ses points. Même si dans la réalité, la plupart des monopalmes approche une forme convexe, dans le cadre de notre projet il est intéressant de ne pas contraindre l'utilisateur à ce critère. La triangulation de Delaunay n'est donc pas la solution que nous retiendrons pour la génération de nos palmes en 3D.

## 2. Heightmap

Afin de trouver une nouvelle solution au problème de la modélisation de monopalmes en 3D, il est utile de repartir à la base et de décrire une monopalme géométriquement :

- Une monopalme est symétrique selon un axe
- Une monopalme peut être modélisée par deux surfaces superposées l'une sur l'autre :
  - Une surface avec du relief pour le dessus
  - Une surface plane pour le dessous

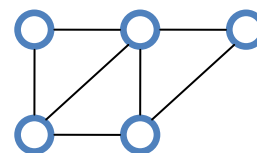
S'il est possible de mailler la surface du dessus de la palme, on peut obtenir de manière très simple la surface plane du dessous (par exemple en mettant la composante z de chaque point du maillage à 0). Il ne restera alors plus qu'à mailler les deux surfaces entre elles pour obtenir un modèle tridimensionnel de notre monopalme.

Le précédent prototype a presque déjà apporté la solution à notre problème. En effet, la génération d'une surface avec du relief est très facile à partir d'une *heightmap*. Une *heightmap* est une image en niveau de gris pour laquelle on associe à chaque pixel de coordonnées (x,y) une composante z calculée à partir de la luminosité du pixel. La luminosité d'une couleur est définie par la formule suivante :

$$\text{Luminosité (couleur)} = \frac{\text{couleur.rouge} + \text{couleur.vert} + \text{couleur.bleu}}{3}$$

Les composantes rouges, vertes et bleues de chaque couleur sont des entiers compris entre 0 et 255, la luminosité est la moyenne de ces trois composantes.

A partir d'une *heightmap* on est en mesure de récupérer une grille de points 3D qu'il ne reste plus qu'à mailler. Le maillage d'une grille est beaucoup plus simple que le maillage d'un ensemble de points quelconques.

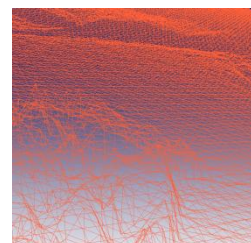


Un second prototype a été réalisé, celui-ci prend en entrée une *heightmap* de la surface du dessus de la monopalme et retourne un modèle 3D au format STL. Le tableau ci-dessous présente les résultats obtenus.

Heightmap	Modèle de face	Modèle de profil

On peut tout de suite noter que les palmes convexes sont correctement modélisées. Il n'y a plus de restriction sur la forme de la palme.

Cependant le maillage est maintenant « naïf » et contient beaucoup plus de triangles que nécessaire. Par exemple, le modèle 3D de la dernière ligne du tableau qui est composé de 620 946 sommets et de 206 982 faces. Il faut également noter que si la *heightmap* n'est pas « propre », le modèle 3D généré ne le sera pas également (cf. la dernière ligne du tableau dont l'image a été récupérée sur Internet sans traitement particulier).



Mis à part les quelques inconvénients cités ci-dessus la modélisation par *heightmap* est très simple à mettre en place et fournit des résultats convenables. Par la suite il sera peut-être intéressant de simplifier le maillage obtenu naïvement à l'aide d'un algorithme (cf.

Computational Geometry Algorithms Library (CGAL) page 21).

### III. Configuration de la simulation

La simulation des calculs via COMSOL doit être transparente pour l'utilisateur. L'interface doit donc être conçue comme un intermédiaire entre COMSOL et celui-ci. Il faut donc définir les différentes étapes que l'utilisateur devra réaliser, pour les décomposer et lancer la simulation une fois qu'elles seront toutes remplies.

Au lancement du logiciel il sera demandé dans un premier temps, soit de choisir parmi l'un des deux modes de calcul, soit de charger une simulation préalablement sauvegardée. Le premier mode analytique, fera intervenir le logiciel MAPLE, pour une simulation approximée. Le mode numérique quant à lui permettra le dialogue avec COMSOL, pour obtenir une simulation dynamique des mouvements de la palme, ainsi que les contraintes mécaniques.

Dans ce mode, la configuration de la simulation sera décomposée en plusieurs étapes. Ainsi la fenêtre de l'application changera selon l'étape, tout en conservant une barre de progression indiquant les étapes passées, et celles qu'il reste à franchir. A tout moment il sera possible de revenir à une étape antérieure via la barre de progression, mais l'accès à une étape suivante ne sera possible qu'après vérification de toutes les données entrées pendant l'étape courante. Enfin, lorsque toutes les étapes auront été validées, un bouton de lancement de la simulation apparaîtra, et après calcul, les résultats seront proposés.

