PFE - Développement d'un outil de segmentation interactive basé superpixels

C. Bordes, A. Longuefosse, L-E. Pommé-Cassiérou, Z. Qiu

Clients : M. Mickaël Clément & M. Rémi Giraud

Chargé de TD : M. Pascal Desbarats

Master 2 Informatique
Université de Bordeaux

Mars 2020

Table des matières

Glossaire

Scribble/Pose de labels La pose de label désigne un processus venant s'appliquer sur la segmentation d'une image visant à accélérer la sélection d'objets dans une image par un utilisateur. Par exemple, dans le cas des superpixels, c'est un outil pertinent car il permettrait de sélectionner tous les superpixels de l'objet souhaité en une seule interaction. Dans ce but, une structure hiérarchique stockant plusieurs échelles de segmentation d'une même image permet de déterminer quels superpixels sont considérés comme faisant partie d'un objet ou non. Cependant, cette technique peut manquer de robustesse car elle est souvent dépendante des distances colorimétriques et spatiales, et peut donc ne pas fonctionner sur certaines images.. 5, 8, 30, 32

Superpixel Un superpixel est défini comme un ensemble de pixels dans une image. Les pixels associés à un même superpixel sont définis en tenant compte de deux propriétés : la distance spatiale et la distance colorimétrique. Dans l'idéal, tous les superpixels d'une image ne sont composés que de pixels proches à la fois en terme de distances et de couleurs.. 1, 3–16, 18, 20–27, 29, 30, 32

FIGURE 1 – Image `peppers.png` segmentée avec environ 300 superpixels

1 Contexte

1.1 Introduction

Ce projet est réalisé dans le cadre de l'UE *Projet de Fin d'Etudes* au sein du Master Informatique de l'Université de Bordeaux. Proposé par M. Mickaël Clément et M. Rémi Giraud, ce projet prend place dans le domaine du traitement d'images, et s'intéresse plus particulièrement à la reconnaissance d'objets dans une image. De nos jours, il existe de nombreuses méthodes de segmentation d'images efficaces permettant la détection et la reconnaissance d'objets dans une image. Cependant, les segmentations obtenues sont difficilement précises à l'échelle pixellique en raison de certains paramètres propres à l'image (contours, fréquences, intensités).

Le sujet de notre projet consiste à développer un outil de segmentation interactive basée superpixels. Utiliser une sous-représentation irrégulière de type superpixels afin de segmenter une image permet dans un premier temps, de réduire la complexité en temps et en mémoire de nos algorithmes de par le fait que le nombre d'éléments à traiter est moindre. De plus, ce type de sous-représentation s'adapte localement au contenu de l'image et permet ainsi d'obtenir une segmentation plus précise de l'image. Ainsi, étant donné un algorithme de segmentation d'images basé sur ce type de sous-représentation, il nous est demandé d'implémenter une interface graphique, en langage Qt ou JavaScript ou en implémentant un plugin Gimp, permettant à un utilisateur d'extraire un objet dans une image de manière efficace et intuitive.

Par conséquent, comme dans tout projet, nous avons commencé par étudier l'existant afin d'être en mesure de faire des choix techniques et de mieux nous approprier le sujet. Nous avons ainsi pu traduire les besoins et demandes des client sous la forme de User Stories qui sont décrites dans la deuxième partie, avec une présentation de notre logiciel. Ensuite, afin d'éclaircir le fonctionnement de notre logiciel, nous présentons son architecture, notamment à l'aide de diagrammes (classes, séquence), et certains algorithmes que nous jugeons nécessaires d'expliquer. Quant à la quatrième partie, elle est dédiée aux tests réalisés identifiant les parties fonctionnelles et non-fonctionnelles de notre programme. Enfin, avant de conclure sur les besoins remplis et non remplis et les améliorations possibles, nous décrivons le déroulement général de notre projet : méthode de suivi de projet, adaptation, organisation.

1.2 Étude de l'existant

Comme introduit ci-dessus, notre projet consiste à développer un outil de segmentation interactive utilisant une sous-représentation de type superpixels permettant à un utilisateur l'extraction d'objets dans une image. Pour l'étude de l'existant, nous allons donc différencier le noyau algorithmique de notre projet et l'interface graphique.

