École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5e année
2012 - 2013

Rapport de Projet de Fin d'études

Optimisation interactive pour la visualisation d'images médicales

Encadrants
Barthélemy SERRES
barthelemy.serres@univ-tours.fr

Etudiant
Julien TERUEL
julien.teruel@etu.univ-tours.fr

Université François-Rabelais, Tours

DI5 2012 - 2013

Version du 30 avril 2013

Table des matières

Table des figures

Liste des tableaux

Remerciements

Je tiens à remercier toute l'équipe FOVÉA pour son accueil, ainsi que plus particulièrement mes encadrants Barthélemy Serres et Gilles Venturini, pour leur aide, leurs conseils et leur constante présence sur toute la durée de mon PFE.

Je tiens également à remercier tous mes amis qui ont pris le temps de participer aux tests utilisateurs.
Je remercie particulièrement Faka, Mithrop, Mr0, Diablogore, Bous, FumistOS, Demon pour les pauses Killing Floor et Call of Duty 4 entre midi et deux.
Je remercie enfin ma concubine pour son soutien tout au long de l'année ainsi que pour son aide de relecture.

Introduction

Ce document a pour but de montrer et d'expliquer le travail réalisé dans le cadre de mon projet de fin d'étude, effectué au sein de l'équipe FOVEA du laboratoire informatique de l'université François Rabelais de Tours.

Nous rappellerons le contexte dans lequel s'inscrit celui-ci puis nous reverrons succinctement les objectifs qui étaient à atteindre. Nous verrons plus en détails le travail réalisé, les difficultés rencontrées puis nous comparerons les résultats obtenus avec les objectifs initiaux.
Une fois la partie développement expliqué et analysé nous disséquerons la partie gestion de projet, nous regarderons en détails si il y a eu des écarts entre le planning prévu et le réel, et le cas échéant nous tenterons d'expliquer ces écarts.

Nous finirons par une conclusion sur le travail réalisé.

Contexte

3.1 Notion de base

Pour être en mesure de bien comprendre le travail réalisé ainsi que le fonctionnement de l'application StéréoApp, il est nécessaire d'avoir des notions de base sur les fonctions de transfert, les histogrammes et les algorithmes génétiques.
Si ces notions vous sont déjà familières, vous pouvez directement passer à la partie StéréoApp présentant l'application, sinon nous allons revoir ces différentes notions.

3.1.1 Histogramme

Un histogramme est un graphique en bâtons, qui sert à représenter l'évolution d'une variable. Par exemple, la température moyenne au cours des mois de l'année.

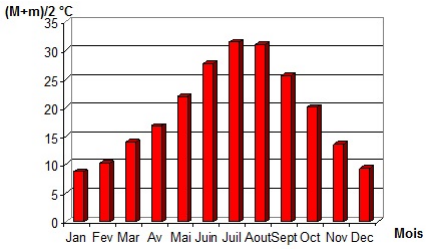


FIGURE 3.1 – Exemple d'histogramme

Le modèle 3D de la zone centrale, n'est que la superposition des images de coupes 2D que l'on peut voir dans la partie droite de l'application. Ces images de coupes sont composées seulement de variations de gris, il y a 2048 niveaux de gris.
Dans la partie basse de l'application, il y a un histogramme, celui-ci représente le nombre de pixels par niveaux de gris. Il est créé en parcourant tous les pixels de toutes les images de coupes 2D. En abscisse nous avons les 2048 niveaux de gris, et en ordonnées le nombre de pixels rencontrés correspondants.

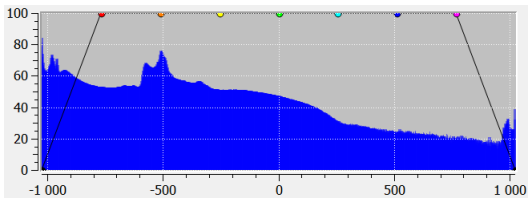


FIGURE 3.2 – Histogramme dans l'interface StéréoApp

3.1.2 Fonction de transfert

Nous allons appliquer une fonction à notre histogramme. Par défaut dans notre application, cette fonction est déterminée par neuf points de référence. Ces points sont positionnés à intervalles réguliers sur notre histogramme, placés à une hauteur aléatoire et ont chacun une couleur aléatoire.
En reliant ces points, nous obtenons une courbe, tous les voxels (équivalent du pixel, mais en 3D) de notre modèle qui correspondent à des points se situant au-dessus de la courbe ne sont pas affichés. Tous les voxels correspondants aux points situés au-dessous de la courbe, sont visibles et colorés selon leur proximité avec les points de référence de la courbe.
Cela veut dire que nous pouvons faire disparaitre des parties de notre modèle pour voir au travers. Nous pouvons donc voir les os au travers de la chair dans notre modèle, il suffit de trouver la fonction de transfert correspondante et c'est là toute la difficulté.

3.1.3 Algorithme génétique

Les algorithmes génétiques sont des algorithmes qui s'appuient sur les principes de l'évolution en biologie. Ils sont utilisés comme heuristique pour des problèmes d'optimisation, ce sont des algorithmes évolutionnistes.
Ces algorithmes travaillent sur des populations de solutions potentielles au problème. Ils effectuent une sélection d'un certain nombre d'individus puis des croisements entre les solutions potentielles sélectionnées et finissent par faire muter certains individus de la population. Ces éléments sélectionnés, croisés et mutés constituent une nouvelle population, sur laquelle on recommence l'opération.
Dans notre cas, un individu est une fonction de transfert, et notre population est un ensemble de fonctions de transfert. À chaque itération, l'utilisateur va sélectionner les images correspondantes à des fonctions de transfert, qui se rapprochent le plus du résultat qu'il recherche. Une fois sa sélection finie il va demander à ce que l'algorithme finisse l'itération. L'algorithme va prendre des individus parmi cette sélection et croiser entre elles les fonctions de transfert, ces résultats sont ajoutés dans la nouvelle population. Cette nouvelle population subit ensuite, des mutations aléatoires.
Une fois tout ce processus effectué, la nouvelle population est présentée à l'utilisateur, qui, soit trouve le résultat qu'il recherchait et s'arrête, soit ne le trouve pas et recommence une sélection d'individus s'en rapprochant et relance une itération.

3.2 StéreoApp

L'application StéréoApp permet de visualiser le résultat d'une IRM en volume 3D. L'interface principale nous permet de manipuler ce volume 3D construit à partir des coupes 2D issues du scanner. Nous pouvons visualiser et naviguer entre les coupes 2D grâce à un widget sur la partie droite que l'on appellera SliceViewer par la suite. Un histogramme de ce volume est disponible en dessous de sa représentation. Sur cet histogramme une fonction de transfert est représentée. Elle est modifiable directement à la souris via les différents points qui la composent. Le volume représenté dans l'interface principale est le résultat de la fonction de transfert appliquée aux coupes.

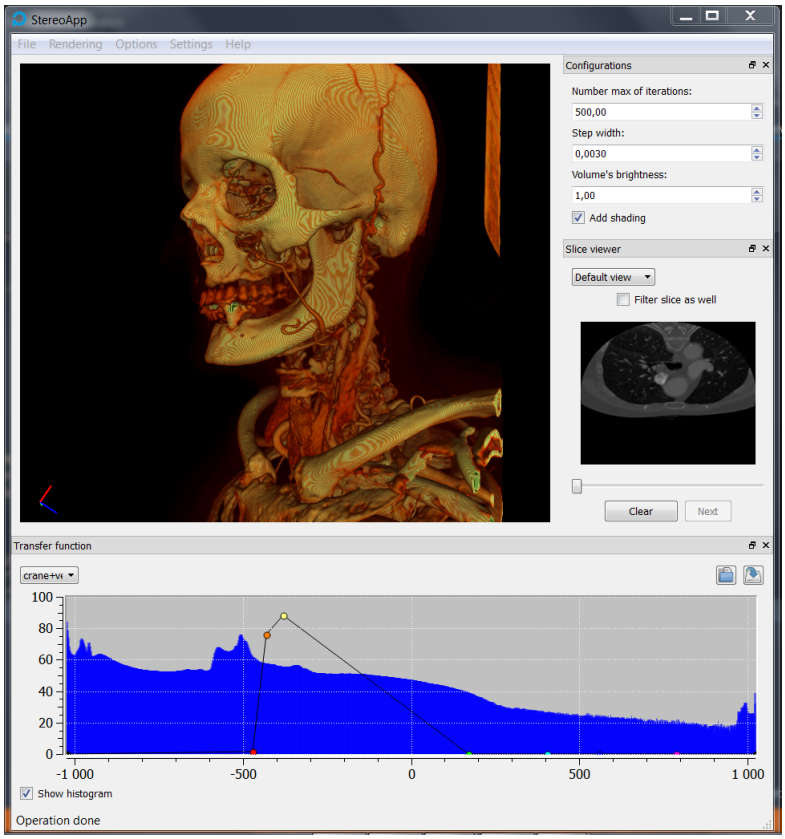


FIGURE 3.3 – Interface principale StéréoApp

L'équipe FOVEA aimerait qu'au travers de l'application, les médecins puissent exploiter tout le potentiel des IRM grâce à l'utilisation de fonctions de transferts sur l'histogramme du volume 3D. Cependant un problème s'est vite posé, la configuration de la fonction de transfert à la souris est complexe et fastidieuse. Il n'est pas aisé de cibler des éléments très précis. Pour remédier à ce problème, l'équipe a mis en place un algorithme génétique pour accélérer le processus de configuration de la fonction de transfert. L'utilisateur sélectionne des représentations 2D se rapprochant du résultat qu'il souhaite avoir, et relance une itération de l'algorithme génétique ayant comme parent la sélection. L'utilisateur recommence ce processus jusqu'à obtenir le résultat souhaité.
Toute la difficulté repose sur la présentation des individus générés par l'algorithme et dans la gestion des sélections. Plus on affiche d'individus, plus on a de chance de converger vite vers le résultat, mais on peut aussi finir par noyer d'information l'utilisateur, ou avoir des temps de calcul trop longs entre chaque itération. Il faut donc choisir une interface pertinente et bien pensée pour guider l'utilisateur au cours des itérations de l'algorithme génétique tout en le soumettant à des temps d'attente acceptables.

3.3 Historique

Mon PFE fait suite à trois autres PFE ainsi que des stages de fin d'études, autant dire que le projet est conséquent.

Le premier PFE de Nicolas Nathan était basé sur la création du modèle 3D à partir des images de coupes 2D (méthode DICOM), ainsi que sur la mise en place de l'histogramme et des fonctions de transfert.
Tandis que le second PFE de Sylvain Botto était basé sur la conception d'une interface pour guider l'utilisateur dans les itérations de l'algorithme génétique. Cette première version d'interface interactive, affichait neuf individus et nous permettait de les sélectionner, de modifier les paramètres de l'algorithme et de passer à l'itération suivante. Seulement en affichant neuf individus, il n'y avait peu de chance de générer un individu intéressant rapidement. L'année suivante un autre PFE a eu pour objet de mettre en place une nouvelle interface, Mélodie Roger a proposé d'afficher beaucoup plus d'individus et de les classer selon leur luminance et leur couleur. L'interface souffre malheureusement de lenteur et a pour inconvénient principal de superposer des individus. La sélection d'individus d'itération en itération n'est pas non plus aisée.
Un nouveau PFE pour la mise en place d'une nouvelle interface a été demandé cette année encore, ce rapport en est la synthèse.

Objectifs

Ce PFE a deux objectifs distincts l'un largement plus prioritaire que l'autre. Nous parlerons dans un premier temps du moins important, pour finir par l'objectif qui est au cœur de ce PFE, à savoir l'interface d'interaction avec l'algorithme génétique.
Pour une présentation plus détaillée des objectifs à atteindre et des développements correspondants, je vous invite à lire le cahier de spécifications lié à ce PFE.

4.1 Sélection restrictive de l'histogramme

Nous avons vu que le volume 3D présenté au cœur de l'interface principale était le résultat de l'assemblage de coupes 2D tiré d'un scan IRM. Nous avons la possibilité de naviguer entre chaque coupes. Mr Venturini a jugé qu'il serait intéressant de pouvoir sélectionner certaines zones sur les coupes.
Chacune de ces zones seraient analysées et nous ferions ressortir sur l'histogramme les valeurs de gris correspondantes à ces zones.
De plus, nous pourrions montrer dans l'interface principale un volume « filtré » où toutes les valeurs de gris non présentes dans les sélections sur les coupes seraient transparentes.

L'utilisateur aurait bien sûr la possibilité d'effacer toutes les sélections d'un coup, de naviguer directement entre les sélections sans avoir à naviguer entre toutes les coupes et de faire des sélections multiples sur différentes coupes.

Tout cela dans le but de pouvoir cibler directement un élément très précis dans les coupes 2D et de voir les parties de l'histogramme correspondantes, et donc placer plus vite et précisément les points de la fonction de transfert dans des zones d'intérêt.

Travail à réaliser pour remplir cet objectif :
- La sélection de zones dans le SliceViewer
- Le support de sélections multiples sur la même slice et entre différentes slices
- La navigation entre les slices portant une sélection
- Le calcul des niveaux de gris sélectionnés
- La gestion du double affichage de l'histogramme
- La gestion du filtrage du volume
- La gestion de la remise en état de l'histogramme

4.2 Visualisation interactive par regroupement

Comme vu dans la partie historique, il y a déjà eu deux tentatives d'interface pour permettre une interaction facile avec l'algorithme génétique. L'interface de Mélodie a l'avantage de présenter beaucoup d'individus à l'utilisateur, cependant ceux-ci peuvent se superposer.

Le but est donc de proposer une interface permettant de présenter beaucoup d'individus sans qu'il ne puisse y avoir de superposition. La sélection des individus doit être claire et rapide. Et tout comme dans les interfaces précédentes, la nouvelle devra permettre la visualisation de l'individu sélectionné dans l'interface

principale.

Nous avons donc eu l'idée de mettre en place une interface gérant des groupes d'individus (regroupés selon leur ressemblance). Ces groupes pourraient être ouverts et présentés sous une autre forme pour permettre la sélection dans un groupe. Les individus et groupes pourront être sélectionnés comme des dossiers et fichiers Windows, supportant la sélection multiple et le drag and drop.

Avec une telle interface, nous pourrions proposer un grand nombre d'individus tout en gardant un affichage clair.

Travail à réaliser pour remplir cet objectif :
- Le calcul de la distance entre deux images
- Le regroupement d'individus ressemblants
- La logique d'affichage (zoomé/dézoomé)
- L'interface zoomée
- La navigation dans un groupe
- La gestion de la sélection d'éléments (individus/groupes)
- L'intégration des fonctions de l'algorithme génétique déjà présent
- L'intégration du wizard de chargement de fonction de transfert

Développement

5.1 Environnement de développement

StéréoApp est dépendant de beaucoup d'autres librairies et d'un environnement spécifique. StéréoApp était développé sous Visual Studio 2008, avec le Framework Qt en version 4.8.1. En plus de cela, il lui fallait les bibliothèques Insight Toolkit (ITK), QWT, NVidia Cuda Toolkit et DICOMREADER. Pour ces quatre dernières, il est nécessaire de les configurer avec CMake et de les recompiler en version Debug et Release, pour pouvoir les utiliser dans notre application.

Ayant eu l'intention dès le départ de travailler mon PFE sur mon ordinateur personnel j'ai voulu migrer l'environnement de développement sur mon poste de travail, par la suite cela s'est avéré payant puisque la migration de celui-ci sur une autre machine de l'école m'a été demandée. Cela me prit un peu plus de trois semaines au début de mon PFE. Le temps de retrouver toutes les dépendances et de mettre à jour les versions des différentes bibliothèques tout en rendant cela compatible avec Visual Studio 2012. N'étant pas familier avec CMake ainsi que la recompilation de bibliothèque pour les intégrer dans un projet Visual Studio, cela me prit beaucoup plus de temps que je l'aurais voulu.

Visual Studio 2012 venant juste de sortir au moment de la migration, aucun outil ne proposait de compatibilité facile. Il m'a donc fallu modifier à la main le fichier de configuration du projet (stereoapp.vcxproj) pour arriver à compiler le projet. Il a aussi été nécessaire d'installer Visual Studio 2010, pour avoir accès au NVidia Cuda Runtime Api rules, nécessaire à la compilation des fichiers Cuda. La compilation de fichier Cuda, fait appel à un outil de Visual Studio « cl.exe ». Visual Studio 2012 étant trop récent son « cl.exe » n'était pas reconnu compatible et la compilation échouait, heureusement il suffit de rajouter le chemin vers la version 2010 dans le path et le tout compile.

Un autre problème est apparu lors de la compilation de la bibliothèque ITK, CMake génère automatiquement un fichier de projet Visual Studio *. vcxproj. Dans ce fichier, il y a principalement des chemins vers d'autres *. vcxproj représentant chacun un module de la bibliothèque. Ces chemins peuvent s'enfoncer loin dans l'arborescence de dossiers qui compose la bibliothèque, tellement qu'au bout d'un nombre limité de caractères (260) le chemin était tronqué. Il a donc fallu venir compléter le chemin manuellement. Cette erreur a été particulièrement difficile à trouver, car le compilateur de Visual ne nous indiquait qu'un échec d'ouverture de projet, alors que toutes les étapes précédentes étaient faites avec succès. Il a fallu remonter jusqu'à la première étape avec CMake.

Pour que cela soit plus facile par la suite j'ai rédigé un document que vous pouvez trouver en annexe, indiquant la marche à suivre pour migrer l'environnement de développement sur une autre machine.

Le bénéfice direct de cette migration fût la mise à jour de certaines bibliothèques, nous sommes passés de la version 4.8.1 de Qt à la version 4.8.4. Nous sommes également passés de la version 3.20 d'ITK à la version 4.2. QWT reste en version 5.2.3, le saut vers la version 6 requiert trop de changements bas niveau dans l'application.
Ces mises à jour rendant l'application finale toujours plus stable et rapide.

FIGURE 5.1 – Technologies utilisées pour le développement du logiciel

5.2 Sélection restrictive de l'histogramme

L'implémentation de cette fonctionnalité s'est effectuée en plusieurs étapes. Nous verrons dans un premier temps la problématique de sélection d'une zone dans le SliceViewer, puis nous aborderons la création et le stockage de l'histogramme correspondant aux niveaux de gris des zones sélectionnées. Nous enchainerons sur la mise en place de l'affichage double dans l'interface principale, pour l'histogramme ainsi que sur le filtrage et la sauvegarde du volume 3D. Nous analyserons les problèmes particuliers rencontrés, puis comparerons le résultat avec l'objectif qui était fixé et nous conclurons avec une réflexion sur les évolutions possibles à apporter.

5.2.1 La sélection

Avant toutes choses, une analyse en amont avait été menée pour trouver les classes correspondantes aux éléments qui nous intéressent.
Le widget Qt affiché à l'écran que nous appelons SliceViewer correspond à la classe GLSliceViewer, ce widget affiche une zone de rendue OpenGL, cette zone de rendu correspond à la classe GLTextureView.

Maintenant que nous avons situé les classes qui nous intéressent pour la problématique de sélection, regardons-les de plus près. La classe GLSliceViewer sert uniquement à afficher la zone OpenGL qu,i elle, contient toutes les informations qu'il nous faut. Nous avons décidé de ne gérer que les sélections de formes rectangulaires sur cette surface OpenGL.
Mais comment faire une sélection ? Qu'est-ce que nous appelons une sélection sur cette surface ? Une sélection se décompose en une série d'actions effectuées par l'utilisateur avec la souris. L'utilisateur va cliquer dans la zone OpenGL, rester cliqué et va glisser la souris jusqu'à l'endroit qui l'intéresse puis relâcher son clic. Pendant le temps où l'utilisateur reste cliqué et déplace sa souris, un rectangle semi-transparent sera dessiné, partant de l'endroit exact cliqué au départ et allant jusqu'à la position actuelle de la souris.
Pour réaliser l'action de sélection décrite ci-dessus, nous devons récupérer plusieurs évènements de notre zone OpenGL :

- Bouton de souris appuyé
- Bouton de souris relâché
- Souris bouge pendant un clic

La classe GLTextureViewer hérite de QGLWidget qui hérite lui-même de QWidget, cela nous permet de récupérer les évènements Qt : QMousePressedEvent

- QMousePressEvent
- QMouseReleaseEvent
- QMouseMoveEvent

Ces évènements sont déclenchés, respectivement par, un appui sur un bouton de la souris, un bouton de souris qui était appuyé et qui vient d'être relâché, et finalement un mouvement de la souris.

À partir de tout cela, l'implémentation est plutôt directe, lorsque l'on détecte un clic gauche de souris on passe un booléen à vrai (selectionInProgress) indiquant qu'une sélection est en cours, et nous sauvegardons la position de l'endroit cliqué (selectionStart).
Lors d'un mouvement, nous sauvegardons la position de la souris (selectionEnd). Le booléen étant passé à vrai, lorsque la scène est redessinée dans la fonction paintGL nous dessinons un rectangle semi-transparent en OpenGL partant de selectionStart et allant à selectionEnd.
Lorsque le clic est relâché le booléen selectionInProgress repasse à faux et l'on nettoie les points selectionStart et selectionEnd.