La première étape sera de sélectionner la technique de modélisation de la palme. Il sera en effet possible de partir d'une photo et de lancer une génération automatique, ou pour réaliser des variations de dessiner soi-même la palme. Dans le cas d'une photo, l'étape suivante sera de spécifier le chemin d'accès à celle-ci, et de lancer la construction du modèle en 3D. Si l'utilisateur préfère réaliser un contour manuellement, la fenêtre de dessin apparaîtra. Comme décrit précédemment, il devra sélectionner un patron de fond, puis réaliser le dessin proprement dit. Il sera intéressant de proposer à l'utilisateur un aperçu en 3D de son travail lors du dessin, selon une orientation prévue. Enfin une fois le dessin validé, l'application générera un modèle 3D à partir de l'image, comme dans le cas d'une photo. En interne, la différence entre les deux approches sera que la photo subira un certain nombre d'opérations pour être épurée, ce qui sera épargné à l'image dessinée.

Une fois le modèle 3D réalisé et vérifié, l'étape suivante consistera à renseigner les grandeurs physiques relatives à la simulation (module de YOUNG, coefficient de POISSON, masse volumique de la palme etc.). Ici, une simple vérification sera réalisée sur la cohérence des valeurs et, le cas échéant, un message demandant une correction sera envoyé à l'utilisateur.

L'étape suivante permettra de rentrer les paramètres de la simulation tels que l'échelle de temps, la fréquence des impulsions données à la palme et leur force. Une fois encore la cohérence des valeurs rentrées sera vérifiée et leur correction demandée, s'il y a lieu.

La dernière étape sera de sélectionner les informations rendues par la simulation. L'utilisateur choisira de cocher des cases en face des données qui l'intéressent pour la simulation et l'animation de la palme. Le format des résultats sera également paramétrable pour pouvoir les récupérer facilement et les exploiter.

Enfin après avoir eu la possibilité de sauvegarder sa progression, l'utilisateur pourra lancer la simulation. Ainsi, s'il souhaite ne modifier que certains paramètres, pour comparer des résultats, il n'aura pas à recommencer toutes les étapes de la configuration.

### IV. Affichage des résultats

Une fois toutes les données rentrées dans COMSOL, la simulation sera lancée et les informations dynamiques seront récupérées et transmises au logiciel MATLAB, pour la réalisation d'une vidéo du mouvement de la palme en 3D.

Les résultats seront donc en deux parties. D'une part, les données physiques comme la portance, la traînée, les zones et les temps d'efforts maximaux seront affichées en haut de la fenêtre des résultats. Ces données seront également exportables dans un format choisi par l'utilisateur. La vidéo représentant la simulation dynamique sera d'autre part jouée dans la fenêtre, et pourra être sauvegardée selon la volonté de l'utilisateur.

Dans la mesure où COMSOL ne permet pas de générer une vidéo satisfaisante du mouvement de la palme, c'est à MATLAB que reviendra cette tâche. Les données dynamiques du résultat fournit par COMSOL seront donc entrées dans MATLAB, et celui-ci réalisera une vidéo selon un angle de vue et un code de couleur que l'on pourra déterminer lors de la configuration de la simulation.

La génération de la vidéo, transparente pour l'utilisateur, devra être porteuse d'informations au niveau des efforts subits par la palme. Pour cela, il est nécessaire d'adopter une échelle de couleurs cohérente. En effet les zones de la palme proches des pieds du nageur, ainsi que les zones des extrémités, subissent des efforts moindres alors que la zone centrale est fortement sollicitée. C'est au niveau de cette zone que les couleurs devront être le plus significatives. Il faudra donc récupérer les valeurs extrêmes des efforts subits par la palme, afin de dérouler l'échelle de couleurs autour de ceux-ci. Les efforts faibles seront représentés sur une échelle ayant peu de variations, ce qui permettra d'avoir une représentation plus fine au niveau des zones à efforts importants. Cette échelle de couleurs sera automatiquement paramétrée dans MATLAB grâce aux informations fournies par COMSOL et aux spécifications de l'utilisateur lors de la configuration.

Enfin, la fenêtre des résultats proposera à l'utilisateur de relancer la simulation en modifiant quelques paramètres, ce qui aura pour effet de revenir à la fenêtre de configuration. Un nouveau cycle pourra alors être engagé, autant de fois que nécessaire.

## V. Choix techniques

### A. Le langage de programmation

Ce projet étant essentiellement centré autour de la réalisation d'une interface graphique et de la manipulation de données graphiques ou géométriques, notre choix s'est naturellement porté vers des langages orientés objet. Parmi ceux que nous avons déjà abordés à l'INSA se trouvent Java et C++. Il est donc nécessaire de faire un choix, car l'utilisation conjointe des deux langages pour ce projet ne semble pas judicieuse et poserait plus de problèmes.

#### 1. Java

Développé par Sun, Java est depuis peu *open-source*. Ceci étant, si Java était retenu, nous n'utiliserions qu'une version officielle pour éviter tout problème de compatibilité.

L'un des principaux avantages de Java est sa portabilité native. C'est-à-dire qu'un même programme est portable sur toutes les plateformes supportées par Java sans avoir besoin de réécrire du code ou de recompiler les sources. À l'inverse, un reproche qui revient de manière récurrente concerne ses problèmes de performances et d'utilisation de la mémoire. Java étant un langage semi-compilé, semi-interprété (le code Java est compilé en *bytecode* qui est interprété par la machine virtuelle), son exécution sollicite généralement beaucoup de ressources mémoire. Le programmeur n'a pas accès à la gestion de la mémoire, puisque le *garbage collector* s'en charge.