1.2.1 Algorithme de segmentation en superpixels

Tout d'abord, l'algorithme de segmentation utilisé est l'algorithme SLIC [1] (Simple Linear Iterative Clustering), algorithme qui nous a été introduit par les clients du projet. SLIC permet la segmentation d'une image en superpixels. En comparaison avec d'autres algorithmes de segmentation basés superpixels, SLIC apparaît comme le meilleur compromis entre efficacité en temps et en mémoire (complexité linéaire en fonction du nombre de pixels dans l'image et indépendante du nombre de superpixels), précision de l'adhérence aux contours de l'image et contrôle sur le nombre et la taille des superpixels.

Algorithme SLIC

Voyons maintenant les différentes étapes et méthodes de l'algorithme SLIC. Une première étape est l'initialisation des k centres de chaque superpixel, ou cluster, répartis de manière régulière sur l'image, k étant choisi par l'utilisateur. Les centres sont représentés en cinq dimensions (x,y,L,a,b) pour leurs coordonnées spatiales et la couleur du pixel dans l'espace *Lab*. Suit également l'initialisation des cartes de superpixels et de distances, le label `-1` est attribué à chaque pixel de l'image et chaque distance est initialisée à l'infini.

Ensuite, chaque pixel de l'image est associé au superpixel le plus proche. Pour cela, l'algorithme utilise une fonction de distance qui prend en compte à la fois la distance spatiale et la distance colorimétrique entre le pixel à assigner et un centre. Cette distance est pondérée par un coefficient choisi par l'utilisateur qui lui permet ainsi de contrôler la taille et la compacité des superpixels.

La complexité linéaire de l'algorithme vient alors de la façon dont sont calculées les distances. Pour chaque centre, l'algorithme se restreint à une zone de recherche proportionnelle à la taille des superpixels et calcule, non pas les distances entre un centre et tous les pixels de l'image mais uniquement les distances entre un centre et les pixels de la zone restreinte. Si la distance nouvellement calculée est plus petite que celle en mémoire pour un pixel de cette zone, elle est modifiée en mémoire et le pixel est associé au superpixel dont le centre est le centre traité. Puis, une fois que chaque pixel est associé au centre le plus proche, la position de chaque centre est mise à jour selon la valeur moyenne de tous les pixels du superpixel (couleur et position).

Ce procédé, calcul des distances et mise à jour des centres, est itéré un certain nombre de fois jusqu'à convergence de l'algorithme, autrement dit jusqu'à ce que le déplacement moyen des centres entre deux itérations soit inférieur à un certain seuil. Il est à noter que les auteurs jugent suffisantes dix itérations pour obtenir un résultat convenable.

Cependant, ces étapes n'assurent pas une connexité globale. Il est possible que certains pixels soient associés au superpixel le plus proche en terme de distances spatiale et colorimétrique mais se retrouvent entourés de pixels appartenant à un superpixel différent. Ces pixels, dits "isolés", rendent alors leur superpixel associé non connexe. Pour y remédier, une dernière étape de ré-assignation des pixels isolés au superpixel le plus proche est appliquée sur l'ensemble de l'image. L'algorithme est décrit ci-dessous en pseudo-code et provient de l'article référencé en [1] :

```
Initialize cluster centers Ck = [lk; ak; bk; xk; yk]T by sampling pixels at
    regular grid steps S.
Move cluster centers to the lowest gradient position in a 3 x 3 neighborhood.
Set label l(i) = -1 for each pixel i.
Set distance d(i) = infinity for each pixel i.
repeat
    for each cluster center Ck do
        for each pixel i in a 2S x 2S region around Ck do
            Compute the distance D between Ck and i.
            if D < d(i) then
                set d(i) = D
                set l(i) = k
            end if
        end for
    end for
    Compute new cluster centers.
    Compute residual error E.
until E <= threshold
```

Notre projet est basé sur une version de l'algorithme SLIC implémentée en langage C++ et utilisant les structures d'OpenCV. Cette version est disponible sur un dépôt Git [2].

Algorithme TurboPixels

En vue de comparer les performances de l'algorithme SLIC avec un autre algorithme de segmentation basé superpixels du même type, il nous faut présenter brièvement l'algorithme TurboPixels [3], publié en 2009. Comme dans l'algorithme SLIC, k centres sont initialisés et répartis de manière régulière sur l'image. Les superpixels croissent ensuite jusqu'à ce que leur vitesse d'évolution soit en deçà d'un seuil et que l'image soit complètement segmentée. Une fois le traitement terminé, la connexité globale à l'image est ici garantie. Toutefois, en comparaison de SLIC et même si la segmentation obtenue est acceptable, TurboPixels a une vitesse d'exécution très lente. De plus, la segmentation obtenue adhère rarement aux contours de l'image.