FIGURE 5.2 – Le SliceViewer avant et après sélection

5.2.2 Le stockage

La sélection étant maintenant terminée, il nous faut correctement stocker plusieurs informations. Pour pouvoir facilement redessiner le rectangle, nous avons besoin de connaître les coordonnées de son point haut gauche et celle de son point bas droit. En plus de cela, il ne faut pas perdre de vue que la zone d'affichage permet de naviguer entre toutes les coupes de l'IRM. Il faut donc que chaque rectangle soit associé à un indice de coupes. Pour cela, nous avons rajouté un indice currentTextureOffset, que nous tenons à jour.
Pour pouvoir le tenir à jour, nous avons dû aller dans la classe GLSliceViewer, qui gère le slider permettant la navigation entre les différentes coupes. À chaque fois que la valeur de celui-ci change une méthode est appelée, elle calcule un offset de coupe et récupère les données liées à cette coupe puis les envoie à la zone d'affichage OpenGL. Nous avons récupéré cet offset et le mettons à jour à chaque fois que celui-ci change.

Nous savons que chaque coupe possède un indice unique, et que pour chaque coupe nous pouvons avoir plusieurs rectangles associés. Nous avons choisi de stocker tous les rectangles sous une forme de liste. Pour associer à chaque indice de coupe la liste des rectangles associés, nous avons choisi une QMap.

	Key lookup		Insertion	
	Average	Worst case	Average	Worst case
QMap<Key, T>	O(log *n*)	O(log *n*)	O(log *n*)	O(log *n*)
QMultiMap<Key, T>	O(log *n*)	O(log *n*)	O(log *n*)	O(log *n*)
QHash<Key, T>	Amort. O(1)	O(*n*)	Amort. O(1)	O(*n*)
QSet<Key>	Amort. O(1)	O(*n*)	Amort. O(1)	O(*n*)

FIGURE 5.3 – Tableau des complexités des algorithmes de base d'une QMap

Comme nous pouvons le voir sur ce tableau, le temps de recherche d'une clé est faible.
Nous avons donc un stockage de la forme QMap< unsigned int, QList<QRect> >.

5.2.3 Analyse

Maintenant que nous savons comment stocker les rectangles de sélections, nous devons analyser les valeurs de gris des pixels se situant sous nos rectangles de sélections.
Ce processus pourrait paraitre coûteux en temps, car nous allons parcourir plusieurs sous-sélections de plusieurs images et ce pixel par pixel. De plus, sachant que toutes les données des images sont dans le même tableau 1D à la suite, tableau affiché en image par OpenGL, la parallélisation n'apporterait aucun

gain dans notre cas.
Mais le tableau étant déjà chargé en mémoire, et les sous-sélections étant de faibles tailles sur de petites images, le processus met un temps acceptable à analyser et afficher le double histogramme.

Pour l'analyse en elle-même, nous mettons en correspondance les valeurs de gris rencontrées avec le nombre de fois que nous l'avons rencontré. Pour associer facilement et rapidement ces données, nous avons une nouvelle fois recours à une QMap pour stocker les données analysées.
Cette QMap n'est vidée que lorsque l'utilisateur demande explicitement la suppression de toutes ses sélections antérieures, cela afin de garder un effet cumulatif entre les différentes sélections.

Mais le travail ne s'arrête pas là pour l'analyse. En effet, si une valeur de gris est présente une seule fois dans notre sélection nous allons l'afficher comme une valeur intéressante sur l'histogramme, alors que celle-ci ne peut être que du bruit sur l'image. Il a donc été nécessaire de ne garder que les valeurs de gris les mieux représentées dans notre sélection. Expérimentalement, nous avons fixé ce seuil à quarante deux pourcent , c'est-à-dire que sur une population de cent valeurs de gris différentes, nous ne gardons que les quarante-deux valeurs les plus représentées dans la population.

Sur cette partie nous avons découvert un défaut lors d'un test qui est intéressant à mon avis. Jusqu'à présent, nous avons détecté une sélection à la souris, dessiné un rectangle de sélection vert, puis analysé l'image. Seul problème au moment de l'analyse le rectangle est déjà dessiné et sa couleur fausse la lecture des valeurs de gris de l'image. Suite à cette découverte, nous avons changé notre processus, maintenant lors d'une sélection nous analysons la partie sélectionnée puis nous dessinons le rectangle de sélection.

5.2.4 Double Affichage

Suite à l'analyse des niveaux de gris et le tri de celle-ci nous passons maintenant à l'étape suivante : l'affichage. Nous devons faire apparaître l'histogramme de nos sélections cumulées. Nous avons choisi de mettre en semi-transparence l'ancien histogramme (l'histogramme de toutes les surfaces de toutes les images), et de rajouter un second histogramme opaque, celui de nos sélections.

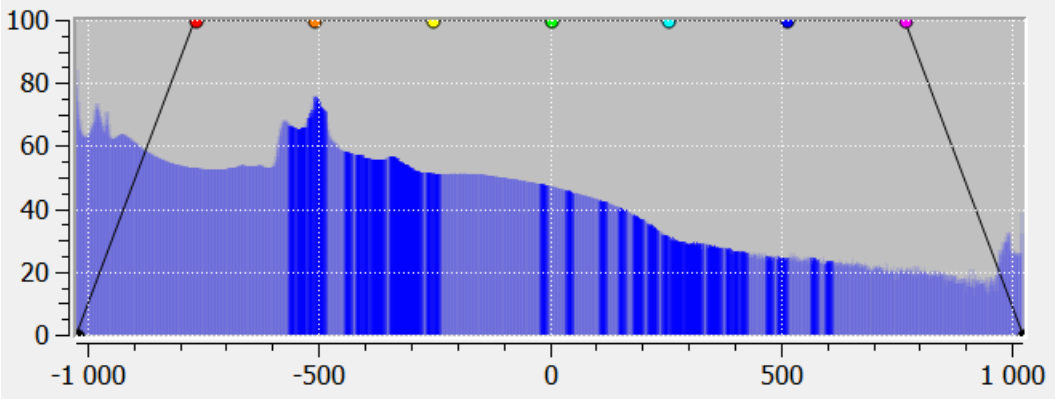


FIGURE 5.4 – Double affichage de l'histogramme en semi-transparence

Pour cela, nous avons créé une classe ManageSelectionHisto qui est un singleton, qui vient directement interagir avec le widget représentant les histogrammes qui est une instance de la classe FunctionTransfertEditor. Notre singleton contient aussi toutes les informations recueillies dans les étapes ci-dessus, afin de pouvoir les passer au FunctionTransfertEditor.

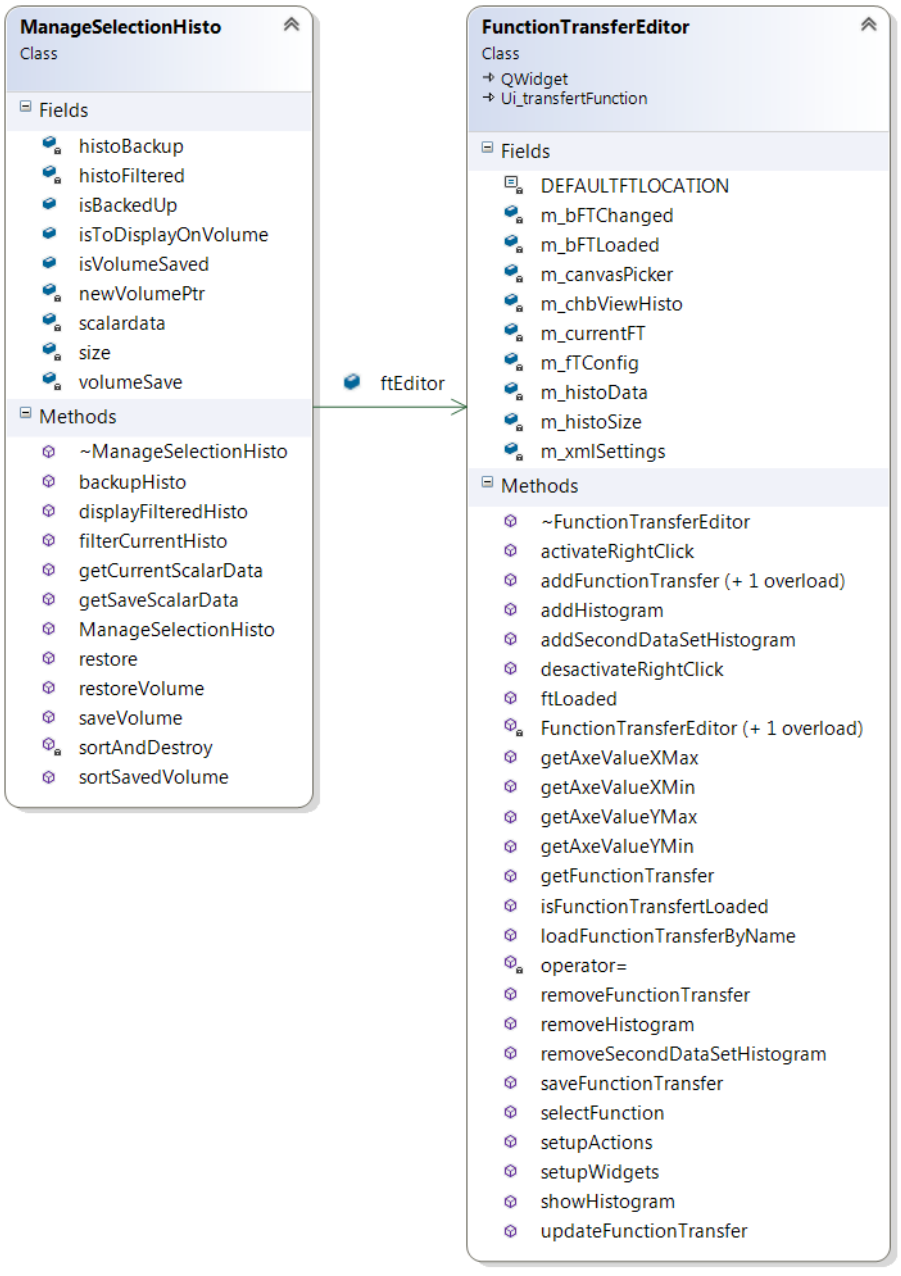


FIGURE 5.5 – Diagramme UML de la classe ManageSelectionHisto

Lorsque le tri des données de l'histogramme de sélection est fini, le singleton rajoute un second volume de données au widget et lui demande de se redessiner. Quand le widget se redessine, il interroge le singleton pour savoir si une sélection restrictive est en cours, et si oui il dessine le premier jeu de données correspondant à l'ancien histogramme avec une transparence, puis dessine le second jeu de données de la sélection restrictive de façon opaque. Dans le cas où le singleton indique au widget qu'il n'y a pas de sélection en cours, celui-ci ne dessine que le premier jeu de données de façon opaque.

Lorsque l'utilisateur demande la suppression de toutes ses sélections, le singleton vient nettoyer le deuxième jeu de données pour éviter toute utilisation de mémoire inutile.

5.2.5 Filtrage et échange de volume

Nous gérons maintenant la sélection et le double affichage de l'histogramme. Il ne nous reste plus qu'à répercuter la sélection restrictive sur le volume. Contrairement au double histogramme où l'on affichait les deux éléments en même temps, il va falloir cacher l'ancien volume, le remplacer par le nouveau volume « filtré ». Nous devons aussi, avoir la possibilité de refaire rapidement l'échange.
Nous ne pouvons pas nous permettre de recharger le volume à chaque fois, il nous faudra donc le dupliquer.

Recherche des données dans le code

La première difficulté rencontrée fût de trouver où étaient exactement les données correspondant au volume 3D affiché. En effet, plusieurs classes portent des noms explicites qui pourraient correspondre, mais le tout est masqué par des typedefs et des QSharedPointers.

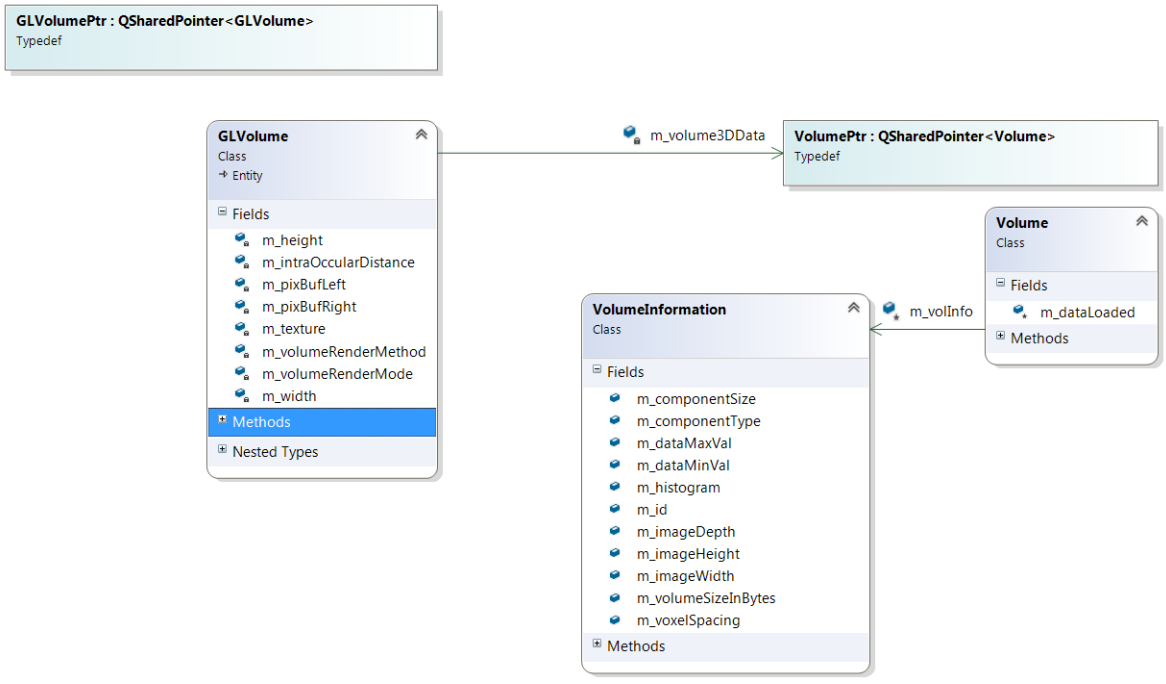


FIGURE 5.6 – Diagramme UML des QSharedPointer

Nous pouvons voir sur ce diagramme UML, que nous partons d'un GLVolumePtr qui est en fait un typedef d'un smarPointer Qt sur un GLVolume, qui est une classe possédant un attribut $m_volume3DData$ qui est un VolumePtr, qui suivant la même logique est un smartPointer Qt sur la classe Volume.
Cette classe est très importante et ambigüe, en effet cette classe possède un attribut $m_volInfo$ qui nous donne toutes sortes d'informations sur le volume dont deux très importantes : la taille en byte du volume

et le type d'encodage du volume. Et c'est avec ces informations que nous pouvons réinterpréter le type de la classe volume pour en faire, par exemple, un VolumeByte8. Une fois réinterprété, nous pouvons accéder aux données brutes du volume.
Cela pris beaucoup de temps de remonter jusqu'au fait de réinterpréter la classe Volume, aucune indication ne laisse à penser que cet objet est plus que ce qu'il n'y parait. C'est en étudiant le code existant lors du chargement d'un volume que nous avons pu retracer et analyser les différentes étapes qui nous ont conduit au dynamic cast.

Les QSharedPointers

Nous avons pu voir que les données du volume se cachaient derrière plusieurs QSharedPointer qui sont des smartPointers au niveau de Qt. Mais qu'est-ce qu'un smartPointer en C++ ? C'est un pointeur qui garde un compteur de référence sur l'élément qu'il englobe. Dès qu'un autre pointeur fait référence à l'élément englobé le compteur du smartPointer est incrémenté, et inversement. Quand le compteur atteint zéro ou si le dernier smartPointer est hors de portée, l'élément englobé est libéré. Ceci dans le but de mieux gérer la mémoire et donc la prévention de segmentation fault.
L'utilisation des smartPointers est délicate, et lors de la mise à jour de Qt en version 4.8, des messages d'alertes préventifs sur leurs mauvaises utilisations ont été rajoutés.
Lors du lancement d'une des deux anciennes interfaces, nous avons obtenu un de ces nouveaux messages : « QSharedPointer : pointer 0x2384d70 already has reference counting ». Après beaucoup de recherches nous avons trouvé un article intitulé « (Not so much) Fun with QSharedPointer » sur le blog d'Evan Teran, dans lequel il explique en détails et par l'exemple l'apparition de cette erreur, ainsi que comment la résoudre.
Pour résumer, le dernier smartPointer possède un risque d'être libéré avant l'élément en mémoire qu'il gère, ce qui peut conduire à une faute de segmentation. Pour remédier à ce problème, il suffit de gérer manuellement la libération du smartPointer concerné. Lors de la création du smartPointer, nous ne lui mettons donc pas de parent, il nous incombe alors de spécifier sa libération.

Sauvegarde et tri

Nous avons profité du fait que le volume est présent dans la classe GLSliceViewer dès son chargement, pour le stocker dès que celui change tout en prenant soin de libérer l'ancien. Pour la copie en elle-même, nous utilisons un memcpy classique.
Dès qu'une sélection est effectuée, le volume actuellement affiché est « filtré », nous mettons toutes ses valeurs de gris qui ne sont pas dans l'histogramme des sélections à transparentes. Pour cela, nous parcourons tous les pixels de toutes les images représentant ce volume. Cette opération prenant beaucoup de temps, nous avons subdivisé le tableau 1D du volume en quatre sous-tableaux, et lancé un thread de filtrage par sous-tableau. Ce découpage nous fait gagner un temps suffisant pour que celui-ci soit acceptable par l'utilisateur.

Un problème de mémoire

Notre application tourne sous système Windows, ce qui veut dire que notre application dispose de base de 2Go de mémoire RAM pour tourner. L'application n'avait aucun problème avec cette limite jusque-là, seulement nous dupliquons une énorme quantité de mémoire (700Mo pour le fichier de test du fémur). Ce dépassement cause l'arrêt pur et simple de l'application au moment de la duplication. Une solution pourrait être de ne pas sauvegarder en mémoire RAM le volume et de le recharger plus tard. Seulement le temps de chargement du volume en RAM est trop important pour que l'utilisation de cette nouvelle fonctionnalité reste confortable à l'utilisation.
Il s'avère qu'il existe une option de compilation dans Visual Studio, qui spécifie que notre programme va utiliser plus de 2 Go de mémoire RAM. Cette option n'a rien à voir avec les quantités de mémoire qu'un OS peut adresser (32bits VS 64bits), elle permet juste d'autoriser un programme à utiliser des adresses de

pointeur d'un plus grand type, pour pouvoir adresser plus de 2 Go de RAM.
Cette option se trouve dans les options du linker, rubrique system, il suffit d'autoriser/LARGEADDRESSAWARE.

Navigation

L'utilisateur peut sélectionner différentes zones dans différentes coupes. Mais le nombre de coupes étant important et la zone réservée au slider (permettant la navigation entre les coupes) étant petite, il devient très difficile de retrouver une coupe sur laquelle nous avons fait une sélection auparavant.
Nous avons donc mis en place un système de navigation, nous avons rajouté un bouton « next » sous le slider permettant de passer à la coupe suivante comportant une sélection.
Le bouton est désactivé au lancement de l'application, il devient actif dès que la QMap contenant les id des coupes associées aux listes de leurs rectangles respectifs a une taille strictement supérieure à 1.
Lorsque le bouton suivant est appuyé, nous cherchons l'id le plus proche supérieur à celui que nous avons dans la liste, et nous envoyons un faux signal au slider déjà en place pour qu'il se mette sur la bonne image. Cela nous permet de profiter de toute la logique d'extraction de l'image à partir de son id déjà présent sur le mouvement du slider.

Analyse du résultat

Voici en image le résultat final qui récapitule tout ce que nous avons pu voir ci-dessus, à savoir : la sélection restrictive sur les coupes, le double affichage de l'histogramme et le filtrage du volume principal.