#### 2. C++

C++ est un langage orienté objet. Contrairement à Java qui est objet « pur », C++ accepte la définition de fonctions en dehors de tout contexte objet. Au contraire de Java, C++ n'est pas directement portable. Ses types primitifs sont dépendants de la plateforme. Par contre, son exécution est beaucoup plus rapide puisque C++ est entièrement compilé. En contrepartie, le programmeur doit la plupart du temps gérer lui-même la mémoire.

Une autre caractéristique de C++ est la notion de *template* qui est plus souple qu'en Java. Enfin, notons que l'héritage multiple et la surcharge des opérateurs sont des avantages non-négligeables par rapport à Java.

### 3. Bref comparatif

Le tableau suivant est issu du LSV (laboratoire spécification et vérification) de l'École Normale Supérieure de Cachan. Il présente succinctement les langages Java et C++.

Java	C++
<b>Généralités</b>	
sans préprocesseur, sans variable globale	avec préprocesseur, avec variable globale
<b>À propos de l'objet</b>	
langage objet « pur », sans héritage multiple, sans surcharge d'opérateur, « liaison dynamique » de toutes les méthodes (sauf celles déclarées <i>final</i> )	langage orienté objet, avec héritage multiple, avec surcharge d'opérateur, seules les <i>virtual</i> fonctions sont liées dynamiquement
<b>À propos des types primitifs</b>	
tout est objet sauf les types primitifs, les types primitifs sont portables (big-Endian), caractère 16-bits Unicode, initialisation automatique	il existe aussi des types <i>struct</i> , <i>union</i> , <i>enum</i> , tableaux... les types primitifs sont "plateforme-dépendants", caractère 8-bits ASCII, initialisation à la charge du programmeur
<b>À propos des pointeurs et structures de données</b>	
les objets sont manipulés par référence, mais il n'y a pas de manipulation explicite ni d'arithmétique de pointeur les références sur les tableaux ne peuvent pas être manipulées comme des pointeurs, avec test automatique de débordement de tableau tableaux multidimensionnels pouvant être non réguliers (les lignes d'une "matrice" peuvent être de longueurs variables), objet <i>String</i> sans <i>typedef</i>	il existe les opérateurs <i>-&gt;</i> , <i>&amp;</i> , <i>*</i> ,  les tableaux peuvent être manipulés avec l'arithmétique sur les pointeurs sans test automatique de débordement de tableau, tableaux multidimensionnels réguliers dont la taille est fixée à la déclaration,  les chaînes de caractères sont des tableaux de caractères terminées par un zéro, avec <i>typedef</i>
<b>Divers</b>	
pré-compilé puis interprété, portable	compilé, architecture-dépendant

### 4. Bilan

Notre projet, en plus de l'interface graphique, va nécessiter des calculs conséquents sur des objets graphiques en 2D ou en 3D. Ces calculs devront s'effectuer le plus rapidement possible pour l'utilisateur. Nous devons donc rechercher de bonnes performances. Celles-ci sont offertes par C++. Dans la mesure où la librairie Qt est portable, le reste de la programmation devrait consister en l'implémentation d'algorithmes qui ne poseront pas de problème de portabilité.

Certaines caractéristiques du C++, telles que l'héritage multiple ou les notions de pointeurs et de références apporteront plus de souplesse à notre programmation. Notons cependant que Java aurait aussi pu convenir. Mais l'existence d'autres librairies implémentées en C++ (notamment CGAL) a finalement confirmé le choix de C++.



## B. L'environnement de développement

Il existe une multitude d'environnements de développement (IDE) qui prennent en charge les langages Java et C++. Parmi eux, seul un nombre plus restreint permet une utilisation plus rapide et efficace de Qt. Eclipse IDE et Visual Studio offrent tous deux des outils de qualité professionnelle.

Cependant, Eclipse a l'avantage d'offrir des fonctionnalités telles que la vérification de la syntaxe en cours de frappe, ainsi que le *refactoring*<sup>4</sup>. De plus, un plugin prêt-en-main existe pour l'utilisation de Qt, que ce soit avec Java ou C++. Dans le cas de Visual Studio, nous n'avons pas trouvé d'explications claires qui permettent d'intégrer Qt, si ce n'est en achetant une licence commerciale de Qt (ce que nous ne ferons pas).

La plupart des membres de l'équipe ont une plus grande expérience avec Eclipse qu'avec Visual Studio. De plus, le projet étant *open-source*, il pourra être utile de vérifier son fonctionnement sous Linux. Visual Studio étant un produit 100% Windows, nous l'écartons comme choix d'environnement de développement.

D'autres IDE, notamment *open-source*, existent également. Mais plutôt que d'apprendre à utiliser un nouvel environnement, il nous est apparu judicieux de profiter de l'expérience que nous avons déjà avec Eclipse.