1.2.2 Interface graphique

Javascript

Concernant l'interface graphique, les clients nous ont fourni le lien vers une interface existante implémentée en JavaScript [4], basée non pas sur des superpixels mais sur une hiérarchie morphologique à partir des arbres couvrants minimums (approche watershed [1]). Cette interface fournit à l'utilisateur la possibilité d'extraire un objet dans une image par pose de labels.

Les résultats obtenus sont convaincants mais pour certaines images dont les contours ne sont pas fortement marqués, obtenir le résultat souhaité n'est pas immédiat en raison de clusters de forme et taille irrégulières dont l'adhérence aux contours n'est pas précise. De plus, l'algorithme de segmentation utilisé dans ce travail n'offre aucun contrôle à l'utilisateur sur le nombre et la taille des clusters, contrairement au logiciel demandé par le client basé sur les superpixels. Il pourrait donc être envisageable de s'inspirer de ce travail concernant l'extraction d'objets par pose de labels et ajouter des fonctionnalités visant à améliorer l'interaction utilisateur.

Plugin Gimp

Gimp est un logiciel d'édition d'image libre et gratuit, disposant d'une API pour intégrer des scripts et plugins. Le principal avantage d'utiliser Gimp est que l'interface graphique est déjà entièrement réalisée, et il suffirait donc d'intégrer les algorithmes relatifs aux superpixels (segmentation, zoom, sélection, fusion..) dans des plugins.

Qt

Qt est une API multi-plateformes développée en C++ dont la phase de compilation se veut simplifiée par le biais du programme *qmake*. Qt offre notamment des composants d'interface graphiques (Widget, Label) et le logiciel *QtDesigner* permet la création d'interfaces graphiques Qt de manière efficace. Un autre point positif concernant l'utilisation de Qt dans un projet de ce type est la communication entre différentes classes via l'utilisation de signaux et slots, particulièrement adaptés aux architectures de type Modèle-Vue-Contrôleur.

2 Analyse des besoins

Sur la base des algorithmes déjà existants et utilisables dans notre projet, il nous faut à présent détailler l'ensemble des fonctionnalités demandées par nos clients, et voir comment elles ont été réalisées. Nous avons formalisé les attentes de nos clients sous la forme de User Stories. Leur priorité de réalisation a été déterminée de la manière suivante : si une fonctionnalité comme un algorithme par exemple est au coeur du logiciel, alors sa priorité est importante. Si au contraire, la fonctionnalité n'a aucun lien direct avec un des algorithmes centraux, alors sa priorité est faible.

2.1 User stories

Afin de formaliser le plus correctement possible les User Stories, nous nous sommes posés la question suivante : "En tant qu'utilisateur, qu'est ce que je veux pouvoir faire sur l'outil développé" ? Chaque point de la liste suivante doit être un élément de réponse à cette question. Ces points sont listés dans un certain ordre permettant à quelqu'un découvrant l'outil de mieux le comprendre.

Segmentation d'une image en superpixels

- L'outil doit proposer un moyen de segmenter des images quelconques en superpixels. Ce point implique de proposer un moyen de visualiser des images, ainsi qu'un moyen de les segmenter.

- Parmi les différentes méthodes qui existent pour segmenter une image, l'outil doit proposer la méthode SLIC.

- Afin d'avoir une vision nette de la segmentation d'une image, il nous a été demandé de réaliser une double vue d'une image côte à côte. Dans la vue de gauche, il doit être possible de visualiser l'image choisie, avec les contours de la segmentation calculée appliqués sur l'image. Dans la vue de droite, il doit être possible de ne visualiser que la carte de segmentation, à savoir, une image blanche de la même taille que celle de gauche, mais avec les contours appliqués par-dessus.

- L'algorithme SLIC dispose de paramètres afin de segmenter une image, comme le nombre de superpixels à générer ou le poids des distances colorimétriques. Ce dernier paramètre doit permettre de déterminer l'appartenance des pixels de l'image à tel ou tel superpixel en attribuant une priorité à la distance spatiale ou colorimétrique suivant les cas. Un poids faible privilégiera la distance colorimétrique et permettra l'adaptation forte de la segmentation aux contours. A l'inverse, si la distance spatiale est privilégiée, elle aura tendance à créer des grilles rectangulaires pratiquement régulières. L'outil développé doit fournir un outil de contrôle des paramètres, par exemple une barre de défilement (ou "slider") pour le nombre de superpixels à générer, et une autre pour le poids des distances colorimétriques. A chaque changement de la valeur de l'un des sliders, l'algorithme SLIC doit être appelé à nouveau sur l'image d'entrée, pour recalculer une nouvelle segmentation.