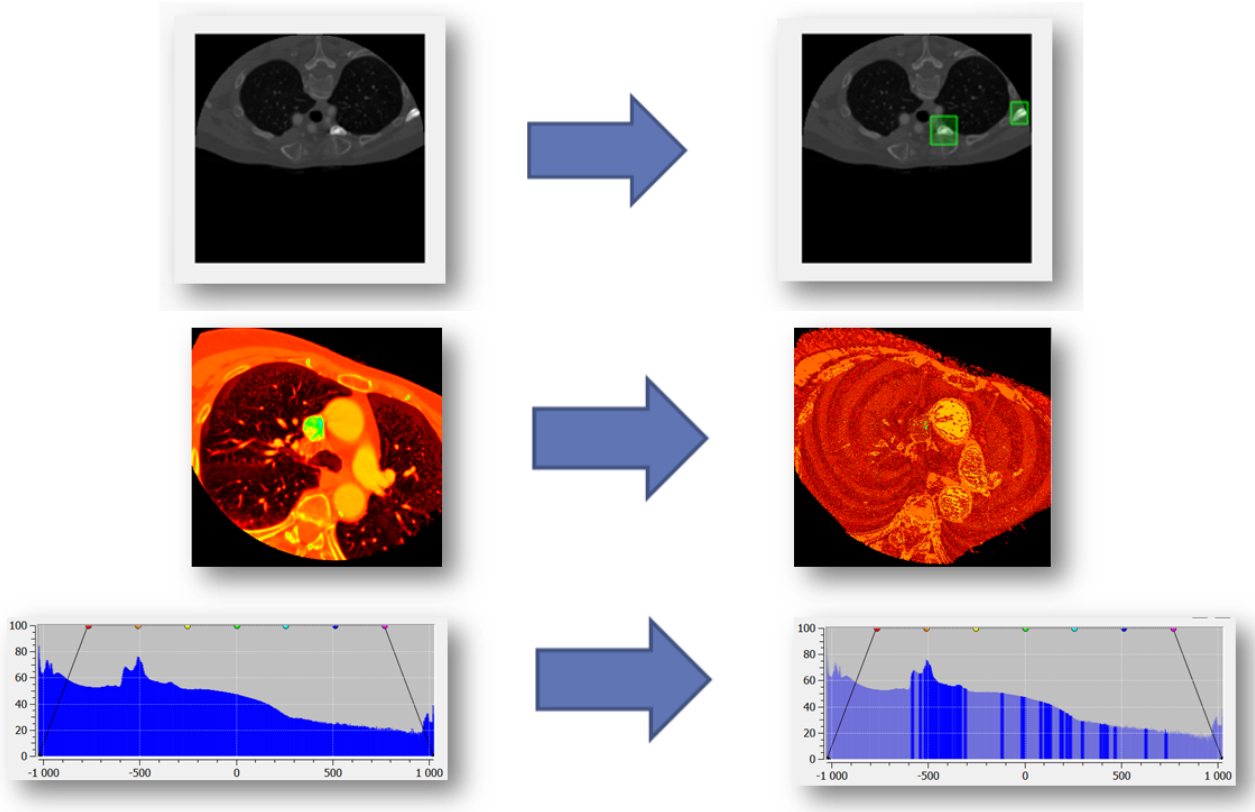


FIGURE 5.7 – Résultat de la sélection restrictive

Sur la gauche de l'image, nous voyons l'affichage sans sélection restrictive, et sur la droite nous voyons l'affichage après une double sélection sur la coupe.
Le résultat est conforme aux attentes. Toutes les fonctionnalités sont présentes et utilisables.

5.2.6 Évolutions possibles

La mise en place de la sélection restrictive était plus un test de faisabilité qu'un ajout final dans l'application. La sélection uniquement par rectangle sur une coupe de petite taille est vraiment basique. Il serait intéressant de proposer toute une interface secondaire permettant à l'utilisateur de vraiment gérer ses sélections, un peu comme dans un outil d'édition d'image, proposer diverses formes de sélection, pouvoir fusionner plusieurs sélections, retirer une zone, un zoom, etc.
On pourrait aussi imaginer que chaque bout de sélection sur la coupe soit de couleurs différentes personnalisables par l'utilisateur. Cette même couleur se retrouverait sur le double affichage de l'histogramme, pour clairement indiquer quelles parties de l'histogramme correspondent à quelles sélections. Par contre il peut y avoir un effet secondaire gênant, si la sélection est trop restrictive le modèle 3D est extrêmement dégradé, au point que cela empêche d'avoir une bonne visibilité. Il faudrait prévoir une case à cocher pour permettre à l'utilisateur de désactiver le filtrage du modèle 3D.

5.2.7 Conclusion

La sélection restrictive est une idée qui me semble pertinente. Le fait de pouvoir restreindre à des zones précises l'histogramme permets de cibler plus facilement les zones d'intérêt. Cela nous permets de calibrer plus facilement la fonction de transfert. Je pense que cette partie peut être la clé de voute de l'application il serait intéressant de restreindre l'algorithme génétique aux parties sélectionnées par l'utilisateur.

5.3 Visualisation par regroupement

L'implémentation de cette nouvelle interface est le cœur de ce PFE. Nous allons donc voir en détail chaque étape de la création de celle-ci.
Cette interface doit supporter l'affichage de nombreux individus, permettre une sélection facile des individus et offrir de bons temps de réponse. Nous sommes partis sur l'idée d'avoir une zone centrale dans laquelle on viendrait glisser les individus ou groupes intéressants. Les individus et groupes seraient disposés autour de cette zone centrale séparée d'une distance représentative de l'écart entre les images les représentant.
Nous parlerons dans un premier temps uniquement d'individus, nous verrons que pour les premiers points abordés la logique reste la même que ce soient des individus ou des groupes.

5.3.1 Mise en place de la scène

Pour créer la nouvelle interface, nous sommes partis d'une feuille blanche. Nous avons choisi d'utiliser les composants du Graphics View Framework de Qt, celui-ci proposant une approche modèle/vue pour gérer un très grand nombre d'éléments.

Notre fenêtre affichant notre interface est une WeatherWindow, elle hérite de QDialog et nous sert juste de container. Nous utilisons une QGraphicsScene, celle-ci contient tous les éléments à afficher, et est d'une taille infinie, la scène est invisible à l'écran. Nous devons utiliser une QGraphicsView pour montrer à l'écran une sous partie de notre scène. Par habitude et pour que cela soit plus facile par la suite, nous appellerons par la suite scene notre QGraphicsScene et view notre QGraphicsView.
Nous allons afficher des individus et des groupes sur notre scene. Pour l'un comme pour l'autre, nous aurons besoin de plusieurs informations communes aux deux, comme une image ou une fonction de transfert. Nous avons donc choisi de stocker toutes les informations communes dans une classe WeatherItem. Nos individus (WeatherIndividual) et groupes (WeatherGroup) héritent de la classe WeatherItem.

Pour la zone centrale, nous avons essayé plusieurs formes, en commençant avec un cercle, cette représentation nous faisait perdre beaucoup d'espace dans les angles et n'était pas très adaptée aux écrans 16 :9. Nous avons choisi un rectangle à bords arrondis, cela garde un certain aspect du cercle tout en ayant la souplesse du rectangle en largeur.

Pour pouvoir positionner facilement les éléments autour de la zone centrale, nous utilisons un QPainterPath. Celui-ci nous donne la possibilité de créer un chemin de forme quelconque et de placer les éléments relativement à ce chemin, le tout en pourcentage. Le début du chemin étant zéro et la fin du chemin étant un. Nous avons placé un chemin plus grand englobant notre zone centrale et rendu celui-ci invisible.
La classe WeatherButtonPixmapItem fut créée spécialement pour notre interface. Celle-ci représente une image qui, lorsqu'elle est survolée par la souris, double de surface. Elle nous sert pour les boutons de la vue intragroupe, mais aussi pour le bouton central de génération.

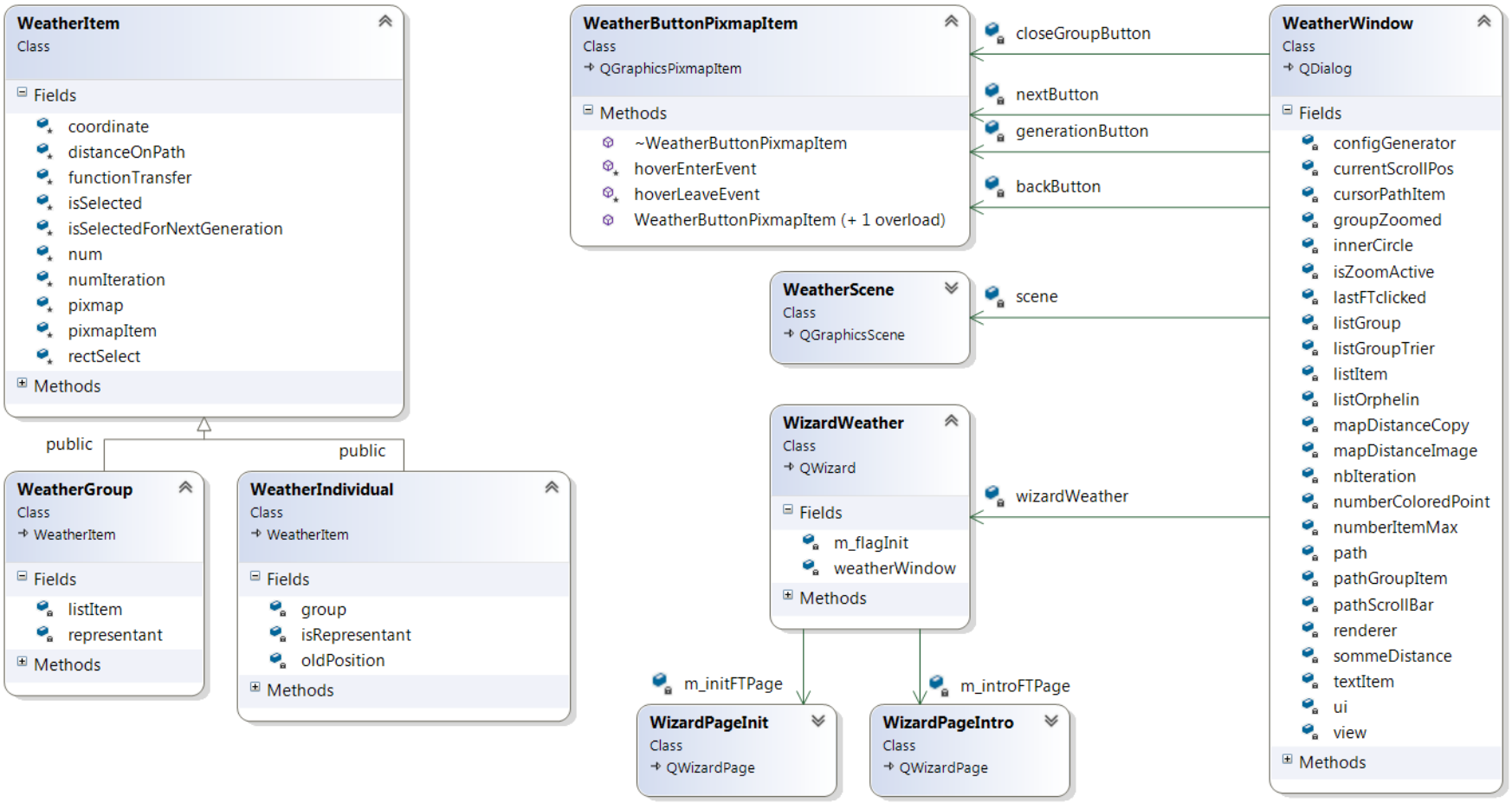


FIGURE 5.8 – Schéma UML de l'architecture de la visualisation par regroupement

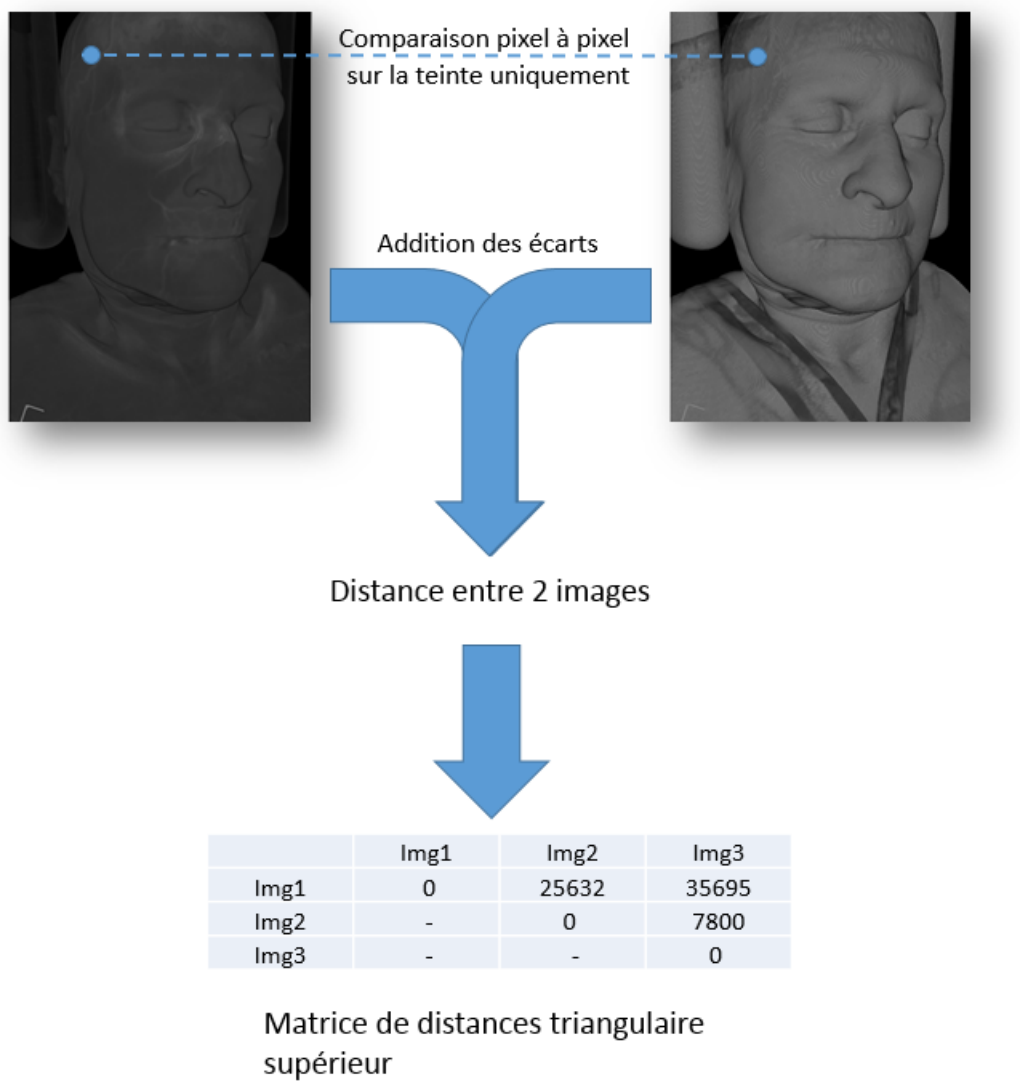
5.3.2 Génération des individus

Qu'appelons-nous individu ? Un individu, dans notre interface, correspond à une fonction de transfert. En effet, une fonction de transfert transforme plus ou moins radicalement l'aspect du volume et la seule information visuelle facilement compréhensible pour l'utilisateur est une capture du volume transformé sous forme d'image.
Un individu est donc une fonction de transfert et sa représentation, la capture de celle-ci. Nos individus sont affichés dans des carrés de taille fixe de cent soixante-quatre pixels de côté, pour pouvoir concevoir plus facilement l'interface.
Notre interface doit être en mesure de produire des individus aléatoirement. Pour cela, il nous a suffi de reprendre une fonction générant des fonctions de transfert aléatoires et de l'adapter à nos besoins. Une fois la fonction de transfert engendrée, nous l'assignons à notre nouvel individu. Lors de cet assignement, avec l'aide du renderer principal, nous en profitons pour générer l'image qui représentera l'individu dans l'interface.
Pour stocker ces images, nous utilisons des QPixmap, classe de la bibliothèque Qt spécialement conçue pour afficher des images à l'écran. Ici, il est inutile de conserver nos images sous forme de QImage, classe spécialisée dans les traitements lourds et non l'affichage.

5.3.3 Placement des individus, distances entre les images

Nous avons choisi de placer les individus selon les écarts entre eux. Cela implique de calculer une matrice triangulaire supérieure de distances entre les images. Ainsi que d'avoir une méthode fiable pour calculer rapidement la distance entre deux images.
Nous avons choisi de calculer cette distance sur un échantillonnage d'images et en utilisant la représentation de couleur HSV (Hue, Saturation, Value) et en ne nous servant que de la teinte. Concrètement nous parcourons les deux images en même temps, nous considérons un pixel sur chaque image, nous regardons leurs différences de teinte et la stockons. Puis nous sautons un certain nombre de pixels (quatre dans l'application), sinon l'opération prend beaucoup trop de temps, et nous recommençons jusqu'à la fin des images. Une fois les deux images parcourues, nous avons leur distance, que nous stockons.

Chaque image possède un identifiant unique, nous avons stocké ces résultats dans une QMap. Chaque couple d'images étant stocké dans un QPoint, ils pouvaient servir de clé dans la map. Nous associons, bien sûr, à chaque couple une distance.



	Img1	Img2	Img3
Img1	0	25632	35695
Img2	-	0	7800
Img3	-	-	0

FIGURE 5.9 – Processus de comparaison entre images

Le processus étant toujours trop lent malgré l'échantillonnage, nous avons mis en place deux niveaux de parallélisation. Nous devons comparer toutes les images entre elles et stocker le résultat.
Chaque couple d'images effectue sa comparaison dans un thread séparé. Dans chacun de ces threads, il y en a quatre autres. Les deux images sont, chacune, découpées en quatre sous images égales. Pour ce faire, nous utilisons la partie QtConcurrent du Fraework Qt, ceci nous permettant de paralléliser très facilement notre code. Nous pouvons très facilement envoyer une fonction s'exécuter dans un autre thread et récupérer le résultat directement dans un objet QFuture. De plus, les threads utilisés sont pris directement dans le pool de thread de l'application, qui est créé au démarrage de l'application automatiquement par Qt.

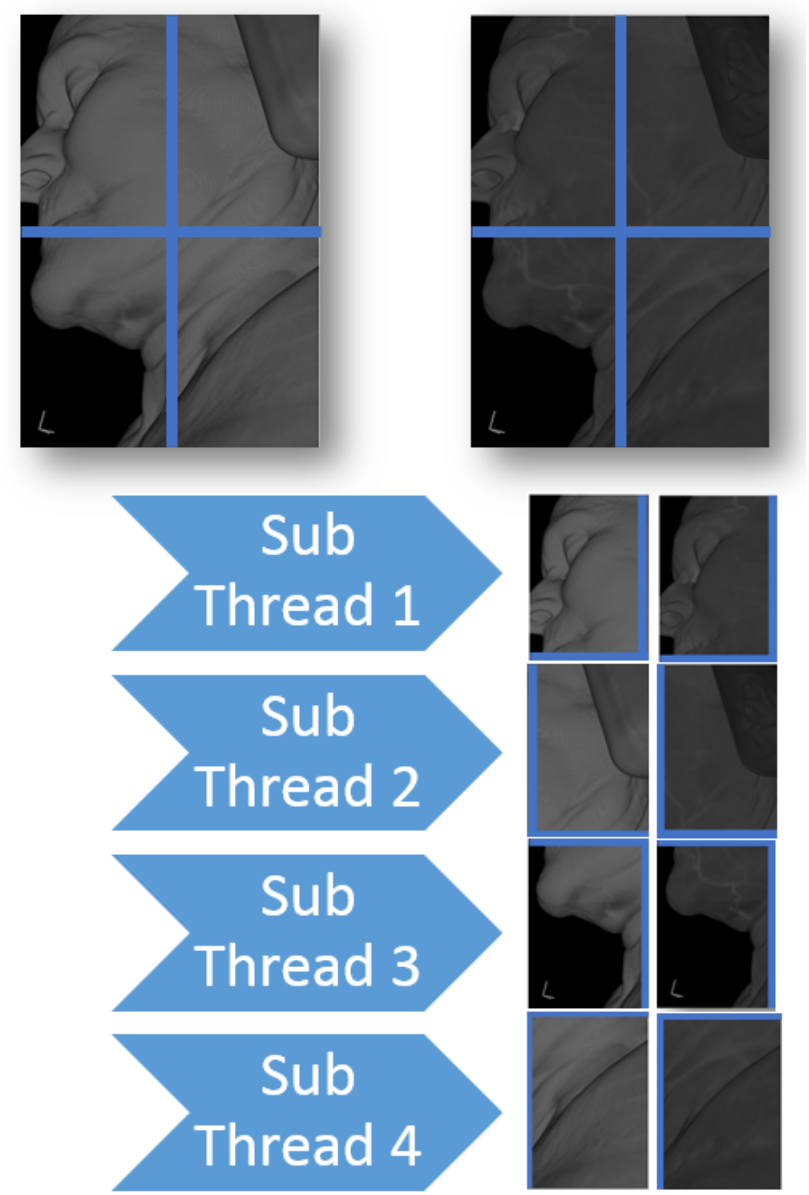


FIGURE 5.10 – Découpages des images en sous parties

Seul problème, lorsque l'on récupérait le résultat de l'écart entre deux images dans un QFutur cela provoquait une fuite mémoire détectée par VLD. Après beaucoup de tests et de recherches, croyant que la faute venait de moi, j'ai trouvé que ceci était un bug actif de Qt 4.8.4 corrigé dans Qt 5.0 et ne pouvant pas être corrigé dans la version 4.8.x (voir lien sur le bug tracker de Qt en annexe).
Le problème a pu être contourné en se passant de l'utilisation des résultats contenus dans les QFuture,

ceux-ci ne servent plus qu'à la synchronisation entre eux. Nous passons, à la fonction de calcul de distance, un pointeur sur un int pour y stocker directement le résultat et ainsi se passer des résultats contenus dans les QFuture.
Une fois la distance entre toutes les images calculée, nous pouvons nous intéresser au placement des individus.
Pour pouvoir les placer à la suite, il est nécessaire de pouvoir les trier dans un ordre de proche en proche. Pour cela nous utilisons l'algorithme BEA, nous avons un tableau dynamique, nous tirons au sort trois éléments du tableau de la population, et les plaçons dans notre tableau dynamique. On insère tous les autres éléments de la population un à un en les plaçant entre chaque couple, de sorte à minimiser l'écart.