## C. La plateforme collaborative

Dans un projet comme le nôtre, il n'est pas concevable de ne pas se servir d'un minimum d'outils collaboratif, sans lesquels le déroulement serait rendu difficile et hasardeux. Heureusement, sont présents sur le web, un certain nombre de sites qui proposent gratuitement ce genre de service, pourvu que le logiciel développé soit publié avec une licence libre. C'est le cas de notre projet.

A l'heure actuelle, trois sites majeurs regroupent la grande majorité des projets *open-source*. Le plus ancien et le plus populaire est *Sourceforge.net*. Viennent ensuite *BerliOS* et le plus récent *GoogleCode*, issu des équipes de R&D de Google.

*BerliOS* et *Sourceforge.net* offrent des services similaires. Il est assez difficile de les différencier et il est fort probable que leur subtilité ne nous intéresse pas dans ce projet. Le choix de *BerliOS* s'est fait, entre-autre, sur le fait qu'une partie de l'interface est traduite en français.

Parmi les services proposés, plusieurs ont particulièrement attirés notre attention. Tout d'abord, le support des systèmes de version CVS et SVN est assuré. Ceci est d'autant plus intéressant qu'Eclipse gère nativement CVS. Ainsi, son utilisation sera facilitée. Notons qu'il existe aussi un plugin SVN pour Eclipse. La possibilité d'héberger des fichiers est aussi un point intéressant : un espace public accessible par FTP permettra de partager des fichiers entre nous ou avec nos encadreurs, que l'on soit ou non à l'INSA. *BerliOS* dispose aussi d'un système de *bug tracking*. Il s'agit d'une application en ligne, où les utilisateurs (ou nous-mêmes) pourront soumettre des bugs à travers un système de tickets. Les membres de l'équipe seront prévenus et un statut sera associé au bug. Cela permettra notamment de garder une trace des bugs rencontrés (même ceux qui seront déjà résolus).

---

<sup>4</sup> Le *refactoring* est une opération qui consiste en la maintenance du code d'un programme. Pour un langage de programmation orientée objet le *refactoring* peut par exemple permettre de renommer des membres de classe et de répercuter les changements dans tout le code source (commentaires inclus). Le *refactoring* est couramment employé pour nettoyer le code.



BerliOs possède d'autres outils tels qu'un forum dédié au projet, ainsi qu'un hébergement Web de 100 Mo par projet. Enfin, chaque compte utilisateur possède un espace de fichiers sur machine Linux accessible par connexion sécurisée SSH.

Il est très probable qu'une partie seulement de ces outils nous intéressera, mais le fait de les avoir tous au même endroit facilitera leur utilisation.

### D. Le format STL

Le format STL pour *Standard Triangulation Language* est un format standard utilisé pour stocker des maillages tridimensionnels et donc des *mesh*. Le choix de STL a été guidé par le logiciel COMSOL qui est capable de traiter ce format nativement. Le format STL existe en deux versions : ASCII et binaire.

#### 1. Version ASCII

La version ASCII (*American Standard Code for Information Interchange*) du langage est très verbeuse mais à l'avantage d'être lisible avec n'importe quel éditeur de texte. Le squelette d'un tel fichier est présenté ci-dessous.

```
solid nomDuMesh
  facet normal nx ny nz
    outer loop
      vertex v1x v1y v1z
      vertex v2x v2y v2z
      vertex v3x v3y v3z
    endloop
  endfacet
  ...
endsolid nomDuMesh
```

La variable « nomDuMesh » représente le nom du modèle généré, celui-ci est optionnel mais s'il existe, on doit le retrouver à la fin du fichier après « endsolid ». Chaque triangle du modèle est représenté par un bloc « facet » et par douze nombres flottants :

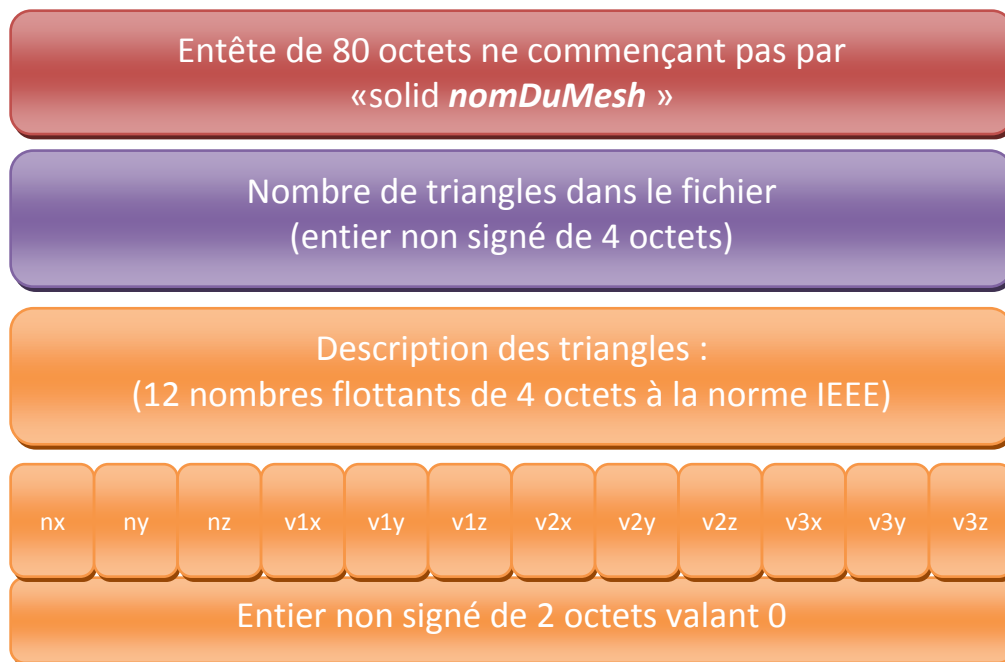
- $nx, ny, nz$  : les composantes  $x, y$  et  $z$  du vecteur normal<sup>5</sup>
- $vix, viy, viz$  : les composantes  $x, y$  et  $z$  du sommet  $i$  du triangle ( $i \in [1, \dots, 4]$ )