Détourage des éléments d'une image avec une précision plus ou moins fine

- Un des objectifs futurs de notre outil étant de pouvoir détourer des éléments d'une image, celui-ci doit disposer d'un mécanisme de sélection des superpixels. Une telle sélection doit pouvoir être réalisée sur l'image de référence (vue de gauche) ou bien sur la carte de segmentation (vue

de droite) de deux manières différentes. La première de ces options est un simple clic sur un pixel appartenant à un superpixel. La seconde est une action de type "cliqué-glissé" depuis un superpixel jusqu'à un autre superpixel. L'ensemble des superpixels parcourus (sur le trajet de la souris) doivent être considérés comme faisant partie de la sélection. Cette sélection doit apparaître sur les deux vues sus-mentionnées en conservant les couleurs de l'image de référence. En conséquence, sur la vue de gauche, sur laquelle l'image est déjà entièrement visible, les superpixels sélectionnés doivent apparaître plus ou moins sombres, et leurs contours affichés dans une autre couleur. Dans le cas de la vue de droite, les pixels sélectionnés peuvent seulement être des copies des zones correspondantes de l'image sachant que l'on vient copier sur un fond blanc uni.

- Le point précédent implique la nécessité de disposer d'un outil de dé-sélection des superpixels fonctionnant de la même façon que l'outil de sélection.

- Certaines zones de l'image étant parfois fines, il peut être avantageux de les agrandir. C'est une des exigences qui est nécessaire pour l'ajout d'une fonctionnalité plus importante encore. L'agrandissement doit pouvoir se faire grâce à la molette de la souris et doit, dans un cas agrandir la zone de l'image autour du pixel pointé par la souris, dans l'autre cas rétrécir l'image de référence. Dans les deux cas, seule la vue de gauche est agrandie ou rétrécie.

- L'utilisation de l'outil de zoom ne doit empêcher la sélection de superpixels ni sur la vue de gauche agrandie ni sur la vue de droite entière. La sélection doit tenir compte du fait que l'image peut être agrandie toutefois.

Segmentation adaptative d'une image en superpixels

- Une des fonctionnalités les plus importantes attendue ici est le fait de pouvoir calculer une nouvelle segmentation sur une zone plus restreinte de l'image. En effet, dans certaines zones, la première segmentation est souvent incorrecte localement et ne permet pas un détourage précis de l'objet. En conséquence, tout l'intérêt de l'outil est de pouvoir regénérer localement cette segmentation. Une fois l'image agrandie, un bouton doit permettre de générer une nouvelle segmentation sur la région d'intérêt, soit la partie zoomée de l'image de référence.

- La nouvelle segmentation peut être recalculée à n'importe quel niveau de zoom. Cependant, si le but est d'améliorer la qualité de la première segmentation à l'échelle de toute l'image, il faut pouvoir disposer d'un moyen de fusionner deux cartes de segmentation et de mettre à jour la carte initiale de droite. Pour savoir quand calculer l'opération de fusion, il faut savoir avant quelles sont les zones à fusionner. Pour cela, l'outil de sélection est utilisé. Sur la fenêtre de zoom, l'utilisateur peut sélectionner des superpixels un par un ou plusieurs à la fois de la même manière que précédemment. Toutefois, la sélection doit avoir pour effet ici de fusionner la nouvelle carte de segmentation, restreinte à la nouvelle sélection de l'utilisateur uniquement, avec la carte d'origine (cf partie 3.3.3). Comme notre client considère que la nouvelle sélection est supposée plus précise, c'est donc elle qui doit primer sur l'ancienne dans la fusion des cartes de segmentation. La fusion doit absolument modifier celle d'origine une fois que l'utilisateur a achevé sa sélection. Il doit toutefois pouvoir réaliser la fusion en plusieurs étapes, et tant qu'il ne modifie pas le niveau de zoom, il doit pouvoir continuer à fusionner les cartes de segmentation.