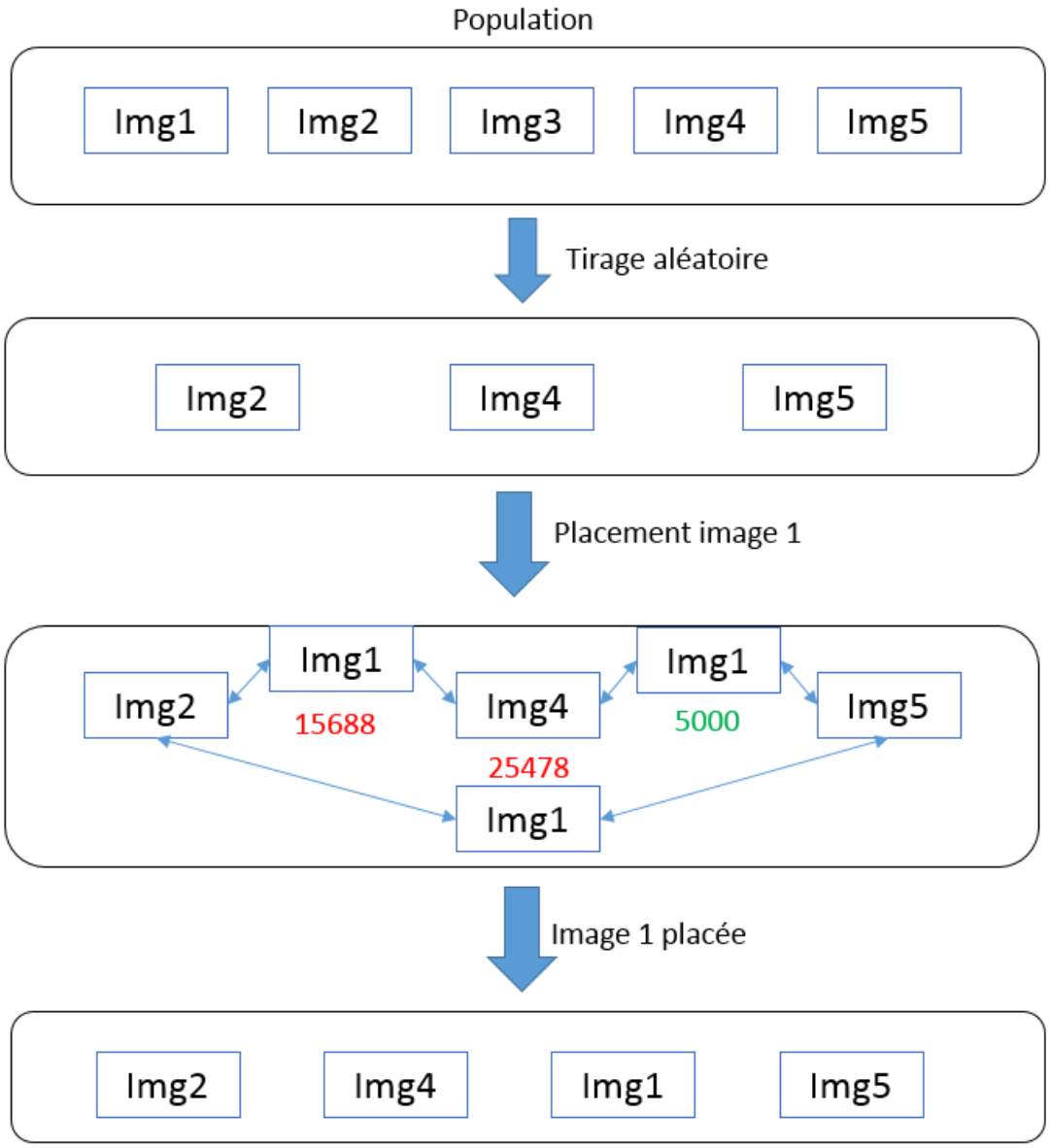


FIGURE 5.11 – Algorithme BEA

Nous avons un chemin englobant la zone centrale, sur lequel nous voulons placer nos individus. Nous pouvons « demander » au chemin quelles sont les coordonnées correspondant à un pourcentage du chemin parcouru, zéro étant le début et un la fin du chemin.
Nous devons afficher un écart représentatif de la distance entre les images. Pour cela nous avons la somme

totale des écarts entre les images. Nous parcourons toutes les images triées dans l'ordre de ressemblance, nous plaçons la première en zéro puis augmentons un compteur de distance pour garder en mémoire la distance totale parcourue.

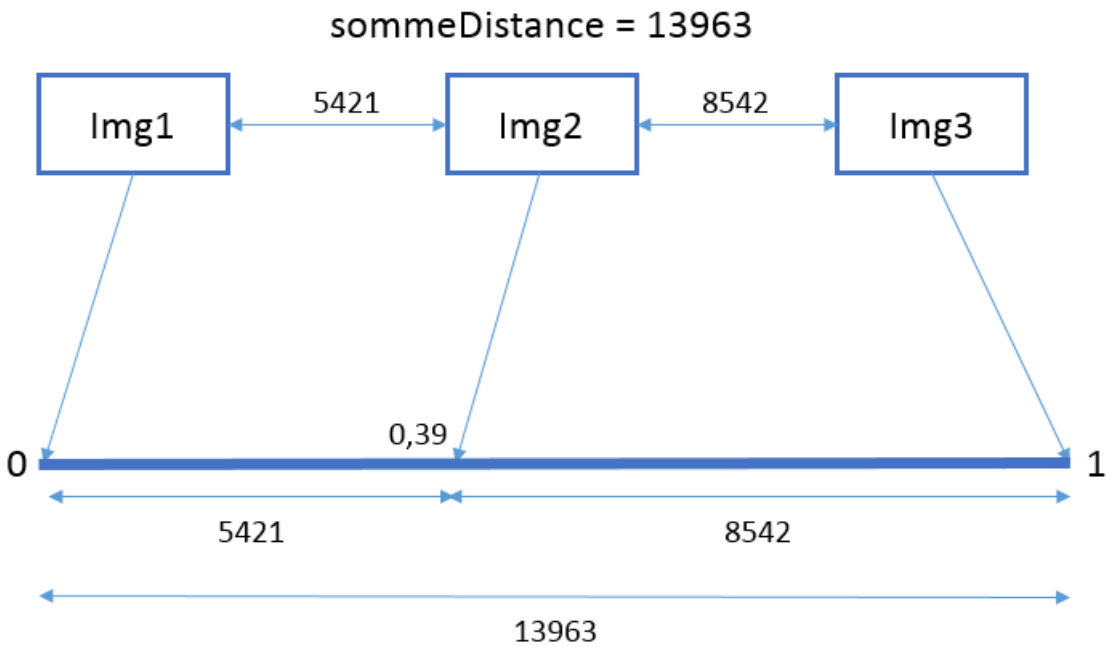


FIGURE 5.12 – Logique de placement des individus

Cette répartition nous assure un placement en fonction de l'écart entre les images. Mais en cas de distance très faible entre deux images, il arrive que nous ayons une légère superposition de celles-ci. Nous avons mis à disposition de l'utilisateur un slider permettant d'augmenter la distance entre les images ou de la réduire d'un certain pas, évitant que l'utilisateur reste bloqué avec des images superposées.
Une fois la position de l'individu déterminée, il nous faut ajouter son image au bon endroit dans la scène. La scène ne gère que des éléments qui héritent de QGraphicsItem, il est donc impossible d'afficher directement le QPixmap que nous avons stocké plus tôt. Mais nous pouvons facilement l'ajouter à la scène qui se charge de nous renvoyer le QGraphicsPixmapItem correspondant.
Nous stockons cet objet dans notre individu et le plaçons aux coordonnés calculées.

5.3.4 Classification ascendante hiérarchique

Nous affichons maintenant des images autour de la zone centrale, avec un espacement fonction de leur distance. Il faut maintenant introduire la notion de groupe dans notre interface. Un groupe sera un ensemble d'individus, le groupe possèdera un individu particulier. Celui-ci sera affiché au sommet de la pile des images, et servira dans les calculs de distances entre les groupes. Nous l'appellerons le représentant du groupe. Un groupe est composé d'au minimum deux individus.
Sachant que le nombre de groupe est défini arbitrairement, et peut être modifié par l'utilisateur, un bon taux de ressemblance au sein d'un groupe ne peut être garanti. En effet, si l'utilisateur spécifie qu'il ne veut que deux groupes dans une population de deux cents individus, au sein des groupes il y aura une énorme différence.
La classification ascendante hiérarchique est un moyen algorithmique de rassembler, dans des groupes, des éléments différentiables par un critère. Le critère dans notre cas, est la distance entre les images. Nous allons maintenant étudier le fonctionnement de l'algorithme mis en place dans l'interface pour effectuer la classification ascendante hiérarchique par saut minimum.

Au début de l'algorithme, nous avons une liste contenant tous les individus, que nous appellerons listItem, cette liste gère aussi bien les individus que les groupes.
Tant que cette liste a un nombre d'éléments supérieur au nombre de groupes voulu, nous effectuons une itération. Une itération consiste à :

- Recherche des deux éléments les plus proches
- Création ou ajout dans un groupe
- Identification de l'élément de saut minimum
- Rajout de la distance au groupe
- Nettoyage

Recherche des deux éléments les plus proches

Nous cherchons les deux éléments qui se ressemblent le plus dans notre population pour les rassembler en groupe. Nous avons à notre disposition la matrice des distances entre images. Il nous faut donc trouver le couple d'éléments ayant la distance minimum. Cela est assez simple, mais nous devons prendre en compte qu'il ne faut pas sélectionner deux fois le même couple. Nous gardons donc dans une liste les couples que nous avons déjà sélectionnés et les excluons lors de la recherche du couple minimum.

Création ou ajout dans un groupe

Maintenant que nous avons notre couple d'éléments ayant la distance minimum, nous devons gérer leur regroupement.
Plusieurs cas peuvent se produire, prenons pour commencer le cas le plus simple, les deux éléments sont des individus. Il suffit alors de créer un nouveau groupe, d'y ajouter les deux individus, de les retirer de la liste de la population et d'insérer le groupe dans la population.
Maintenant si l'un des deux éléments est un groupe et l'autre un individu, il faut ajouter dans ce groupe l'individu en question et le retirer de la population.
Et si les deux éléments sont des groupes, il faut ajouter au groupe un, tous les éléments du groupe deux, et supprimer le groupe deux de la population.

Identification de l'élément de saut minimum

Maintenant que nous avons notre nouveau groupe ou ancien groupe modifié, il nous faut identifier son individu représentant.
L'individu de saut minimum c'est l'individu du groupe qui est le plus proche d'un autre élément de la population. Nous devons donc comparer tous les individus membres du groupe avec tous les autres éléments de la population et garder l'individu du groupe correspondant au minimum. Lorsque nous comparons un individu du groupe avec un autre groupe, nous prenons sa distance par rapport au représentant de l'autre groupe.
Une fois l'individu trouvé nous spécifions au groupe que c'est son représentant, celui-ci peut alors générer son image.
Un groupe est représenté dans l'interface comme étant une pile des images le composant. L'individu au sommet de la pile étant le plus visible, ce sera le représentant.
Pour générer l'image du groupe, nous recopions intégralement l'image du représentant dans une zone plus petite, et les bouts des autres images décalés en diagonale.

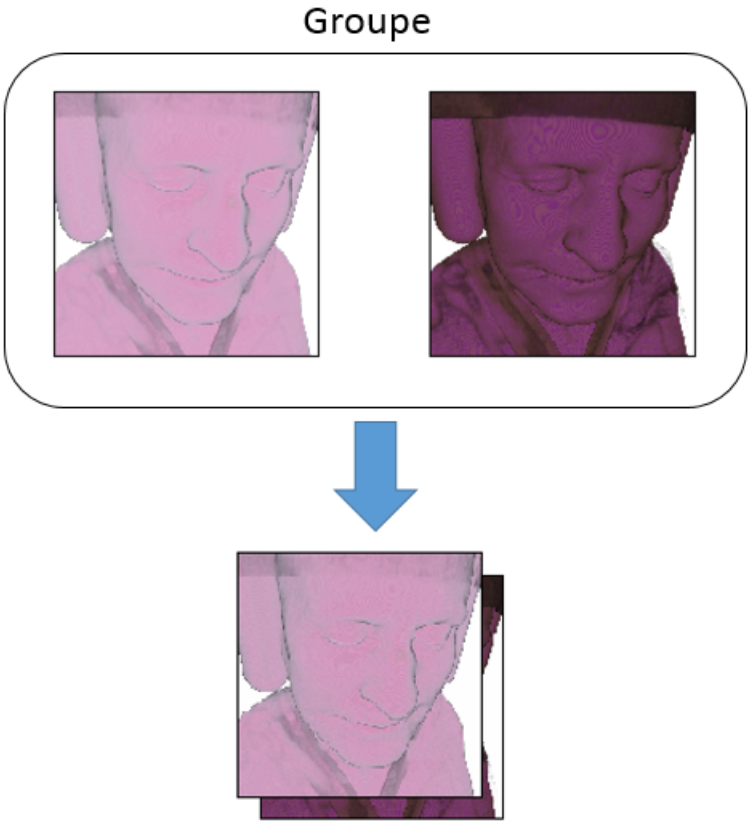


FIGURE 5.13 – Image représentant un groupe

Rajout de la distance au groupe

L'individu représentant du groupe étant identifié, nous devons maintenant modifier la matrice des distances pour intégrer la distance entre notre nouveau groupe et les autres éléments de la population. Sur cet exemple nous venons de créer un nouveau groupe : le groupe Groupe1, celui-ci a pour représentant l'image Img3.

	Img1	Img2	**Img3**	Groupe1
Img1	0	56 411	**8142**	**8142**
Img2	56 411	0	**12 458**	**12 458**
Img3	**8142**	**12 458**	0	0
Groupe1	**8142**	**12 458**	0	0

FIGURE 5.14 – Matrice des distances avec rajout d'un groupe

L'utilisateur peut choisir le nombre de groupe à afficher, il se peut donc que nous soyons amenés à effectuer à nouveau cet algorithme, pour ne pas altérer la matrice de départ avec nos ajouts de groupe, nous dupliquons la matrice des distances et ajoutons les groupes dans celle-ci. Ainsi nous gardons la matrice de distances intacte et pouvons recalculer la classification ascendante hiérarchique sans avoir à recalculer la matrice des distances entre les individus de base de la population.

Nettoyage

Pour finir, enlevons de la matrice des distances tous les individus et groupes n'étant plus pertinents. Les individus ayant rejoint un groupe ne doivent plus apparaitre dans la matrice des distances. Si deux groupes ont été fusionnés, le groupe vide doit aussi être enlevé de la matrice, sinon nous risquons de le sélectionner dans la recherche des deux éléments les plus proches.

La classification ascendante hiérarchique est de loin l'opération la plus longue de tout le processus d'affichage, comme nous le montre cet exemple, effectué avec une cinquante individus au départ et la formation de six groupes.

```
/*** New Cycle ***
 Temps trie selection :  0.018
 Temps generation :  0.02
 Temps clear old data :  0.009
 Temps clear scene :  0.001
 Temps recopie :  0.696
 Temps calculDistance :  0.487
 Temps CAH :  3.954
 Temps generatePixmap :  0.027
 Temps sortList :  0
 Temps DisplayScene :  0.003
/*** End Cycle ***
```

FIGURE 5.15 – Temps d'exécution des fonctions principales

Il serait intéressant de chercher à améliorer l'algorithme, et peut être même de trouver un moyen de le paralléliser.

5.3.5 Sélection d'éléments

Nos éléments étant regroupés et placés, il nous faut pouvoir les sélectionner. Attention à ne pas confondre, nous parlons de sélection graphique dans le sens repère visuel qu'un élément est sélectionné sans aucun autre effet, et de sélection pour la génération suivante dans le sens l'élément a été placé dans la zone centrale et est sélectionné comme parent pour l'algorithme génétique.
Nous allons d'abord définir quelques règles, un élément sera sélectionné par un clic gauche de souris, un nouveau clic gauche sur le même élément le désélectionnera. Plusieurs éléments pourront être sélectionnés si l'utilisateur maintient la touche ctrl enfoncée et effectue des clics gauches sur plusieurs éléments. Si un ou plusieurs éléments sont sélectionnés, tout clic gauche dans l'interface annulera toute la sélection. Et enfin tous les éléments commençant un drag and drop seront automatiquement sélectionnés.
Sachant que l'utilisateur va effectuer ses clics dans la view et non sur la scene comme on pourrait le croire, c'est donc dans la classe WeatherView, que nous avons implémenté la redéfinition des évènements de la souris. Tout comme pour la sélection restrictive dans l'histogramme nous aurons besoin des trois évènements de la souris : MousePressEvent, MouseReleaseEvent et MoveEvent. Ainsi que de deux évènements du clavier : KeyPressEvent et KeyReleaseEvent. À Partir de ces cinq évènements, nous devons créer la logique décrite ci-dessus.

La sélection simple

Pour la sélection graphique simple d'un élément, il nous suffit de regarder si lors du mousePressEvent, le clic est bien un clic gauche et si un élément de la scène est sous la souris. Nous indiquons via un booléen qu'une sélection vient de commencer.
Mais pour savoir si un élément est sous le clic de la souris, il faut prendre en compte un détail important. Dans la mise en place de la scène, nous avons vu que la scène peut avoir une taille infinie et qu'une vue était une fenêtre sur une partie de la scène. Il faut donc bien penser à traduire les coordonnées du clic dans la view en coordonnées de la scene.
La scene est capable de nous envoyer un pointeur sur le QGraphicsItem à une certaine position, nous lui envoyons donc la position du clic et nous comparons le pointeur aux QGraphicsPixmapItem de nos éléments. Si notre booléen de sélection commencé est toujours vrai et qu'aucun drag and drop n'a été détecté lors du mouseReleaseEvent, nous sélectionnons l'élément qui est sous la souris. Cette sélection consiste à ajouter à l'élément un contour vert de sélection, ainsi que de passer le booléen de sélection de notre élément à vrai.

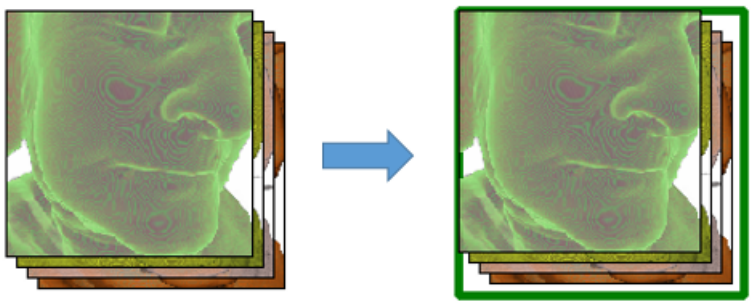


FIGURE 5.16 – Cadre de sélection

Lors de cette sélection, un signal est émis à la fenêtre principale de l'application pour que la fonction de transfert actuellement affichée soit remplacée par celle de l'élément sélectionné ; permettant à l'utilisateur d'explorer le volume correspondant.

La sélection multiple

Nous devons tout d'abord détecter et stocker quelles sont les touches enfoncées par l'utilisateur. Pour cela nous stockons dans une liste dynamique toutes les touches enfoncées, lors de l'évènement keyPressedEvent, et nous retirons les touches vues dans le keyReleaseEvent de la liste. Nous pouvons donc à tout moment savoir si la touche ctrl est maintenue enfoncée par l'utilisateur. Au moment où le bouton de la souris est relâché, avant de tout désélectionner, nous regardons si la touche ctrl est présente dans la liste des touches enfoncées, si elle l'est nous n'effectuons pas la désélection. Ceci nous donne alors un effet cumulatif de deux sélections séparées.

Le drag and drop

Tout d'abord qu'est-ce qu'un drag and drop et à quoi va-t-il nous servir ? Un drag and drop, c'est cliquer sur un élément, et tout en restant cliqué, déplacer sa souris jusqu'à un endroit précis puis relâcher le bouton de la souris. Lors du déplacement de la souris, l'élément se déplace avec la souris, et s'il est relâché à un endroit prévu à cet effet, il y reste, sinon il retourne à sa position de départ.
Dans notre interface, cette logique nous servira à sélectionner pour une génération suivante et non plus que graphiquement les éléments.
Pour effectuer une sélection pour la génération suivante il nous faut « drag and dropé » des éléments déjà sélectionnés graphiquement dans la zone centrale de notre interface.
Nous avons vu que lors d'un mousePressEvent, nous passons un booléen de début de sélection à vrai. S'il y a un mouseMoveEvent et que ce booléen est à vrai, cela veut dire que nous commençons un drag and drop. Nous passons donc le booléen dragAndDropInProgress à vrai. Nous stockons aussi la position de départ de l'élément au cas où le drop n'est pas effectué au bon endroit. Nous rajoutons une teinte verte à l'élément subissant le drag and drop pour signaler visuellement le début de l'opération à l'utilisateur. Maintenant que le dragAndDropInProgress est à vrai tous les mouseMoveEvent suivants auront pour effet de déplacer l'élément avec la souris, de sorte à avoir un effet collé sous la souris.
Au moment du mouseReleaseEvent, puisque nous avons dragAndDropInProgress à vrai, nous savons que c'est la fin d'un drag and drop. À ce moment-là, nous regardons si l'élément est contenu dans le rectangle qui forme la zone centrale de sélection pour la génération suivante. Si celui-ci est bien contenu dans la zone centrale, nous le laissons à cet endroit et lui indiquons qu'il est sélectionné pour la génération suivante.
Sinon il retourne aux coordonnées de placement de l'élément.
Nous avons parlé du drag and drop d'un seul élément pour que cela soit plus facile à comprendre, mais celui-ci peut se faire sur une sélection multiple, tous les éléments sélectionnés seront déplacés da la même façon et sélectionnés pour la génération suivante s'ils terminent leur voyage dans la zone centrale.