---

<sup>5</sup> Le vecteur normal est le vecteur unitaire perpendiculaire au triangle.

## 2. Version binaire

La version binaire du format a l'avantage d'être beaucoup plus compacte que la version ASCII. Le fichier STL au format binaire a la forme suivante :



Le fichier est donc constitué d'un entête de 80 octets. Celui-ci ne doit pas commencer comme le début d'un fichier STL ASCII car il n'y a pas d'autres distinctions entre la version ASCII et binaire du format. Après l'entête, un entier non signé de 4 octets indique combien de triangles sont décrits dans le fichier. Enfin chaque triangle est décrit à la suite par ses 12 nombres flottants de 4 octets à la norme IEEE ainsi qu'un entier non signé de 2 octets valant 0. La norme du format indique que ce dernier entier est le « *attribute byte count* », cependant il n'est pas utilisé (d'où sa valeur nulle).

Dans le cadre du projet, pendant la phase de développement c'est la version ASCII du format qui sera privilégiée dans un souci de débogage. Par la suite, il sera plus judicieux de passer au format binaire dans la mesure où celui-ci est beaucoup plus compact. De plus lors de la génération d'un fichier STL contenant des milliers de triangles, ce qui prend le plus de temps, ce n'est pas la triangulation mais l'écriture sur le disque du fichier.

## E. Les librairies

### 1. Qt

Ce projet étant avant tout un moyen de simplifier l'utilisation de logiciel tels que COMSOL ou MATLAB, et ce, via une interface graphique répondant à de nombreuses exigences, le choix d'une librairie graphique est très important et doit se baser sur un certain nombre de critères. En effet, les exigences de l'interface sont, pour l'utilisateur, principalement la simplicité, l'intuitivité et la portabilité, tandis que pour les programmeurs, les exigences sont l'accessibilité de la librairie en fonction du langage de programmation utilisé, ainsi que la possibilité de dessiner une palme de façon

interactive et de pouvoir programmer l'interface entière avec une relative simplicité. De plus, cette bibliothèque doit être gratuite et utilisable avec n'importe quel environnement de développement.

Plusieurs bibliothèques graphiques portables existent : wxWidget, GTK... Mais celle qui paraît être la plus adaptée au projet est sûrement Qt, qui possède un certain nombre d'avantages.

Tout d'abord, la question du langage de programmation (C++ ou Java) n'étant pas réglée dès le départ, il se trouve que Qt existe pour ces deux langages (Qt Jambi pour Java) et qu'il possède les mêmes fonctionnalités dans les deux cas.

De plus, cette bibliothèque est très utilisée dans de nombreuses applications libres ou commerciales : Google Earth, KDE, Skype, VLC media player, etc. Très connue, elle est donc sérieuse, très bien conçue et surtout très bien documentée, et ce, pour la version C++ comme pour la version Java. Elle est ainsi très facile d'accès pour les programmeurs grâce, notamment, à sa structure de classes solide et son code facilement manipulable.

Concernant la portabilité, Qt offre exactement ce dont le projet a besoin : par simple recompilation du code, on peut faire fonctionner le logiciel aussi bien sous Windows que sous Linux ou Mac OS, ainsi que sur des plateformes embarquées.

Mais un des plus gros avantages de Qt est certainement les outils fournis avec. En effet, en plus de la documentation complète hors-ligne fournie avec Qt très utile pour les programmeurs, un outil de développement de fenêtres graphiques existe et est très facile à utiliser. Il s'agit de QtDesigner, qui permet de créer facilement à la souris une interface graphique pour le logiciel, en restant éloigné du code (C++ ou Java), ce qui peut éviter un certain nombre de *bugs*.

Cette interface pourra ensuite être mise en relation avec les autres logiciels. En effet, Qt permet d'interagir avec d'autres logiciels grâce à des classes spécialisées. La bibliothèque Qt est donc bien adaptée puisque, mis à part l'interface, un des points importants du projet consiste à faire communiquer des applications entre elles, ce qui est permis par Qt et ses nombreuses fonctionnalités.