- L'effet de ce précédent point doit cependant pouvoir être annulé si l'utilisateur dézoome complètement l'image. Il doit pouvoir en ce cas manipuler à nouveau la sélection dans l'image principale sans que la fusion ne rentre plus en ligne de compte.

- Comme il peut s'avérer pénible de sélectionner à nouveau une potentielle grande zone de l'image

après l'avoir agrandie et après avoir appuyé sur le bouton recalculant une nouvelle segmentation, il nous a été demandé de propager l'éventuelle sélection (c'est-à-dire l'ensemble des superpixels sélectionnés) à chaque niveau d'échelle. Ceci signifie de rechercher la correspondance entre "anciens" superpixels et "nouveaux" superpixels et de sélectionner les nouveaux superpixels correspondants.

Fonctionnalités favorisant l'ergonomie du logiciel

- L'interface doit disposer d'un moyen de réinitialiser la sélection pour éviter de devoir tout désélectionner manuellement. Cette action peut se faire via un bouton par exemple.

- Quels que soient les changements appliqués sur l'une des vues de gauche ou de droite, les mêmes changements doivent être appliqués à l'autre vue pour qu'elles restent systématiquement cohérentes l'une avec l'autre (dans le cas de la sélection, de la fusion des cartes de segmentation par exemple).

- Pour aider à la compréhension de l'interface et des modifications appliquées, l'ajout d'un deuxième curseur nous a été demandé, celui-ci devant copier le curseur de l'utilisateur lors de la manipulation de l'image de gauche au minimum. Ce second curseur doit respecter les contraintes imposées par le zoom.

- Toujours dans l'idée d'aider à la compréhension de l'utilisateur, des informations sur le pixel courant (position, couleur, superpixel associé) doivent être affichées et modifiées instantanément selon la position du curseur de la souris.

- L'outil doit disposer d'un outil d'exportation de la zone sélectionnée par l'utilisateur.

- Le détourage de l'objet doit pouvoir se faire de manière efficiente, en minimisant les interactions utilisateurs.

Extensions du logiciel

- L'outil doit laisser la possibilité d'ajouter plus tard d'autres algorithmes de segmentation (autres que SLIC) afin de pouvoir comparer leurs performances respectives.

- Le détourage manuel d'objets pouvant être laborieux selon les images traitées, un outil alternatif permettant une sélection plus efficace doit être implémenté : la sélection par pose de labels.

2.2 Choix des outils

Concernant les outils et langages à utiliser, nous devions faire un choix entre Javascript, Gimp et Qt, sachant que l'algorithme fourni pour le calcul des superpixels, SLIC, est trouvable en Python, C ou C++.

Les inconvénients du Javascript sont la difficulté d'adaptation pour des grands programmes et le manque de performance. De plus, il aurait fallu mettre en place un wrapper pour utiliser le code fourni en tant que bibliothèque dynamique, et ce choix ne semblait pas correspondre avec la courte durée du projet.

Le plugin Gimp semblait être une idée intéressante pour éviter de perdre du temps sur la création de l'interface graphique. Le but du projet étant de servir pour de la recherche dans le traitement

d'image en comparant différents algorithmes de segmentation, cela reviendrait soit à créer plusieurs plugins, ce qui rend difficile l'interaction et la comparaison, soit à créer un seul plugin qui rassemble tous les algorithmes. Le problème de cette approche est que l'utilisation d'un plugin n'est plus du tout justifiée, celui-ci représentant seulement un module et non une application à part entière.

Notre choix s'est donc porté sur le framework Qt, car il présente des outils de développement *Qt-Creator* et *QtDesigner* adaptés à la création d'interfaces, une large documentation et des performances viables pour un rendu fluide.

Le choix du langage s'est porté sur le C++, qui permet la gestion des exceptions, la surcharge de fonctions, et l'utilisation de classes adaptées à notre projet. De plus, ce langage s'associe parfaitement avec le framework Qt, ainsi qu'avec la bibliothèque OpenCV, qui facilite les opérations sur les matrices et est utilisée dans la base de code fournie en C++.