5.3.6 Vue intragroupe

Nous voulons pouvoir « regarder » dans un groupe d'individus. Pour cela nous avons choisi de représenter cette vue détaillée par un zoom sur la partie de l'interface où se situe le groupe, après un double clic sur le groupe en lui-même. Une fois dans cette vue, nous affichons autant d'individus que nous permet l'espace disponible, d'un côté de la vue zoomée, et une partie de la zone centrale de l'autre côté. Puisque nous ne voyons pas tous les individus, des boutons suivant et précédent sont disponibles pour permettre la navigation dans le groupe. Une barre de défilement nous montre visuellement où nous nous situons dans le groupe. Il y a aussi un bouton fermer, pour revenir à l'interface non zoomé.

Mise en place de la vue

La mise en place de la vue fut délicate, nos éléments étant des images à placer le long d'un chemin, des problèmes géométriques se sont posés.

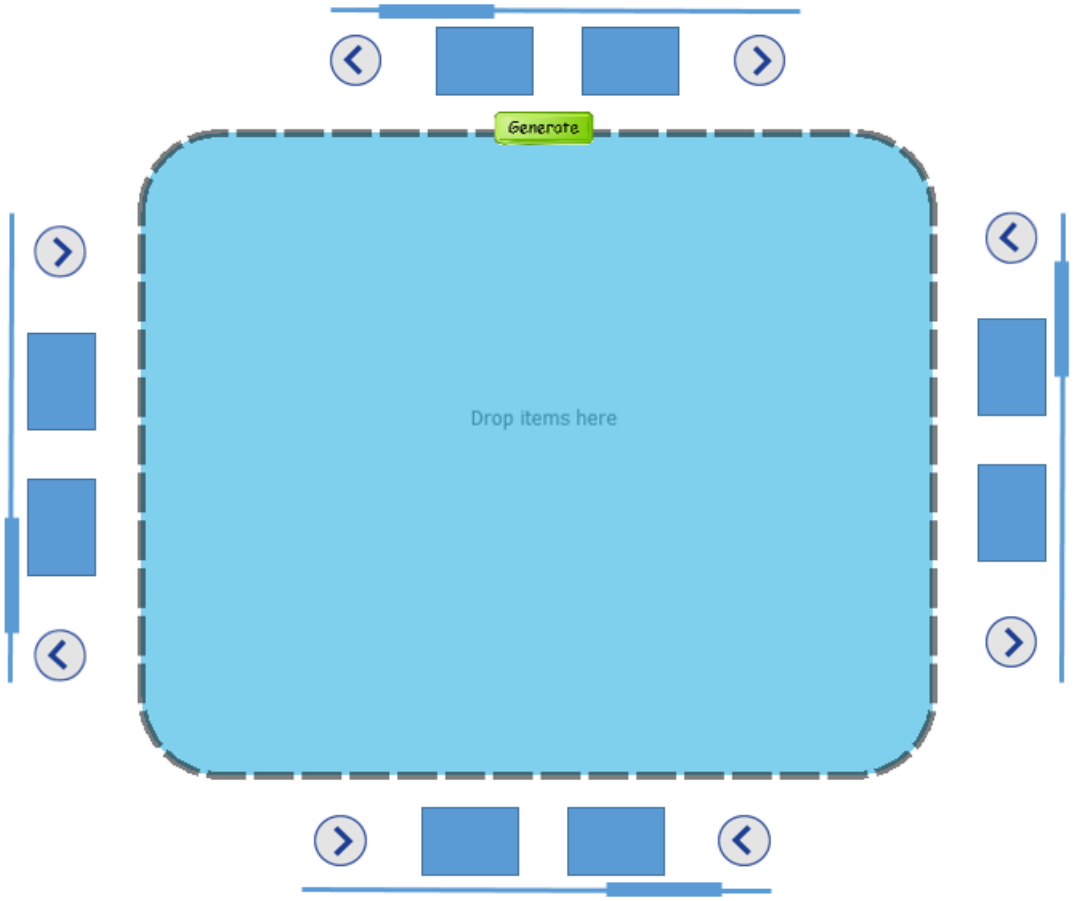


FIGURE 5.17 – Schéma de la vue intragroupe

Sur le schéma ci-dessus, nous avons placé une maquette de la vue intragroupe en fonction de l'endroit où le groupe se trouve. Quand le zoom se fait au-dessus de la partie centrale, il n'y a pas de problème, le dessin se fait de la gauche vers la droite, les boutons suivant et précédent sont correctement positionnés, la barre de scroll est dans le bon sens. Il ne faut pas oublier que ces éléments sont placés sur un chemin, qu'ils ne font que suivre.
Si le zoom se fait sur la partie droite, le dessin se fait du haut vers le bas, nous devons appliquer une rotation aux images des boutons suivant et précédent, sinon tout le reste est conforme à nos attentes. Faut-il encore pouvoir détecter le fait que le zoom soit sur le côté droit.
Nous arrivons maintenant à la partie délicate, jusque-là le principe de chemin s'est avéré très utile et pratique. Mais nous pouvons voir sur le schéma qu'au moment où nous sommes sous la zone centrale tout le placement est inversé, en effet le dessin s'effectue à ce moment-là de la droite vers la gauche inversant la barre de défilement ainsi que les boutons suivant et précédent.
Si nous zoomons sur la partie gauche, le dessin se fait de bas en haut, et pour ce cas-ci cela nous convient, il faudra ici aussi appliquer une rotation aux boutons, tous les éléments de la vue sont placés correctement.
Il a donc fallu corriger deux problèmes : la rotation des boutons selon leurs emplacements et la barre de scroll inversée sur le dessous de la zone centrale.

Tout d'abord, regardons de plus près la technique pour extraire le sous-chemin du groupe zoomé par rapport au chemin principal.

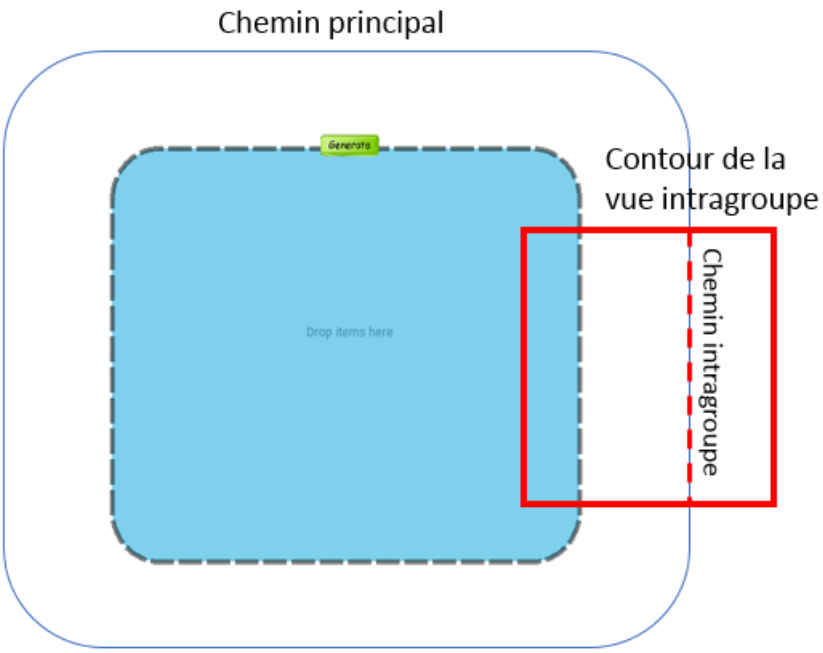


FIGURE 5.18 – Création du chemin de la vue intragroupe

Une fois la vue zoomée sur la zone qui nous intéresse nous parcourons le chemin principal avec un très petit pas et nous regardons si le point du chemin est visible, auquel cas nous le stockons. Une fois tout le chemin principal parcouru, nous construisons un nouveau chemin composé uniquement des points visibles du chemin principal, cela nous donne le chemin intragroupe.
Pour remédier au problème de chemin inversé sous la zone centrale, il suffit de parcourir le chemin principal en sens inverse. Il ne nous reste plus qu'à savoir quand nous sommes dans un cas en dessous de la zone centrale.
Pour cela nous avons découpé l'interface en quatre zones, chaque zone est un triangle stocké sous forme

d'un QPolygonF, cela nous permet de savoir à quelle zone appartient n'importe quel point.

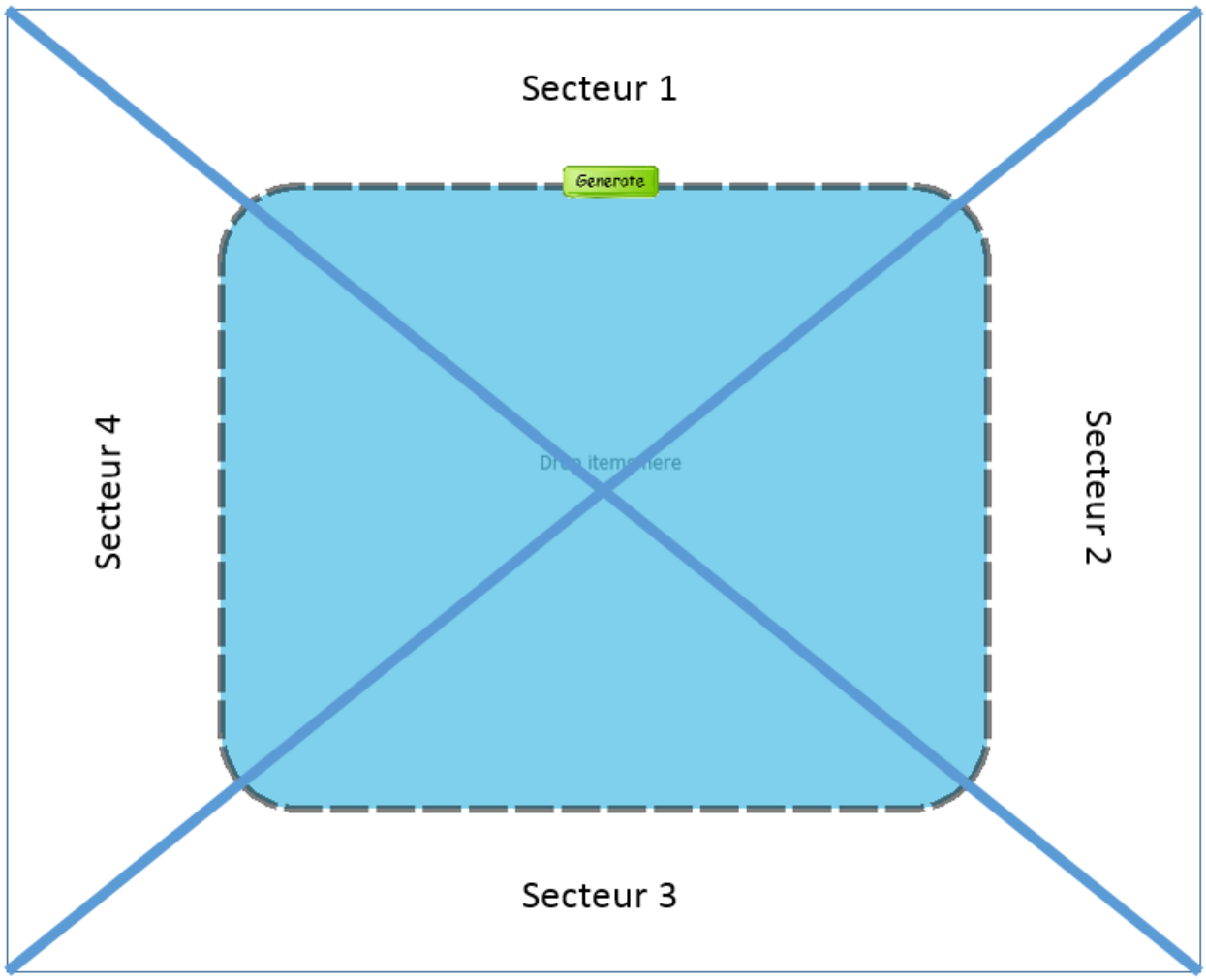


FIGURE 5.19 – Découpage en secteurs de la zone centrale

Lorsque la position du groupe zoomé est dans le secteur 3 nous inversons le parcours du chemin principal lors du parcours pour créer le chemin intragroupe.
En plus de régler notre problème de chemin inversé, nous pouvons savoir dans quel secteur se trouvent nos boutons et adapter leurs rotations en fonction.
Seulement un cas particulier dans le coin supérieur gauche persiste. Ce coin est spécial puisque c'est ici que le début et la fin du chemin se rejoignent. Lorsque nous parcourons le chemin principal, nous utilisons la fonction QPoint QPainterPath : : pointAtPercent(float).

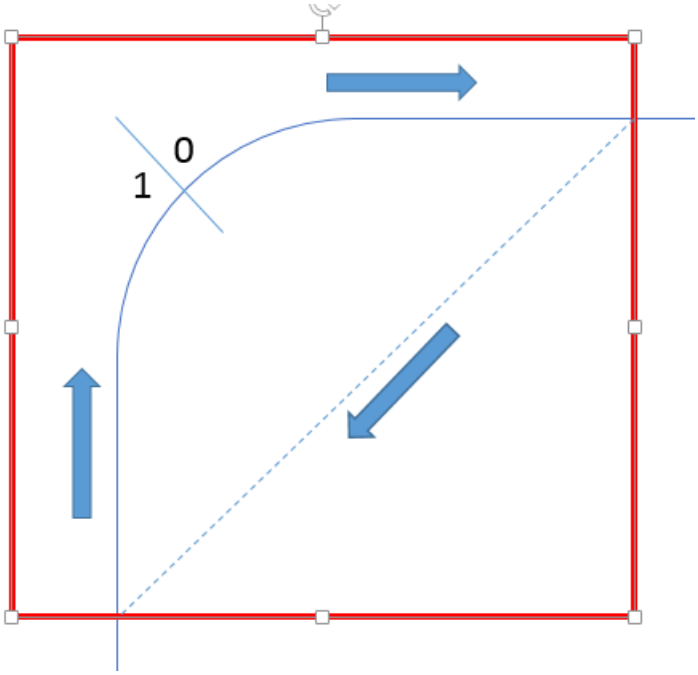


FIGURE 5.20 – Schéma du problème du chemin circulaire

Et dans ce cas précis, quel que soit le sens de parcours du chemin, nous avons une diagonale indésirable qui se forme.
Pour parer à ce problème, nous avons décidé de trier les points visibles stockés lors du parcours du chemin principal. Nous les trions sur deux critères, par y décroissant et x croissant. Puis nous traçons le nouveau chemin.
Cela résout notre dernier problème de placement.

La sélection intragroupe

sont compatibles avec l'interface zoomée. La sélection, désélection multiple où non ainsi que le drag and drop sont directement supportés puisque nous avons redéfini les évènements dans la vue. En contrepartie la vue zoomée nous impose de gérer des cas particuliers. Par exemple, lorsque l'on sélectionne pour la génération suivante un élément du groupe, il faut qu'un nouvel élément, s'il y en a, le remplace et que la barre de défilement s'adapte au changement.
Si nous sélectionnons tous les éléments d'un groupe depuis sa vue intragroupe, il faut qu'au moment où le dernier élément est sélectionné pour la génération suivante, que le groupe se ferme, et qu'il soit détruit.
Il faut aussi gérer le cas où l'utilisateur ne laisse qu'un seul élément dans le groupe et referme celui-ci, le groupe ne peut pas contenir qu'un seul élément, il faut donc détruire le groupe et afficher uniquement le dernier élément sous forme d'individu.
De même il faut gérer plusieurs cas spécifiques lors des désélections d'individus qui étaient présents dans des groupes.

Algorithme de sélection intragroupe Lorsque l'utilisateur sélectionne pour la génération suivante tous les individus d'un groupe depuis la vue intragroupe, la vue se ferme automatiquement.
Derrière ce mécanisme se cache un algorithme que nous allons voir en détail. Chaque fois qu'un individu est sélectionné depuis la vue intragroupe, un signal est envoyé à l'interface pour qu'elle gère les cas particuliers.
Dans tous les cas, on retire l'individu de la liste des individus composant le groupe, nous stockons dans l'individu le numéro de son ancien groupe et nous le rajoutons dans la liste des éléments de l'interface non zoomée.
Puis nous demandons à la scène de redessiner la vue intragroupe, cela ayant pour effet de mettre à jour la barre de défilement et de compléter le « trou » créé par l'individu enlevé.
Si en plus l'individu que l'on retire est l'avant-dernier, le groupe aussi subit des modifications. En effet, puisque l'on retire l'avant-dernier individu du groupe il faut que le groupe disparaisse et que le dernier individu prenne sa place dans l'interface non zoomée.
Nous retirons donc au groupe son dernier élément que nous plaçons à sa place dans l'interface non zoomée, et ajoutons celui-ci dans une liste spéciale appelée liste des orphelins. Le groupe est quant à lui caché et non détruit puisqu'il pourrait resservir.

Si l'individu que l'on retire est le dernier du groupe, nous fermons la vue intragroupe et cachons le groupe.

Algorithme de désélection intragroupe Selon le même principe, chaque fois qu'un individu qui appartenait à un groupe est désélectionné, un signal est envoyé à l'interface non zoomée pour gérer les cas particuliers.
Si le groupe de l'individu ne contient aucun autre individu, l'individu désélectionné prend la place du groupe dans l'interface non zoomée, et celui-ci est ajouté à la liste des orphelins.
Si le groupe de l'individu ne contient aucun autre individu, mais qu'il possède un frère dans la liste des orphelins, les deux individus sont remis dans leur groupe, l'orphelin est retiré de la liste. Le groupe est affiché à sa place, son image est générée à partir des deux individus. Les deux individus sont supprimés de

la scène.

Si le groupe de l'individu contient déjà d'autres individus, on rajoute simplement notre individu dans son groupe, on génère une nouvelle image le représentant et on supprime de la scène notre individu.

Navigation et barre de défilement Pour que la barre de défilement épouse la même courbe que le chemin intragroupe, nous avons placé un chemin invisible qui englobe nos individus et selon le même principe que pour la création du chemin intragroupe, nous ne prenons que les points du chemin visibles entre les boutons suivant et précédent.
Une fois le chemin de la barre de défilement en place, nous calculons la distance entre les boutons suivant et précédent pour savoir combien d'images nous pouvons mettre entre eux. Puis nous plaçons en fonction du nombre d'images totales par rapport au nombre d'images affichées, le nombre de crans de la barre de défilement.

5.3.7 Cycle

Lorsque l'utilisateur appuie sur le bouton de génération sur le bord de la zone centrale, tous les éléments sélectionnés pour la génération suivante doivent être envoyés à l'algorithme génétique déjà en place. Si aucun élément n'est sélectionné nous générons aléatoirement une nouvelle population. Il nous a fallu retrouver dans le code existant la partie correspondant à l'algorithme génétique, analyser les éléments attendus en entrée et adapter nos données.
Nos individus WeatherIndividual possèdent tous une fonction de transfert les définissant, or l'algorithme génétique demande des FTConfigContainer. Nous devons donc convertir nos individus en FTConfigContainer. Cette classe possède plusieurs options qui ne nous intéressent pas, pour l'instant le seul attribut important est la fonction de transfert, c'est elle qui la stocke pour être utilisée dans l'algorithme génétique. Pour la conversion nous créons de nouveaux FTConfigContainer et leur passons juste la fonction de transfert de nos WeatherIndividual.
Maintenant que notre population est prête, nous devons l'envoyer dans l'algorithme génétique. Celui-ci s'appelle via une classe contenant la configuration des différents paramètres de l'algorithme, comme le taux de mutation ou de croisement. Cette classe s'appelle FTConfigGenerator, elle possède une méthode generate qui prend en entrée une liste de FTConfigContainer, notre population, et nous en renvoie une autre qui correspond à la population résultante de l'algorithme génétique.
Une fois la nouvelle population récupérée, nous devons faire la conversion inverse. C'est-à-dire extraire les fonctions de transfert des FTConfigContainer et créer de nouveaux WeatherIndividual avec celles-ci.
Une fois la nouvelle population récupérée dans le bon « format », nous devons nettoyer entièrement la scène ainsi que toutes nos structures de données, pour repartir d'une feuille blanche. Ensuite nous repartons avec la même logique d'affichage : nous calculons les distances entres les images, nous effectuons la classification ascendante hiérarchique, nous trions avec l'algorithme BEA et nous affichons la scène.