Pour finir, les besoins du projet sont entre autre de pouvoir afficher et traiter des images, et de dessiner de façon interactive un modèle de palme en deux dimensions. Or, dès la première version de Qt, des classes permettaient la création de dessin (points, courbes, etc.) avec récupération des coordonnées, le tout de façon optimisée. Les différentes versions de Qt (actuellement en version 4) ont amélioré toutes les possibilités de dessin graphique, et un programme donné en exemple avec la bibliothèque prouve qu'il est facile de créer une fenêtre dans laquelle un utilisateur peut venir déplacer des points pour obtenir la forme qu'il souhaite. Ces points peuvent ensuite être facilement récupérés sous forme de coordonnées, afin de pouvoir les traiter et les transformer en un objet 3D acceptable par COMSOL.

L'importation et l'affichage d'images, ainsi que de vidéos est tout à fait possible avec la bibliothèque Qt, voire même facile.

Ainsi, tous les éléments nécessaires à la création d'une interface compatible avec les exigences du projet sont présents dans la bibliothèque Qt, et l'intégration de ces éléments dans le logiciel final est également facile. C'est-à-dire que l'utilisateur n'aura aucun logiciel supplémentaire à

installer, car il suffit d'intégrer, avec le programme, des fichiers DLL contenant le code de Qt, ou bien de compiler le code en récupérant les parties de Qt dont celui-ci a besoin. Comme cette bibliothèque s'intègre facilement à un environnement de développement comme Eclipse, son utilisation est d'autant plus facile et elle se révèle donc être le meilleur choix pour ce projet.

### 2. Qhull

Qhull est une bibliothèque mathématique écrite en langage C capable entre autre de calculer la triangulation de Delaunay d'un ensemble de points de dimension  $n$  ( $n \geq 2$ ). La bibliothèque est distribuée librement avec son code source.

Cette bibliothèque a été utilisée dans le cadre de la réalisation du premier prototype permettant de modéliser un modèle 3D de monopalmier. Nous avons choisi cette bibliothèque pour sa renommée, en effet, elle est utilisée par MATLAB et Blender<sup>6</sup>, mais également parce que Qhull est distribuée sous la forme d'un programme en ligne de commande. Dans le cadre de la réalisation d'un prototype, il est beaucoup plus simple d'utiliser une ligne de commande plutôt que d'appréhender du code en langage C.

Le détail sur l'utilisation de la bibliothèque ne sera pas développé ici, dans la mesure où elle ne sera pas directement utilisée dans la suite du projet.

---

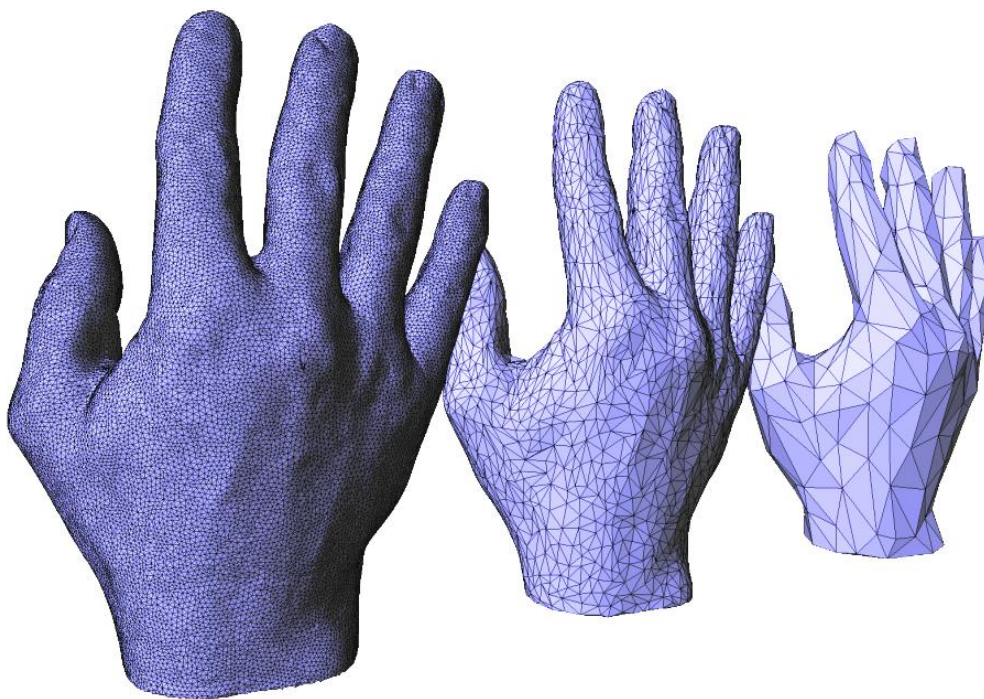
<sup>6</sup> Blender est un outil de modélisation 3D sous licence GNU General Public Licence

### 3. Computational Geometry Algorithms Library (CGAL)

La réalisation du prototype, sur l'utilisation d'une *heightmap* pour modéliser une monopalme en trois dimensions a soulevé un problème. Le maillage effectué est naïf, en effet, la *heightmap* permet d'obtenir une grille de points. Cette grille est actuellement triangulée sans optimisation. Chaque point devient le sommet d'un ou plusieurs triangles.