2.3 Présentation de l'outil

Segmentation de l'image

La version finale du logiciel s'organise autour d'une seule fenêtre. Au lancement du logiciel, une image est ouverte par défaut. L'utilisateur peut également choisir une image en cliquant sur `File` puis `Open`. Cette image est redimensionnée de sorte que la fenêtre soit inférieure à une dimension fixée (1600x900) pour éviter des problèmes d'affichage. L'image est ensuite segmentée avec l'algorithme basé sur SLIC et avec les paramètres définis par les sliders : Nombre de superpixels et poids des couleurs. La segmentation des superpixels peut être mise à jour en faisant varier les sliders, moyennant un léger temps de calcul (cf tests de performance, Partie 4.2.2). Les sliders n'ont d'effet que lorsqu'ils sont relâchés afin d'éviter des temps de calcul trop importants.
En bas à gauche de la fenêtre se trouvent des informations sur le pixel de l'image où est positionné la souris (position, valeurs RGB, couleur, et label du superpixel parent).

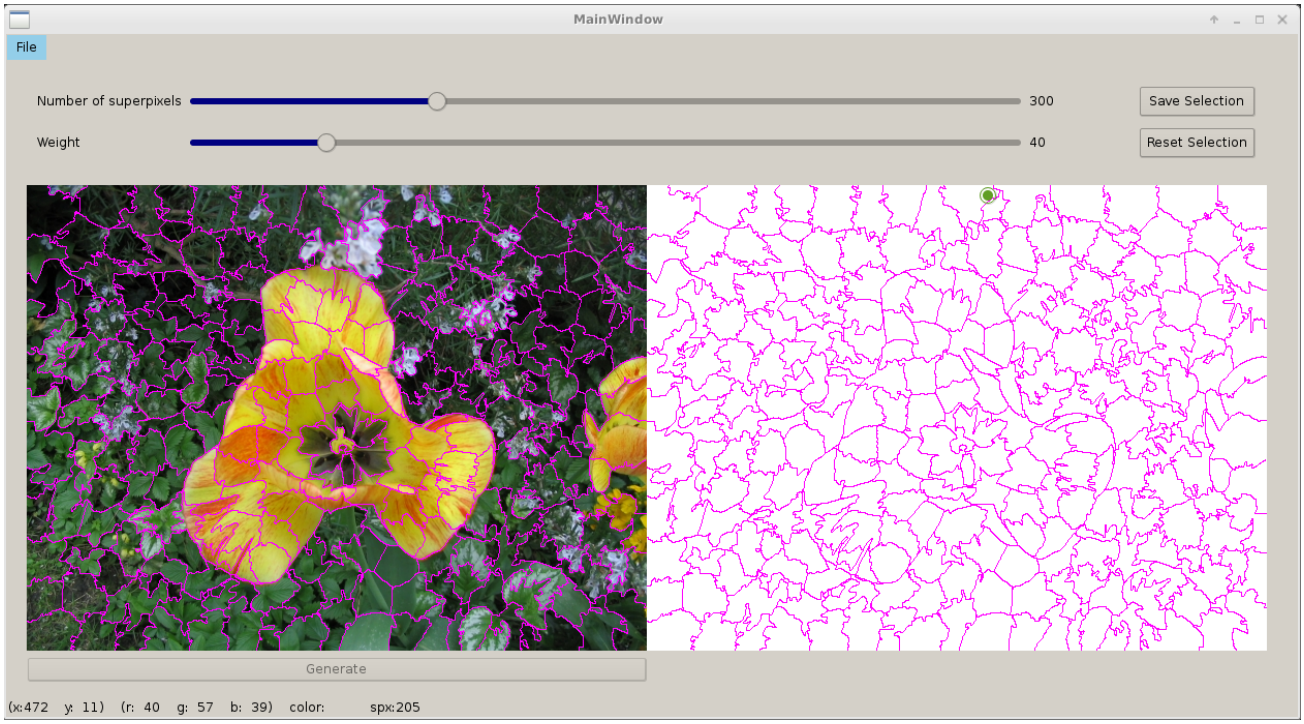

FIGURE 2.1 – Logiciel après ouverture d'une image

Sélection des superpixels

Après la génération de la carte de superpixels, l'utilisateur peut sélectionner des superpixels pour détourer l'objet souhaité. La sélection se fait sur l'image de gauche (un curseur symétrique s'affiche en même temps sur l'image de droite pour aider à la sélection) ou sur l'image de droite avec le clic gauche (simple clic ou clic + déplacement souris), et la dé-sélection avec le clic droit. Les superpixels sélectionnés sont alors affichés sur l'image de droite en plus de la carte de superpixels, et sont grisés et leurs contours affichés d'une couleur différente sur l'image de gauche.
Enfin, il est possible de supprimer entièrement la sélection en appuyant sur le bouton `Reset Selection`, ou de sauvegarder l'image détourée avec la bouton `Save Selection`.