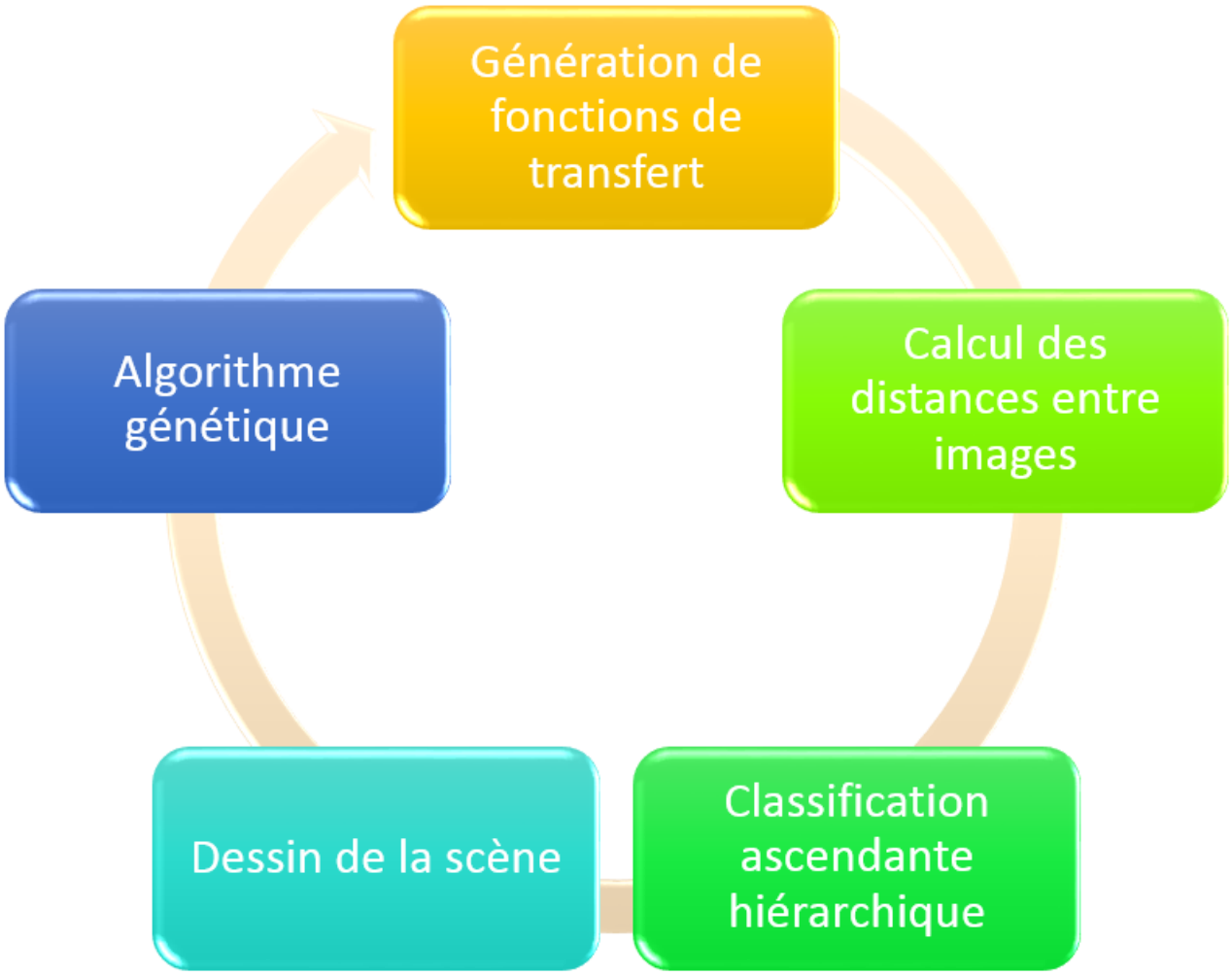


FIGURE 5.21 – Etapes d'un cycle

La réutilisation du code de l'algorithme génétique devait nous faire gagner beaucoup de temps, finalement nous en a fait perdre encore plus. En effet, dès le premier test, l'application rencontrait une double libération de mémoire lors de l'opération de nettoyage des classes utilisées avec l'algorithme génétique. En mettant cette partie en commentaire l'application et le cycle fonctionnait parfaitement. Après analyse de l'exécution avec VLD, je me suis aperçu qu'il y avait beaucoup de fuites mémoires. Le code de l'algorithme génétique était entièrement écrit sans aucune gestion de la mémoire, sans parler de l'indentation, j'ai même pu trouver des variables non utilisées signalées par le compilateur qui était alloué au milieu de la méthode. J'ai donc du parcourir tout le code et avec l'aide de VLD, corriger toutes les fuites mémoires, cela me pris énormément de temps, le code n'étant pas de moi, mais surtout à cause d'un cas particulier.

Lors de la succession de l'enjambement et de la mutation, différentes populations sont utilisées, mais étant composées de pointeurs sur des FTConfigContainer, les populations s'entrecroisent. Cela rend impossible le parcours des populations pour supprimer les FTConfigContainer qui les composent, puisque certains sont présent plusieurs fois dans des populations différentes et que d'autres ont leur fonction de transfert directement dans un de nos WeatherIndividual.

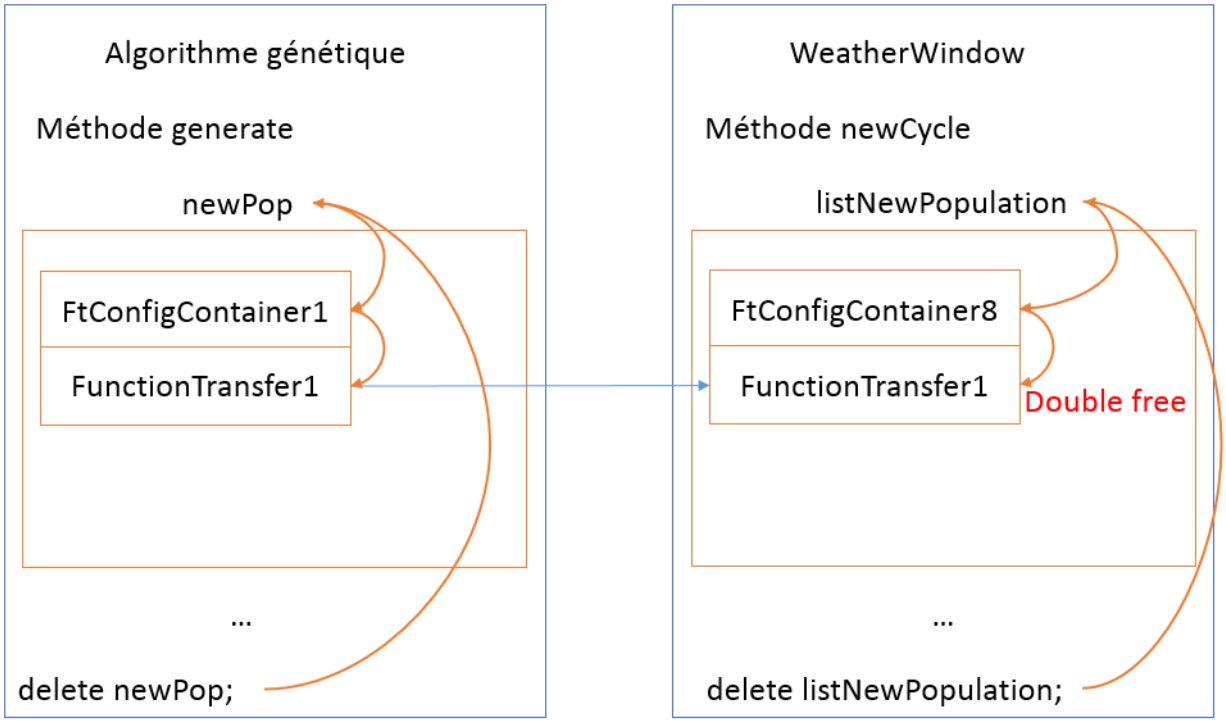


FIGURE 5.22 – Schéma d'une fuite mémoire

Pour remédier à cela, au moment de supprimer les populations, je n'ai pas trouvé d'autre moyen que de mettre toutes les fonctions de transfert des FTConfigContainer à NULL, de libérer les classes FTConfigContainer qui se détruisent maintenant sans libérer les fonctions de transfert NULL. Cela résout deux problèmes, le fait de libérer les FTConfigContainer résout le problème de fuite mémoire, le fait qu'il ne détruise plus automatiquement la fonction de transfert, résout le problème de double libération de la mémoire.

5.3.8 Wizard

Lors du lancement de l'interface de Mélodie, un wizard nous permet de configurer les paramètres de l'algorithme génétique, ainsi que de charger si on le souhaite des fonctions de transfert précédemment sauvegarder. Ceci est très utile si on ne veut pas partir de zéro lors de la première génération de population.

Nous avons donc décidé de reprendre le code du wizard, et de ses pages, pour les intégrer dans notre interface.
Nous avons dû dupliquer le code du wizard et de ses pages, Mélodie affichant son interface directement dans la dernière page du wizard, nous ne pouvions pas utiliser sa version du wizard, car nous savions d'avance que nous allions grandement le modifier.

Après analyse du code, nous avons repris uniquement les parties utiles. À savoir, la page permettant de charger des fonctions de transfert sauvegardées et le bouton permettant d'ouvrir une interface de configuration des paramètres. Nous avons retouché toutes les interfaces des pages pour que le tout paraisse plus propre, et encore une fois nous avons dû corriger des fuites mémoire, cette fois plus légère puisque c'était juste une mauvaise compréhension du système de hiérarchie de parents dans Qt.

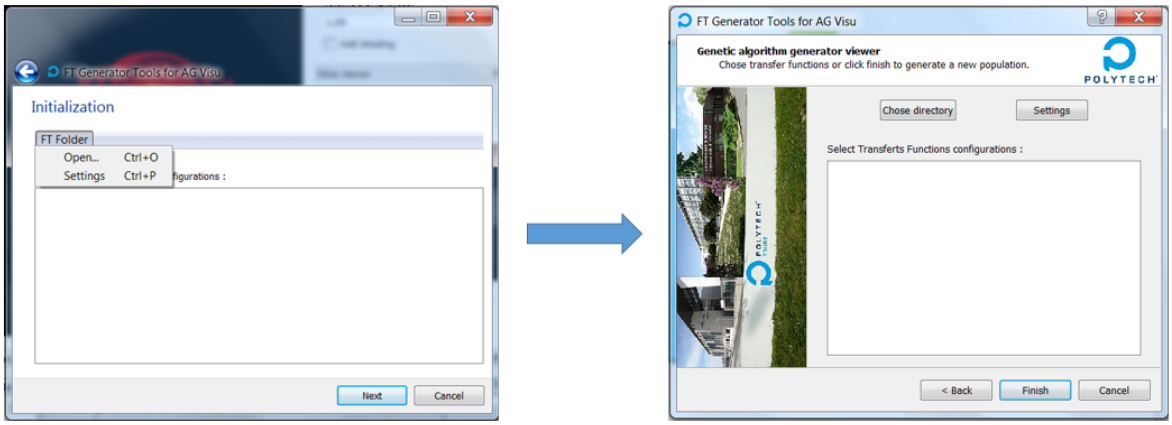


FIGURE 5.23 – Avant après design du Wizard

Pour l'interface de la page de configuration des paramètres de l'algorithme génétique, nous avons trouvé un fichier d'interface de Qt généré avec Qt Designer, correspondant à notre vue actuelle, nous l'avons donc modifié pour la mettre en forme correctement et lorsque nous avons relancé l'application aucune de nos modifications n'étaient prises en compte. Après quelques recherches nous nous sommes aperçus que pour cette page, toute l'interface graphique était redéfinie à la main directement dans le code et que le fichier d'interface de Qt n'était jamais utilisé. Nous avons supprimé tout le code relié à l'interface codée à la main et correctement relié le fichier d'interface à cette classe. Nous avons également effectué ces modifications dans le code du wizard de Mélodie.

5.3.9 Evolutions possibles

L'évolution la plus importante à faire concerne le design de la nouvelle interface, revoir les couleurs et pourquoi pas les formes, pour que cela fasse plus professionnel.
Mise à part esthétique, il y a plusieurs améliorations qui pourraient être intéressantes.
L'interface a actuellement un temps d'attente entre les itérations, d'un tiers dépendant du calcul de différence entre les images. Il pourrait être intéressant de tester différentes méthodes pour ce calcul afin d'avoir un temps de réponse plus rapide. Il pourrait être intéressant de revoir totalement l'algorithme de classification hiérarchique ascendante qui est responsable au deux tiers du temps d'attente entre les itérations.
Dans la vue intragroupe, il faudrait rendre la barre de défilement active, pour que l'utilisateur puisse interagir directement avec celle-ci.
L'évolution la plus importante découle des premiers retours de tests utilisateurs effectués, il serait intéressant de garder dans la zone centrale les individus parents de la génération actuelle.

Gestion de projet

6.1 Planning

Comme expliqué dans la partie environnement du projet de ce rapport, j'ai perdu du temps dès le départ en migrant l'application sur mon ordinateur portable. Le planning prévu pour le projet fourni avec le cahier de spécification tient compte de ce retard.

Tâche	Charge J/H	27/09/2012	29/11/2012	03/12/2012	06/12/2012	31/01/2013	07/02/2013	21/03/2013	26/03/2013
Prise en main existant	**10**								
Rédaction CdS	*2*								
Migration du logiciel	*8*								
Sélection dans les imagettes et restriction de l'histogramme	**9**								
Fonction de détection de clic dans l'imagette	*1*								
Fonction de dessin du rectangle de sélection	*1*								
Fonction de sauvegarde des niveaux de gris sélectionné	*2*								
Fonction d'ajout/suppression/reset de la zone de sélection	*2*								
Modification graphique de l'IHM existante	*2*								
Fonction de calcul de l'histogramme selon les zones sélectionnées	*1*								
Weather View	**17**								
Conception de la nouvelle zone d'affichage	*3*								
Intégrations des fonctions de croisement et de mutation de Mélodie	*4*								
Choix du rendu parmi les algorithmes proposés	*4*								
Fonction de calcul de distance entre deux images/volumes	*2*								
Fonction de placement et d'affichages des images	*2*								
Fonction de gestion de la sélection pour la génération suivante	*2*								
Remise en place des logs	**1**								
Adapter la classe de log existante	*0,5*								
Remettre en place du système de log	*0,5*								
Rédaction des rapports et différents documents	**37**	CdS V1					CdS Final		
Test	**5**								

FIGURE 6.1 – Planning prévisionnel

Toute la première partie du PFE a été consacré à l'étude de l'existant, à la réflexion sur le travail à réaliser ainsi que sur la rédaction du cahier de spécifications. Cette première partie correspond à la partie de l'année ou nous avions un jour dédié au PFE par semaine. Dès que nous sommes rentrés dans la seconde partie de l'année avec trois jours dédiés au PFE par semaine, le développement a commencé. Comme nous avons pu le voir le développement s'est déroulé en deux grandes phases, la mise en place de la sélection restrictive et la création de la WeatherView. La mise en place de la sélection restrictive fut plus aisée qu'initialement prévue et pris donc moins de temps, la tâche a été surestimée d'environ six jours hommes. La création de la WeatherView a, quant à elle, été largement sous-estimée, elle a pris fin avec plus de vingt jours homme. Ce retard s'explique en partie par la gestion de bugs rencontrés, des fuites mémoire ou encore des cas particuliers non prévus rencontrés lors du développement de cette partie. L'autre partie du retard s'explique par une large sous-estimation de celle-ci, et révèle donc une mauvaise analyse du travail à réaliser en amont.
La partie de remise en place des fichiers de logs fut rapide comme prévue dans le planning initial.

6.2 Outils utilisés

Au fil du projet nous avons eu des besoins spécifiques pour lesquels il nous a fallu trouver des logiciels gratuits.
Par exemple, lors de la migration de l'environnement de développement nous avons eu besoin de savoir quelles étaient les dépendances d'un .exe, pour cela mon encadrant m'a orienté directement vers le logiciel Dependency Walker, qui me fut très utile.
Plus tard, lors de l'optimisation de certaines fonctions comme le calcul de distance entre images ou encore, la classification ascendante hiérarchique, j'ai eu besoin de savoir combien de temps passait le programme dans chaque fonction. Pour cela, j'ai trouvé le logiciel Very Sleepy, qui est un profileur de code, et nous donne pour une exécution le temps exacte passé dans chaque fonction de l'application.

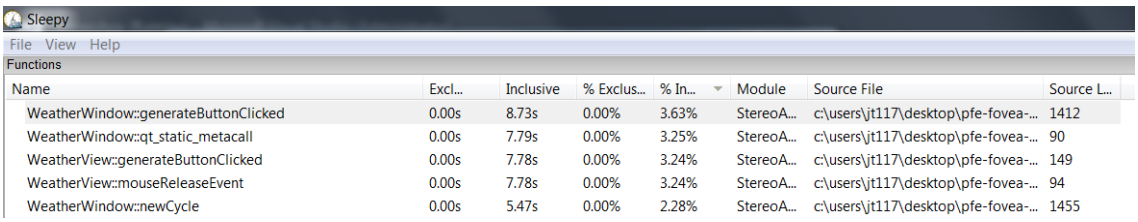


FIGURE 6.2 – Interface de Very Sleepy

Pour le contrôle de qualité du code en C++, il y a le logiciel CppCheck qui fournit un retour détaillé sur la meilleure pratique de codage à avoir.

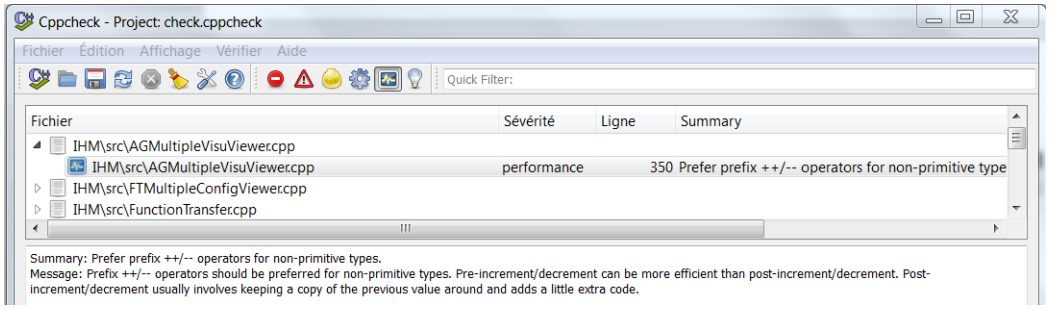


FIGURE 6.3 – Interface de CppChecker

Et pour finir le logiciel le plus important, à savoir VLD (Visual Leak Detector), qui permet de détecter

et retracer les fuites mémoire lors de l'exécution du programme.
Aucun outil de gestion de version n'a été utilisé, une seule personne travaillant sur le code n'a pas besoin d'avoir de gestionnaire de version. Pour la sauvegarde, un disque dur externe a été utilisé.

Conclusion

Ce projet m'a permis d'approfondir mes connaissances ainsi que d'acquérir de nouvelles compétences. Étant Qt Ambassador, j'étais déjà familier avec l'environnement de travail, mais j'ai quand même appris de nouvelles choses, notamment à contourner les bugs du framework.
Toute la démarche de prise en main sur un projet aussi conséquent me sera très utile en entreprise. J'ai beaucoup apprécié le côté réel du projet, c'est une application qui a un but concret, et cela est motivant.
J'ai pu découvrir les possibilités d'analyses que présente un scanner IRM, et comment en modifiant une fonction de transfert on peut « voir » à travers la peau. Je trouve toujours cet effet bluffant. Au cours de mon PFE, j'ai pu mettre en place un système de sélection restrictif, qui me semble être une bonne idée qui pourrait être amenée à évoluer en une interface secondaire à part entière. J'ai aussi rajouté une nouvelle interface d'interaction avec l'algorithme génétique déjà en place. Cette nouvelle interface apporte des idées nouvelles qu'il faudra valider ou invalider avec des tests utilisateurs. À la suite de ces tests il faudra, soit garder et améliorer cette interface, soit en développer une autre entièrement, basée sur une autre idée.
Tous les objectifs du cahier de spécification ont étaient remplis. Je suis fier des résultats obtenus, ainsi que de la quantité et qualité du travail que j'ai fourni.
Je suis content d'avoir pu apporter une contribution sur un projet comme celui-ci. Même si ma nouvelle interface n'est pas conservée, les quelques fuites mémoire et bug que j'ai corrigé persisteront encore un peu dans l'application.
J'ai aussi apprécié l'implication de mon encadrant et de Mr Venturini dans le projet. Ils passaient régulièrement nous voir, et étaient toujours disponibles dans leur bureau pour nous aider. Je les en remercie encore une fois.