Actuellement la modélisation de la palme avec la *heightmap* ne pose pas de problème de performance, la génération ne prend pas plus de deux secondes pour la plus grosse image testée (cf. Heightmap page 10). Cependant s'il s'avère que COMSOL a du mal à traiter un modèle 3D qui possède trop de triangles, il nous sera alors possible de travailler à la simplification du maillage obtenu par la *heightmap*. Il existe quelques algorithmes permettant de simplifier le maillage mais ils sont assez complexes à implémenter.

CGAL est une bibliothèque graphique rassemblant de nombreux algorithmes destinés à la géométrie (dans le plan et dans l'espace). La bibliothèque est libre, réalisée en C++ et fonctionne à la fois sous Windows et sous Linux. Aucun travail de fond n'a été effectué sur cette bibliothèque pour le moment. Nous nous sommes contentés de parcourir la documentation en ligne et les exemples présentés. Néanmoins, il nous paraissait important de mentionner cette bibliothèque qui nous sera peut-être très utile à travers son algorithme de simplification de *mesh*, comme en témoigne l'image ci-dessous présentant la simplification d'un mesh (issue de la documentation en ligne de la bibliothèque).



Simplification d'un mesh à l'aide de CGAL.

## **F. Logiciels scientifiques**

Notre projet doit s'interfacer de manière transparente avec trois logiciels scientifique destinés à effectuer des calculs : COMSOL, MATLAB et MAPLE.

### **1. COMSOL**

COMSOL est un logiciel permettant de faire des simulations physiques complexes, en deux ou trois dimensions. Les simulations portent sur des modèles d'objet complexes que l'on peut par exemple importer au format STL. COMSOL peut être piloté entièrement en ligne de commande grâce à des scripts MATLAB. C'est cette capacité à fonctionner en ligne de commande qui permettra d'avoir une interface transparente entre notre projet et COMSOL. Nous utiliserons COMSOL pour simuler le mouvement tridimensionnel de la monopalme dans un milieu aquatique.

### **2. MAPLE**

Tout comme COMSOL, MAPLE est un logiciel de calcul scientifique pilotable à l'aide de scripts. Il est cependant moins adapté au calcul de simulations physiques et servira à étudier la déformation d'une section de palme (donc en deux dimensions).

### **3. MATLAB**

MATLAB est un logiciel de calcul scientifique mais également un langage de programmation. Dans le cadre de notre projet c'est le langage de programmation (plus précisément de script) MATLAB que nous serons amenés à utiliser.

## VI. Estimation de la charge initiale

Dans l'état actuel des choses et compte tenu du fait que les spécifications fonctionnelles ne sont pas encore rédigées, il est difficile d'estimer objectivement la charge de travail qu'il reste à accomplir sur tout le projet. Néanmoins, ci-dessous se trouve une première esquisse de l'estimation des charges.

- Rédaction des spécifications fonctionnelles
  - Réalisation du prototype de démonstration 20 jours
  - Rédaction des spécifications 15 jours
  - Validation des spécifications 10 jours
- Élaboration de la planification initiale 3 jours
- Rédaction du rapport de conception logicielle 8 jours
- Préparation des deux premières soutenances 3 jours
- Développement
  - Implémentation des algorithmes de reconnaissance de contours 10 jours
  - Implémentation de la génération du fichier STL 15 jours
  - Développement de l'interface utilisateur 65 jours
  - Développement des scripts d'interfaçage avec COMSOL et MAPLE 30 jours
  - Intégration 10 jours
  - Tests unitaires continu
  - Documentation continu
- Validation 10 jours
  - Recette 2 jours
  - Corrections
- Déploiement de l'application 2 jours
- Rédaction du rapport final
  - Rédaction du rapport 15 jours
  - Préparation de la soutenance finale et de la démonstration 5 jours

Cette estimation est relativement grossière et ne met pas en évidence l'ordonnancement des tâches les unes par rapport aux autres. C'est au dossier de planification initiale qu'il adviendra de présenter un premier diagramme de Gantt.

## VII. Conclusion

L'objectif de notre projet est donc la création d'un outil faisant le lien entre l'utilisateur et les logiciels COMSOL et MAPLE. Cet outil fournira une interface permettant simplement et d'une manière intuitive la modélisation d'un modèle de monopalme en trois dimensions. Dès lors la monopalme modélisée, les utilisateurs auront la possibilité de paramétrer une simulation. C'est à ce moment que notre logiciel communiquera avec les logiciels scientifiques pour lancer la simulation puis récupérer les résultats et les afficher à l'utilisateur.

Cette phase de pré-étude a permis de trouver puis de retenir des solutions relatives aux objectifs que nous nous sommes fixés :

- La création d'une interface entre l'utilisateur et notre logiciel. Cette interface sera implémentée à l'aide de la bibliothèque Qt.
- L'utilisation de divers algorithmes et bibliothèques permettant la création d'une monopalme en 3D au format STL.
- La création d'une interface transparente à l'utilisateur entre notre logiciel et les logiciels scientifiques COMSOL et MAPLE.