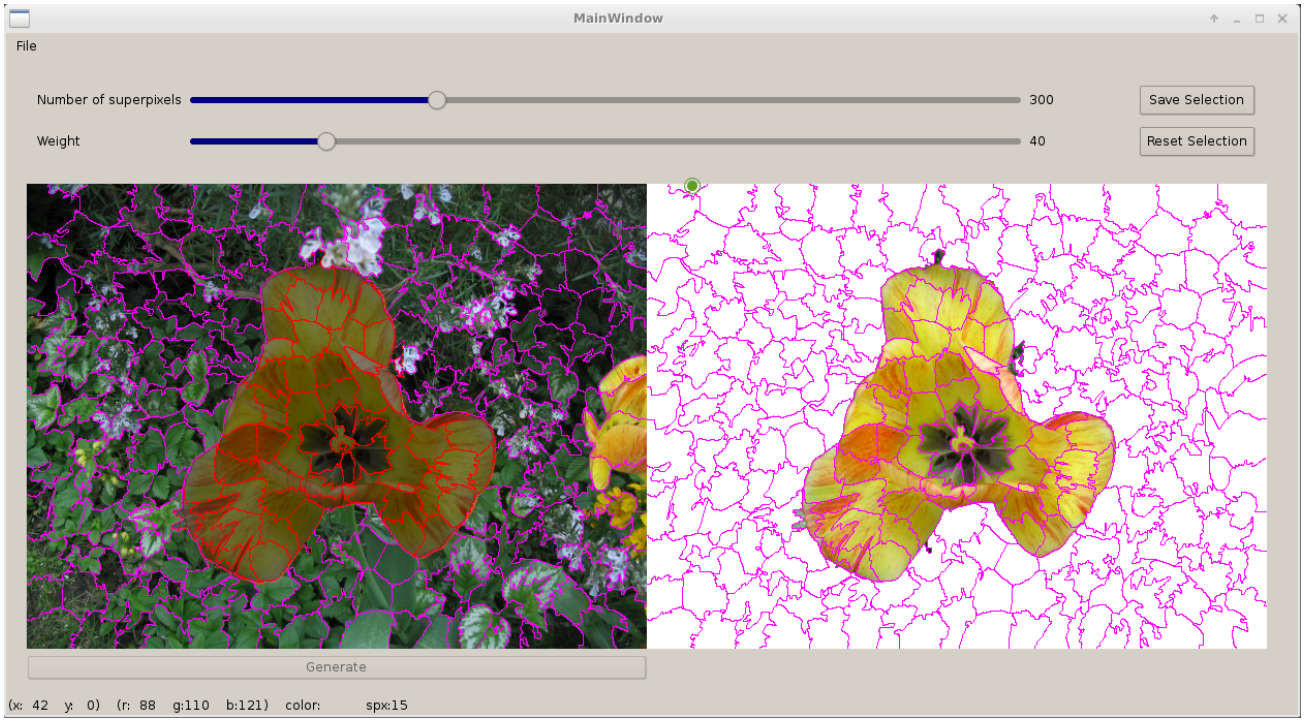

FIGURE 2.2 – Sélection grossière des superpixels de l'objet

Zoom

Il est ensuite possible de zoomer sur une partie de l'image avec la molette de la souris (vers l'avant pour zoomer et vers l'arrière pour dé-zoomer) pour voir plus précisément les superpixels sélectionnés et ainsi déterminer ceux qui sont corrects et ceux qui nécessitent un raffinage.