Références

Billet de blog sur les QSharedPointer http://blog.codef00.com/2011/12/15/not-so-much-fun-with-qsharedpointer/

Documentation officielle de Qt http://qt-project.org/doc/qt-4.8/

Guide d'optimisation en C++ http://www.tantalon.com/pete/cppopt/main.htm

Page du bugtracker de Qt sur la fuite mémoire de QtFuture https://bugreports.qt-project.org/browse/QTBUG-28242?page=com.atlassian.jira.plugin.system.issuetabpanels:all-tabpanel

Page d'aide sur la limitation mémoire http://stackoverflow.com/questions/8300899/visual-studio-2010devenv-exe-o

Aide pour compiler Qt 4.8.4 à partir des sources pour Visual Studio 2012 http://stackoverflow.com/questions/12113400/compiling-qt-4-8-x-for-visual-studio-2012

Site officiel CMake http://www.cmake.org/

Site officiel Insight Toolkit (itk) http://www.itk.org/

Site officiel QWT http://qwt.sourceforge.net

Drivers et toolkit CUDA https://developer.nvidia.com/cuda-downloads

Site officiel Qt http://qt-project.org/

École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique

Cahier de spécification système & plan de développement			
Projet :	**Optimisation interactive pour la visualisation d?images médicales**		
Emetteur :	Julien Teruel	Coordonnées : EPU-DI	
Date d'émission :	6 janvier 2013		
Validation			
Nom	Date	Valide (O/N)	Commentaires

Barthélemy Serres : 05/01/2013 O "Je pense que nous sommes bien d'accord sur le travail à faire. Concernant les tests, il faudra se servir de ton schéma pour construire des tests utilisateurs, à faire apparaitre dans un cahier de tests, mais ce sera pour plus tard."

Historique des modifications		
Version	Date	Description de la modification

00 : 03/12/2012 ; Version initiale
01 : 30/12/2012 ; Version 1 : Modifications basées sur les retours de Barthélemy.

Table des matières

Cahier de spécification système

1.1 Introduction

L'objectif de ce document est de spécifier le travail à réaliser. Pour cela nous analyserons en détails le logiciel existant, puis nous verrons quelles sont les améliorations demandées par l'équipe FOVEA, pourquoi de telles changements sont demandés ainsi que comment ceux-ci seront intégrés dans le logiciel existant.
Ce document servira de référence en cas de conflit lors de la livraison du produit final. Il fera foi des engagements pris par les deux parties.
Dans ce document la maîtrise d'ouvrage fera référence à l'équipe FOVEA, et plus particulièrement à l'encadrant du projet de fin d'étude Barthélemy Serres. La maîtrise d' ?uvre fera référence à l'étudiant Julien Teruel.

1.2 Contexte de la réalisation

1.2.1 Contexte

Ce projet de fin d'étude fait suite à trois autres projets de fin d'études réalisé respectivement par Nicolas Nathan (2010), Sylvain Botto(2011), Mélodie Roger(2012).Chaque projet vient améliorer et compléter la base existante de Nicolas.
Chacun d'entre nous peut être amené, un jour, à subir un IRM pour des raisons médicales. L'analyse de ces images par des médecins est essentiel pour le suivi du patient. A l'heure actuelle les médecins modifient le rendu de l'IRM via quelques fonctions prédéfinies, mais pas se déplacer librement à la recherche d'un élément très particulier.
Le logiciel StereoApp est un logiciel de visualisation d'image médicale. Il prends les images DICOM produites en sortie par l'IRM, et en fait une représentation 3D sur l'écran d'un ordinateur. Et si l'équipement est adéquate le logiciel peut même afficher le résutlat en 3D stéréoscopique.

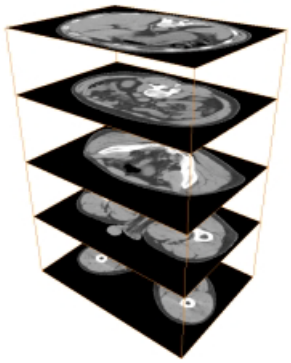


FIGURE 1.1 – Image 2D assemblé en une image 3D - DICOM

La manipulation de la fonction de transfert sur le modèle 3D permet aux médecins d'exploiter pleinement toutes les données disponible dans un IRM. Seulement la manipulation de la fonction de transfert est très délicate et peut demander beaucoup de temps si l'on cherche à voir un élément précis dans le modèle. Pour essayer de guider l'utilisateur dans ses recherches, un algorithme génétique prends des points au hasard sur la fonction de transfert et les fait varier, en espérant se proposer une fonction donnant un résultat proche du résultat recherché. Le but du projet de fin d'étude de Sylvain, puis de Mélodie fut de proposer une interface graphique permettant d'afficher les résultats de cet algorithme.
L'interface graphique de Sylvain proposait neufs images résultantes de l'algorithme génétique, que l'utilisateur pouvait sélectionner, pour générer une nouveau jeu d'images. Ainsi de proche en proche il pouvait plus facilement s'orienter vers le résultat espéré. Seulement afficher un jeu de neufs images, réduit énormément

les chances de générer une images potentiellement intéressante. Et sachant que c'est à partir de ces individus que l'algorithme va générer le jeu d'images suivant, cela restreint directement les images atteignables.

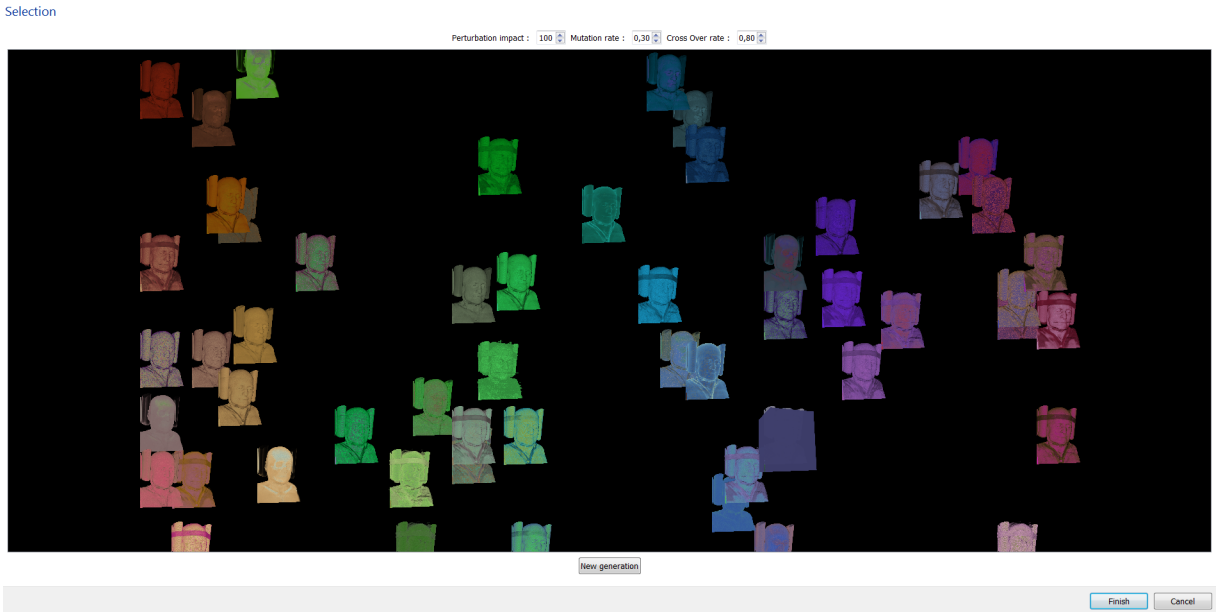


FIGURE 1.2 – Interface conçue par Mélodie

C'est là qu'est intervenue Mélodie, proposant une interface affichant beaucoup plus d'individus, et les classant selon leurs couleurs et luminances pour essayer de grouper les images se ressemblant. L'utilisateur peut alors choisir tous les individus se rapprochant de ce qu'il recherche, pour ensuite générer un nouveau jeu d'images. Ces deux interfaces ainsi que le principe d'un algorithme génétique seront expliqué plus en détails dans la suite du cahier de spécification.

1.2.2 Objectifs

StereoApp est un logiciel d'aide à la décision destiné aux médecins. Le sujet de ce projet de fin d'étude, consiste à améliorer le logiciel existant, en proposant une nouvelle interface graphique pour les résultats de l'algorithme génétique. Mais aussi, dans l'idée de restreindre l'algorithme génétique pour affiner le guidage de l'utilisateur, de proposer un outils permettant de sélectionné des plages de couleurs sur les images de coupes 2D, et ainsi restreindre l'histogramme et par la même occasion l'amplitude laissé à l'algorithme génétique.
Il est aussi question de tester l'application avec du matériel reçu par le laboratoire FOVEA, à savoir un écran 127cm avec fonctionnalité 3D, pour afficher notre interface principale afin que le médecin puisse voir en détail le modèle sur lequel il travaille. Et un écran d'ordinateur tactile, pour tout ce qui est sélection dans l'interface de génération de l'algorithme génétique.

1.2.3 Bases méthodologiques

Le logiciel existant repose sur beaucoup de technologie, le coeur du programme est en C++ avec l'utilisation du framework Qt dans sa version 4.8.1, les zones d'affichages d'images 2D et 3D sont rendu grâce à OpenGL et CUDA. La lecture des fichiers source(DICOM) est assurée par la librairie Insight Toolkit. Et enfin certain des widgets de l'application comme la zone d'affichage de la fonction de transfert font appel à une bibliothèque de widgets pour Qt, QWT.

FIGURE 1.3 – Technologies utilisées dans StereoApp

Les différentes vues respectent le modèle MVC. Le gestionnaire de version SVN sera utilisé au cours du projet.

1.3 Description générale

1.3.1 Environnement du projet

Le logiciel StereoApp est conçu pour Windows XP et supérieur. Il peut être utilisé sur un écran 3D avec des lunettes actives, mais cela n'est pas une obligation la 3D stéréoscopique peut être désactivé dans les options du logiciel. Le logiciel sera aussi compatible avec les écrans tactiles. L'environnement d'utilisation idéal comprends un écran HD 3D de taille supérieur ou égale à cent centimètres de diagonales, et un écran d'ordinateur de vingt-quatre pouces tactile. Le grand écran sera utilisé pour afficher la partie principale de l'interface du logiciel à savoir notre représentation 3D manipulable, sur laquelle ou pourra voir tous les détails nécessaire à une analyse. L'écran secondaire tactile servira quant à lui lors de la phase de sélection d'individus lors d'une génération de l'algorithme génétique.
Ce projet ne dépends d'aucun autre projet de fin d'études.

1.3.2 Caractéristiques des utilisateurs

Les utilisateurs du logiciel sont des médecins cherchant à analyser précisément et rapidement le résultat d'un IRM. Ils veulent pouvoir rapidement accéder aux informations qu'ils recherchent et ne sont pas habitués à des interfaces lourdes peu claires, la réactivité sera cruciale. Il leur faut une interface simple, stable avec une visibilité importante du modèle principal. Actuellement le logiciel souffre de temps de latence important selon l'action effectuée. Des pop-up d'aide seraient les bienvenues pour guider l'utilisateur au travers des outils, notamment le nouvel outil de sélection dans les images de coupes 2D.

- Connaissance de l'informatique bureautique standard.
- Très peu d'expérience sur le logiciel.
- Utilisateurs très réguliers de l'application.
- Pas de notion de gestion de droit utilisateur.

1.3.3 Fonctionnalités et structure générale du système

Grâce au logiciel l'utilisateur doit pouvoir :

- Ouvrir un fichier volumique (.raw) et le visualiser en 3D ;
- Déplacer, zoomer, faire pivoter le modèle 3D
- Charger/Sauvegarder, interagir avec des fonctions de transfert
- Lancer et paramétrer l'algorithme génétique
- Faire une sélection et relancer l'algorithme génétique avec de nouveaux parents.

1.3.4 Contraintes de développement, d'exploitation et de maintenance

Contraintes de développement

- Nécessite une carte graphique supportant l'API CUDA de NVidia http://www.nvidia.fr/object/cuda_gpus_fr.html
- Ajout du support tactile pour l'utilisation futur d'écran tactile
- Langages de programmation imposé : C++, ainsi que différentes librairies Qt, ITK, QwT, OpenGl, Cuda.
- Environnement de développement Windows imposé à cause de Microsoft Visual Studio 20XX
- Algorithme génétique imposé
- Délai de réalisation de huit mois
- Migration du projet sur une autre machine plus puissante

1.4 Description des interfaces externes du logiciel

1.4.1 Interfaces matériel/logiciel

Il est nécessaire de rappeler que le logiciel à besoin de données 2D respectant le format DICOM en entrée pour produire une représentation 3D. Il est donc essentiel que l'IRM produise un fichier de sortie respectant ce standard.
Le logiciel permet l'affichage du modèle en 3D, bien que optionnel si l'on veut profiter de cette fonctionnalité il est nécessaire d'avoir un écran compatible NVidia Vision 3D, ainsi que des lunettes actives NVidia, et bien sûr une carte graphique supportant le jeu d'instructions CUDA.
Pour pouvoir profiter de l'application dans les meilleurs conditions il est fortement recommandé d'avoir au moins un périphérique de type souris ou pavé tactile, de manière à manipuler facilement le modèle 3D.

1.4.2 Interfaces homme/machine

Nous continuerons à utiliser la même politique de messages d'erreur, à savoir que toutes les situations anormales mais prévisible de fonctionnement seront affichées à l'écran dans une QMessageBox : :error décrivant au mieux le problème rencontré.
Pour mieux accompagner l'utilisateur lors de démarche "lourdes", il a été choisi d'utiliser des wizards pour guider celui-ci, comme dans le cas de l'utilisation de l'algorithme génétique.

La fenêtre principal est un ensemble de dock widget, tout ajout de nouveau panneau doit respecter cela (hériter de QDockWidget). Dans le but de créer une expérience utilisateur uniforme, les dock widgets permettant de déplacer les différentes fenêtre tout en réorganisant les autres de façon optimal pour un meilleur affichage.

Tout changement graphique de l'ordre des couleurs, des polices, espacement ou autre taille de bordures doit obligatoirement passer par un fichier CSS, unique à l'application.

1.5 Architecture générale du système

Le projet StereoApp est découpé en plusieurs parties :
- Cuda : Classes gérant l'affichage 3D tirant partie du GPU
- DataLoader : Classes chargeant en mémoire les données liées au modèle
- Exception : Classes gérant les exceptions personnalisées
- IHM : Classes gérant l'affichage d'interfaces utilisateur

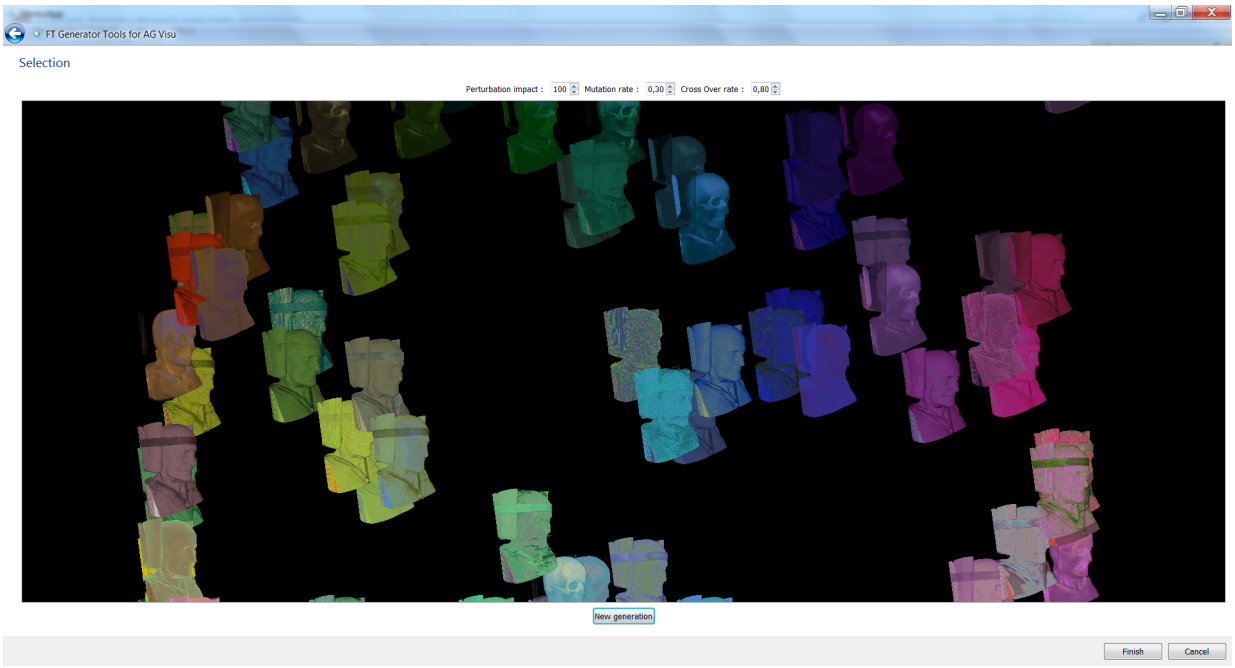


FIGURE 1.4 – Interface d'interactions avec l'algorithme génétique

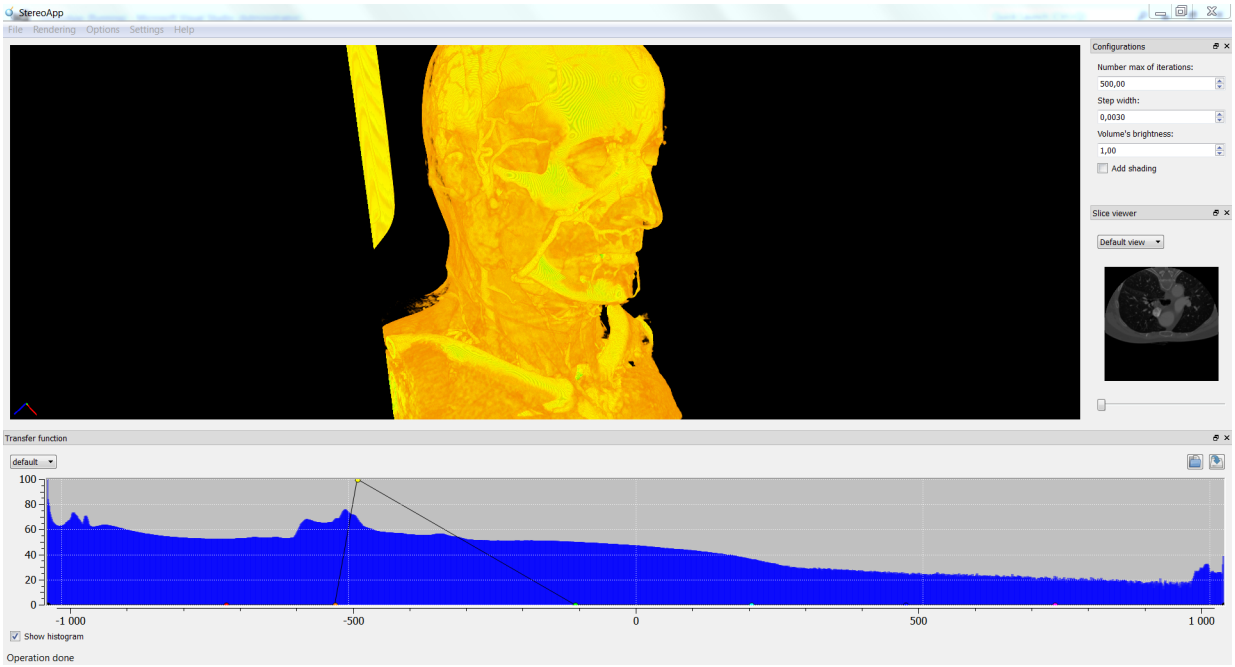


FIGURE 1.5 – Interface principale

- Interaction : Classes gérant les périphériques claviers/souris en rapport avec l'IHM
- IO : Classes gérant la lecture/écriture de fichiers (principalement pour lire les différents formats d'images)
- Math : Classes gérant les vecteurs, matrices à trois dimensions, projection...
- Object3D : Classes représentant un objet 3D
- Rendering : Classes contenant la caméra pour la vue 3D et gérant les différents éléments liés à la projection de notre scène.

- Thread : Classes utilisées pour gérer les threads de l'application
- Utils : Classes gérant la mémoire
- XML : Classes pour la gestion des fichiers XML

Les modifications à apporter au logiciel ne concerneront pas tous les modules. En effet, nos deux tâches principales touchent toutes deux aux interfaces de l'application. Par conséquent, nous allons plutôt nous concentrer sur les classes du module IHM. Il va falloir identifier les classes concernées par l'affichage des données 2D de coupes, pour pouvoir rajouter des outils de sélection. Ainsi que les classes affichant le résultat des itérations de l'algorithme génétique, pour venir intégrer notre nouvelle vue dans l'existant.

1.6 Description des fonctionnalités

Dans cette partie nous allons détaillé toutes les fonctionnalités qui seront rajouter dans le cadre du projet de fin d'étude.