Néanmoins, certaines solutions n'ont pas encore données lieux à un choix définitif. C'est le cas du choix d'un algorithme de détection de contours ou encore de l'éventuelle utilisation de la librairie CGAL pour la simplification des maillages obtenus à l'aide de *heightmaps*.

La poursuite de notre projet va nous mener à la rédaction des spécifications détaillées. Pour illustrer ces spécifications et obtenir une validation des utilisateurs, un prototype de l'application va être réalisé. En parallèle nous étudierons plus en détail les possibilités d'interfaçage entre notre logiciel, COMSOL et MATLAB. Cette dernière étape passera notamment par l'étude du langage de script MATLAB qui est celui utilisé par COMSOL. Enfin des tests seront effectués pour valider tous les choix que nous avons établis durant cette pré-étude.



## VIII. Bibliographie<sup>7</sup>

(s.d.). Récupéré sur Qhull: <http://www.qhull.org>

*Active appearance model*. (s.d.). Récupéré sur [http://en.wikipedia.org/wiki/Active\\_Appearance\\_Model](http://en.wikipedia.org/wiki/Active_Appearance_Model)

*Active contour*. (s.d.). Récupéré sur [http://en.wikipedia.org/wiki/Active\\_contour](http://en.wikipedia.org/wiki/Active_contour)

*Active shape model*. (s.d.). Récupéré sur [http://en.wikipedia.org/wiki/Active\\_shape\\_model](http://en.wikipedia.org/wiki/Active_shape_model)

*Bug tracking system*. (s.d.). Récupéré sur [http://en.wikipedia.org/wiki/Bug\\_tracking\\_system](http://en.wikipedia.org/wiki/Bug_tracking_system)

Cacciola, F. (s.d.). *Triangulated Surface Mesh Simplification*. Récupéré sur [http://www.cgal.org/Manual/3.3/doc\\_html/cgal\\_manual/Surface\\_mesh\\_simplification/Chapter\\_main.html](http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/Surface_mesh_simplification/Chapter_main.html)

Cachan), T. L. (s.d.). *Bref comparatif entre Java™ et C++*. Récupéré sur <http://www.iro.umontreal.ca/~dift1020/cours/ift1020/communs/Cours/C17/comparatif.pdf>

*CGAL - Computational Geometry Algorithms Library*. (s.d.). Récupéré sur <http://www.cgal.org>

*Concurrent Version System (CVS)*. (s.d.). Récupéré sur [http://en.wikipedia.org/wiki/Concurrent\\_Versions\\_System](http://en.wikipedia.org/wiki/Concurrent_Versions_System)

*Delaunay Triangulation*. (s.d.). Récupéré sur [http://en.wikipedia.org/wiki/Delaunay\\_triangulation](http://en.wikipedia.org/wiki/Delaunay_triangulation)

*Détection de contours*. (s.d.). Récupéré sur [http://fr.wikipedia.org/wiki/Détection\\_de\\_contours](http://fr.wikipedia.org/wiki/Détection_de_contours)

*GIMP Documentation*. (s.d.). Récupéré sur <http://docs.gimp.org/>

*Heightmap*. (s.d.). Récupéré sur Wikipedia: <http://en.wikipedia.org/wiki/Heightmap>

Jean Christophe Beyler, B. I. (s.d.). *Génération de terrain et triangulation de Delaunay*. Récupéré sur Developpez.com: <http://fearyourself.developpez.com/tutoriel/jeu/Delaunay>

*Modèle de contour actif*. (s.d.). Récupéré sur [http://fr.wikipedia.org/wiki/Modèle\\_de\\_contour\\_actif](http://fr.wikipedia.org/wiki/Modèle_de_contour_actif)

*qdelaunay*. (s.d.). Récupéré sur <http://www.qhull.org/html/qdelaun.htm>

Schwartz, P. (s.d.). *Méthode des contours actifs*. Récupéré sur <http://khayyam.developpez.com/articles/algo/contours-actifs/>

Stegmann, M. B. (s.d.). *Active Appearance Models*. Récupéré sur <http://www2.imm.dtu.dk/~aam/>

*STL format*. (s.d.). Récupéré sur [http://en.wikipedia.org/wiki/STL\\_format](http://en.wikipedia.org/wiki/STL_format)

*SVN*. (s.d.). Récupéré sur [http://en.wikipedia.org/wiki/Subversion\\_\(software\)](http://en.wikipedia.org/wiki/Subversion_(software))

---

<sup>7</sup> À la date du vendredi 24 octobre 2008, tous liens Internet de la bibliographie fonctionnent.

## IX. Annexes

### A. Décompte des heures

Le tableau ci-dessous met en évidence le temps passé à la réalisation de chaque tâche liée à notre projet jusqu'à l'élaboration de ce dossier.

Tâche	Temps passé (en heure)
Réunions de groupe	90
Rédaction des comptes rendus de réunions	2
Rédaction et mise en forme du compte rendu	21,5
Recherches sur Qt	10
Recherches sur le format STL (+ réalisation des prototypes)	12
Recherches sur les algorithmes de détection de contours	4
Étude puis choix d'une plateforme de développement (BerliOS)	4
<b>Total</b>	<b>143,5</b>