1.6.1 Restriction de l'histogramme

Identification

- Restriction de l'histogramme sur une partie des coupes 2D
- Cette fonction est une évolution suite aux tests réalisés par Mélodie. En effet, le fait de pouvoir restreindre l'histogramme à certaine valeurs représentatives d'une certaine zone de l'IRM, permettrait à l'algorithme génétique de proposer des solutions dans un espace plus réduit et donc de s'approcher du résultat attendu, en moins d'itération. En effet, il sera plus probable de générer un résultat proche de ce qu'attends l'utilisateur, celui-ci sera donc amener à faire moins de cycle génération/sélection.
- Primordiale

Description

L'algorithme prendra en entrée l'histogramme complet ainsi qu'un objet de la classe ZonesCoupeSelection, créée dans le but de stocker les différentes sélections faites par l'utilisateur, puisque celui-ci pourra ajouter ou enlever des zones de sélection via l'affichage des coupes 2D.
Il interagira ensuite sur l'histogramme pour dans un premier temps sauvegarder l'actuel, et le remplacer pour un nouveau, calculé uniquement sur les valeurs des zones des coupes 2D sélectionnées.

1.6.2 Nouvelle vue d'affichage des résultats

Identification

- FTWeatherVisualisation
- La nouvelle vue reprendra en partie des éléments de l'interface conçue par Mélodie, comme les fonction de mutation croisement etc... mais diffèrera de par le principe d'affichage. En effet, nous partons dans une optique de représentation d'un nombre beaucoup plus grand d'individus, mais lorsque ceux-ci ce superposent, nous les grouperons et permettront de faire un "zoom" sur ce groupe pour mieux le voir, ainsi nous espérons avoir une meilleur visibilité et lisibilité des résultats de l'algorithme génétique.
  Un groupe sera représenté par plusieurs éléments fils considérés comme représentants du groupe, agencés en forme de triangle et entourés d'un contour de couleur. Ce contour de couleur deviendra plus vif lors d'un passage de la souris sur le groupe, et le curseur de l'utilisateur changera d'icône pour un icône de zoom classique. Lors d'un clic pour ouvrir un groupe le curseur redevient normal et une fenêtre, presque aussi grande que celle de sélection, viens se placer en superposition et afficher le contenue du groupe. Une icône de croix sera placé en haut à droite pour pouvoir fermer le groupe. Celui-ci pourra également être fermé par un clic sur la fenêtre de sélection place en arrière plan. La sélection dans le groupe se fera comme dans l'écran de sélection.

– Primordiale

Description

La vue prendra lors de sa création en paramètre une liste des éléments à afficher. Elle permettra à l'utilisateur de pouvoir sélectionner selon les préceptes Windows différents éléments fils de la génération de l'algorithme génétique. Si un groupe est affiché à l'écran l'utilisateur pourra cliquer sur ce groupe et ainsi avoir un "zoom" sur ce groupe en particulier dans lequel il pourra sélectionner des éléments. Une fois fini l'utilisateur pourra retourner en arrière en "fermant" le groupe, ce qui gardera sa sélection active.

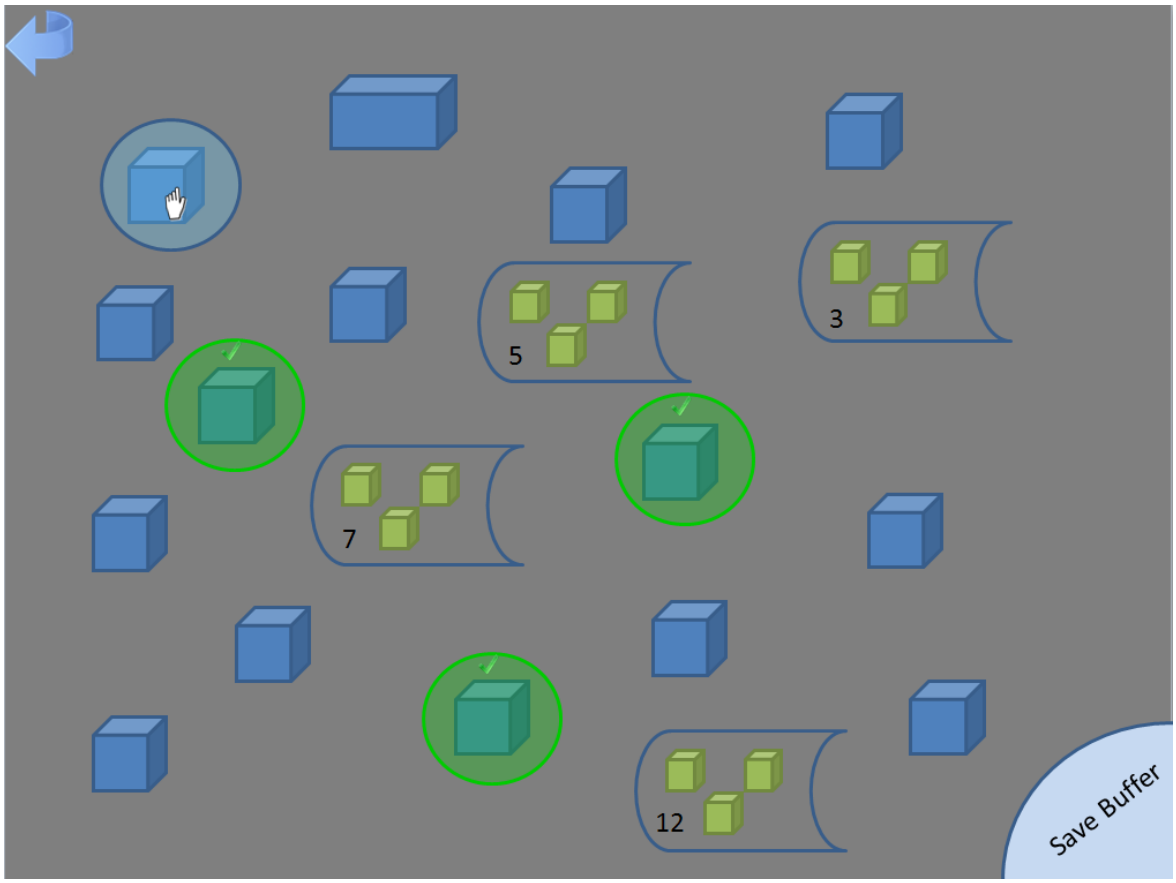


FIGURE 1.6 – Mockup de la weather view de base

Tout comme dans les deux vues précédentes, un clic sur un éléments le rends visible dans l'affichage principal, permettant de mieux le voir ainsi que de le manipuler.
La vue va donc devoir répartir les différents éléments, elle va pour cela devoir calculer la distance entre différentes images, et implémenter un des algorithmes parmi les choix proposer et vue avec les encadrants du projet (FishEye, Neighbourg, Arbre, Cercle).

1.7 Conditions de fonctionnement

1.7.1 Performances

L'affichage de la vue 3D et ses réactions lors de sa manipulation sont instantané, l'utilisateur va être amené à très souvent utiliser l'interface de visualisation de l'algorithme génétique, il va donc vouloir un temps de réponse instantané comme pour l'affichage 2D. Cependant dans la vue créer par Mélodie les

temps d'interactions sont de l'ordre de deux trois secondes. Il faudra analyser et prévoir des mesures pour contrer cet effet indésirable.

1.7.2 Capacités

Le logiciel est limité par le temps de génération d'une nouvelle population via l'algorithme génétique. Plus on va vouloir générer d'éléments plus la génération va être longue. Il est difficile de donner une estimation du nombre puisque cela va surtout dépendre du matériel. A l'heure actuelle dans l'interface proposer par Mélodie le système est déjà au dessus de ces capacités avec une génération de Décrire les limites des problèmes traitables par le système et les limites des éventuelles extensions comme par exemple :

- nombre max de terminaux ;
- nombre max de points d'acquisition ;
- nombre max de transactions simultanées de tel type, etc. ;
- capacité max de stockage ;
- taille max des données traitées ;
- etc.

1.7.3 Contrôlabilité

Les fichiers de log ont été désactivés suite au passage sur des systèmes plus récent il seront rétablies, ils logent tout le comportement de l'application, ainsi que les interactions utilisateurs.

1.7.4 Sécurité

Aucune gestion des utilisateurs ou de confidentialité dans le programme.

Plan de développement

2.1 Découpage du projet en tâches

Dans cette partie nous allons découper chaque tâches en sous tâches en faire une estimation temporelle qui nous donnera par la suite un diagramme de Gantt de départ qui servira de fils rouge tout au long du projet. Il sera par ailleurs intéressant de comparer ce diagramme théorique avec le diagramme de fin de projet réel pour analyser les écarts.

2.1.1 Prise en main de l'existant

Description de la tâche

Cette tâche consiste a se familiariser avec l'architecture et le code existant. Nous avons a notre disposition l'expertise des encadrants ainsi que les différents rapport et documents de projet de fin d'études précédent. Cette tâches est cruciale, on ne peut pas savoir comment intégrer des nouvelles fonctionnalités dans un logiciel dont on ne connait pas le fonctionnement.

Livrables

Ce cahier de spécification, sa clarté, ces explications et schémas techniques, sont un gage du travail réalisé pour prendre en main l'existant puisqu'il doit y faire référence de manière précise.

Estimation de charge

Estimation : 10 jours/homme

Contraintes temporelles

Contraintes temporelle forte, tant que l'analyse n'est pas effectuée et validée le développement ne peut être commencé. C'est une tâche difficilement quantifiable, dont la durée et la qualité de réalisation sont les fondations du projet.

2.1.2 Sélection dans les imagettes et restriction de l'histogramme

Description de la tâche

Cette fonctionnalité a pour but de restreindre les parties de l'histogramme et donc restreindre les possibilités de valeurs pour les fonctions de transfert et donc permettre une convergence plus rapide vers le résultat attendue par l'utilisateur.

Cycle de vie

Implémentation des fonctions :
- Fonction de détection de clic dans l'imagette
- Fonction de dessin du rectangle de sélection
- Fonction de sauvegarde des niveaux de gris sélectionné
- Fonction d'ajout/suppression/reset de la zone de sélection
- Modification graphique de l'IHM existante
- Fonction de calcul de l'histogramme selon les zones sélectionnées

Cette tâche ainsi que toutes ses fonctions devront être livrés en version finale, et faire l'objet de test unitaires.

Livrables

Livraison du composant logiciel de sélection dans la zone 2D ayant pour effet de restreindre les valeurs de l'histogramme.

Estimation de charge

Estimation : 11 jours/homme

Contraintes temporelles

Tâche signalée prioritaire par les encadrants.

2.1.3 Weather View

Description de la tâche

Conception d'une nouvelle interface graphique, affichant la population générer par l'algorithme génétique. Elle permettra la sélection/reset/ajout d'éléments dans la sélection courante. Elle supportera la création automatique et la gestion des groupes.

Cycle de vie

Implémentation des fonctions :
- Conception de la nouvelle zone d'affichage
- Intégrations des fonctions de croisement et de mutation de Mélodie
- Choix du rendu parmi les algorithmes suivant proposés par Mr Venturini (FishEye http://www.cs.umd.edu/class/fall2002/cmsc838s/tichi/fisheye.html, Neighbors, Arbre, Cercle)
- Fonction de calcul de distance entre deux images/volumes
- Fonction de placement et d'affichages des images
- Fonction de gestion de la sélection pour la génération suivante

Cette tâche ainsi que toutes ses fonctions devront être livrés en version finale, et faire l'objet de test unitaires.

Description des fonctions du cycle de vie

- Conception de la nouvelle zone d'affichage : squelette de base de la nouvelle vue, cette tâche comprends la création de la fenêtre en elle même, le support et l'affichage d'individus. L'ouverture de la fenêtre sera déclenchée comme précédemment via le menu dans la barre d'outils de la fenêtre principale.
- Intégrations des fonctions de croisement et de mutation : lorsque l'utilisateur clique sur le bouton de nouvelle génération, l'application fera appel aux algorithmes déjà en place pour gérer et effectuer les croisements et mutations nécessaires sur une population définie, et ce dans un temps raisonnable.
- Choix du rendu parmi les algorithmes : implémentation d'algorithmes de rendu et test de ceux-ci pour pouvoir sélectionner celui représentant le mieux les données. Cet algorithme gèrera l'emplacement des individus dans notre fenêtre de rendu. Il prendra une population d'individus et les placera sur la zone de dessin de la fenêtre.
- Fonction de calcul de distance entre deux images/volumes : maintenant que nous affichons les individus, il faut pouvoir détecter les superpositions d'individus et pouvoir les comparer pour garder les plus représentatifs du groupe et les afficher en entête du groupe dans la fenêtre de visualisation. Pour cela il faudra tester si un écart sur les valeurs de pixels ou de voxels est le plus approprié.

- Fonction de placement et d'affichages des images : fonction qui affichera les individus et les groupes d'individus formés sur la fenêtre de visualisation. Elle comprends aussi la logique d'ouverture et fermeture de groupe.
- Fonction de gestion de la sélection pour la génération suivante : gestion des interactions souris et tactiles pour la sélection d'individus. Un clic gauche sélectionne un individus, un clic gauche glissé sélectionne tous les individus touchant le rectangle de sélection, un maintien de la touche shift plus un clic gauche rajoute un élément à la sélection active. Un clic en dehors de tout éléments desélectionne la sélection courante.

Livrables

Livraison du composant logiciel de la nouvelle vue.

Estimation de charge

Estimation : 20 jours/hommes

2.1.4 Remise en place des fichiers de logs

Description de la tâche

Suite a une migration système l'ancien mécanisme de log est devenue obsolète. C'est un composant important et il doit être remis en place.

Cycle de vie

Implémentation des fonctions :
- Adaptation de la classe de log
- Remise en route des logs

Cette tâche ainsi que toutes ses fonctions devront être livrés en version finale, et faire l'objet de test unitaires.

Livrables

Livraison du composant logiciel de log.

Estimation de charge

Estimation : 1 jour/homme

2.1.5 Rédaction des rapports et différents documents

Description de la tâche

Rédaction de différents documents tout au long du projet. Document expliquant le processus de migration de l'application. Rédaction du rapport du projet de fin d'études. Et si il s'avère nécessaire de tout autre guide ou documents pouvant aider à expliquer le code et les fonctionnalités rajoutées.

Livrables

Tous les documents et rapport produits.

Estimation de charge

Cette tâche est transverse à toutes les autres et sera effectué en parallèle.

Contraintes temporelles

Certain document ont une date de rendu strict à respecter.

2.2 Planning

Tâche	Charge J/H	27/09/2012	29/11/2012	03/12/2012	06/12/2012	31/01/2013	07/02/2013	21/03/2013	26/03/2013
Prise en main existant	**10**								
Rédaction CdS	*2*								
Migration du logiciel	*8*								
Sélection dans les imagettes et restriction de l'histogramme	**9**								
Fonction de détection de clic dans l'imagette	*1*								
Fonction de dessin du rectangle de sélection	*1*								
Fonction de sauvegarde des niveaux de gris sélectionné	*2*								
Fonction d'ajout/suppression/reset de la zone de sélection	*2*								
Modification graphique de l'IHM existante	*2*								
Fonction de calcul de l'histogramme selon les zones sélectionnées	*1*								
Weather View	**17**								
Conception de la nouvelle zone d'affichage	*3*								
Intégrations des fonctions de croisement et de mutation de Mélodie	*4*								
Choix du rendu parmi les algorithmes proposés	*4*								
Fonction de calcul de distance entre deux images/volumes	*2*								
Fonction de placement et d'affichages des images	*2*								
Fonction de gestion de la sélection pour la génération suivante	*2*								
Remise en place des logs	**1**								
Adapter la classe de log existante	*0,5*								
Remettre en place du système de log	*0,5*								
Rédaction des rapports et différents documents	**37**			CdS V1			CdS Final		
Test	**5**								

2.3 Phase de test

Nous allons ici évoquer les principaux cas d'utilisation de l'application, qui serviront de cas de test, lors de nos différents développement.

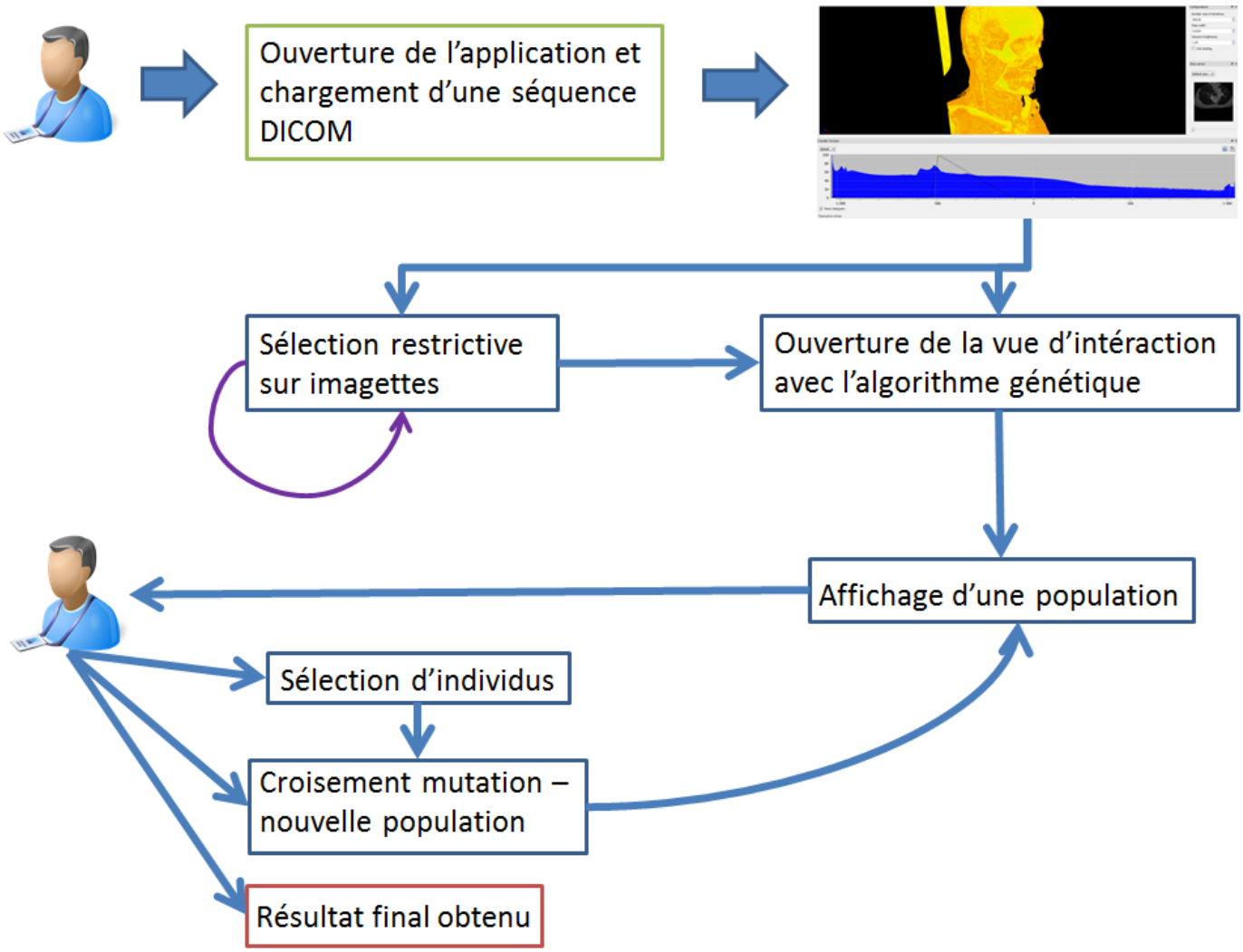


FIGURE 2.7 – Cas d'utilisation courant

Glossaire

FOVEA : Fouille visuelle de données et algorithmes biomimétiques
PFE : Projet de fin d'études
IHM : Interface Hommes-machines
IRM : Imagerie par résonance magnétique
MVC : Modèle/Vue/Contrôleur

Références

Rapport de **PFE** de **Sylvain Botto**
Rapport de **PFE** et de stage de **Nicolas Nathan**
Rapport de **PFE** et de stage de **Mélodie Roger**

Optimisation interactive pour la visualisation d'images médicales

Département Informatique
5e année
2012 - 2013

Rapport de Projet de Fin d'études

Résumé : Ce document décrit le travail effectué au cours de mon projet de fin d'étude sur le logiciel StéréoApp. Il décrit en détail la conception et la mise en place d'une nouvelle interface d'interactions avec un algorithme génétique. Elle repose sur le regroupement d'individus semblables, dans le but d'afficher un nombre important d'individus. Ainsi que l'ajout d'une sélection restrictive sur des coupes 2D d'IRM. Celle-ci permet de filtrer le modèle 3D et de cibler plus facilement les zones d'intérêt.

Mots clefs : Algorithme génétique - Fonction de transfert - Classification ascendante hiérarchique - Comparaison d'images

Abstract: This document is the synthesis of my end-of-studies project on the application "StéréoApp". It will describe in details the conception of a new graphical interface which enable the user to interact with a genetic algorithm. The idea behind this interface is to regroup look alike individuals, to help us show more individuals on the screen at the same time. It will also describe a new functionality, a restrictive selection on MRI 2D cuts. We believe this new feature will enable the user to target, quickly, useful areas.

Keywords: Genetic algorithm - Function transfer - Hierarchical clustering - Images distances

---

Encadrants
Barthélemy SERRES
barthelemy.serres@univ-tours.fr

Etudiant
Julien TERUEL
julien.teruel@etu.univ-tours.fr

Université François-Rabelais, Tours

DI5 2012 - 2013