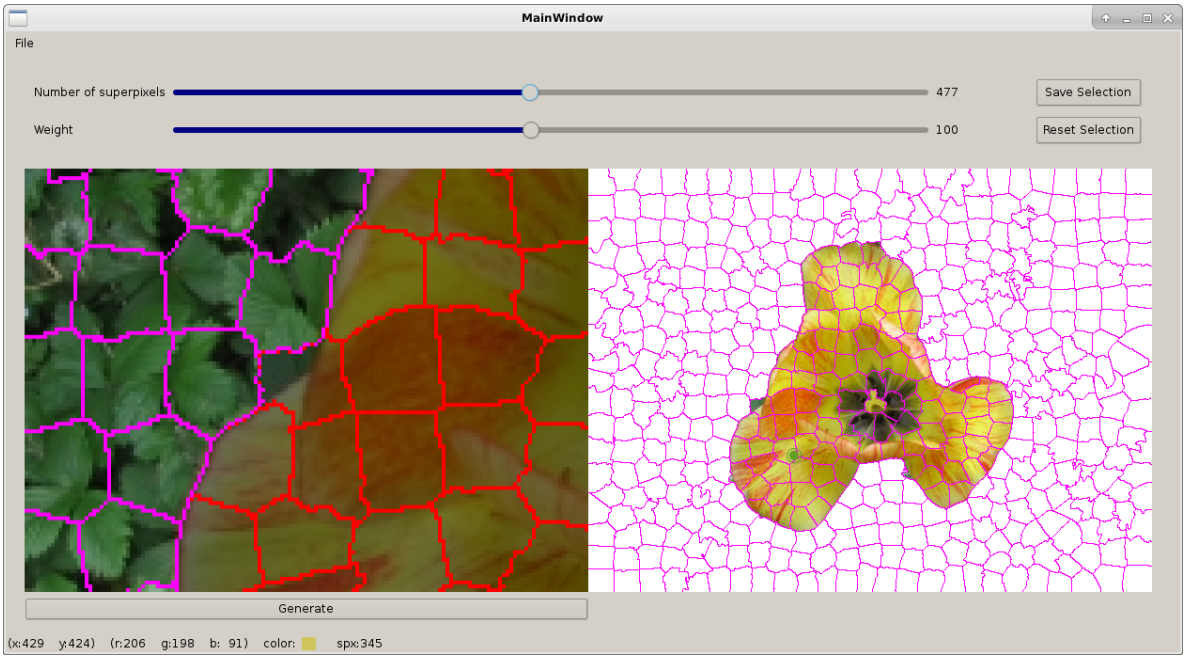

FIGURE 2.3 – Zoom sur une zone sélectionnée qui nécessite un raffinage

Après un clic sur le bouton `Generate` situé sous l'image de gauche, une carte de superpixels va être recalculée sur la partie zoomée de l'image, ce qui va permettre une segmentation plus précise. De plus, la sélection des superpixels précédents est propagée sur la nouvelle carte, pour éviter de devoir sélectionner à nouveau toute cette partie.

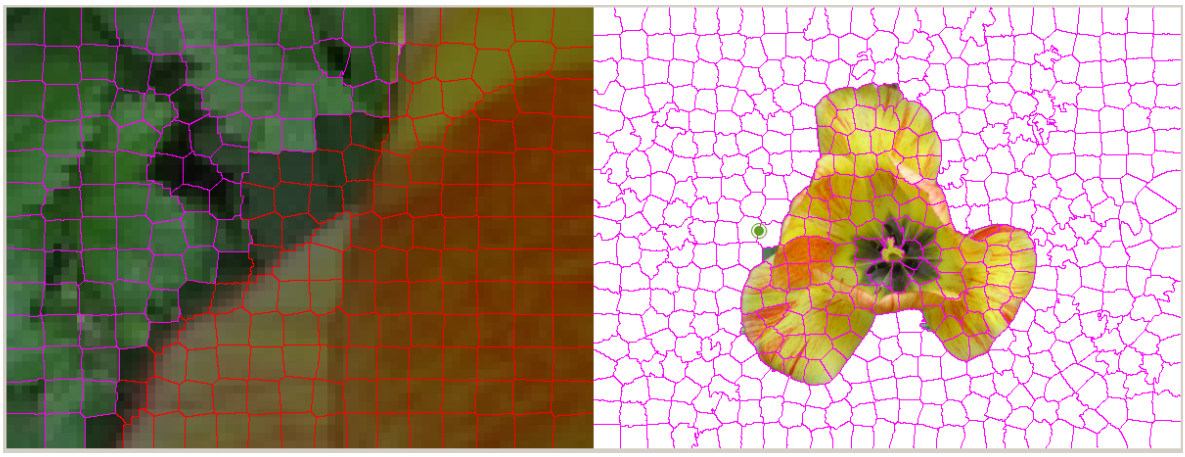

FIGURE 2.4 – Génération de nouveaux superpixels

Fusion des superpixels

Il est possible de sélectionner ou dé-sélectionner des superpixels sur la nouvelle carte de superpixels (image de gauche zoomée), ce qui va directement mettre à jour la carte de segmentation initiale (image de droite), en fusionnant les anciens superpixels avec les nouveaux sélectionnés/dé-sélectionnés.

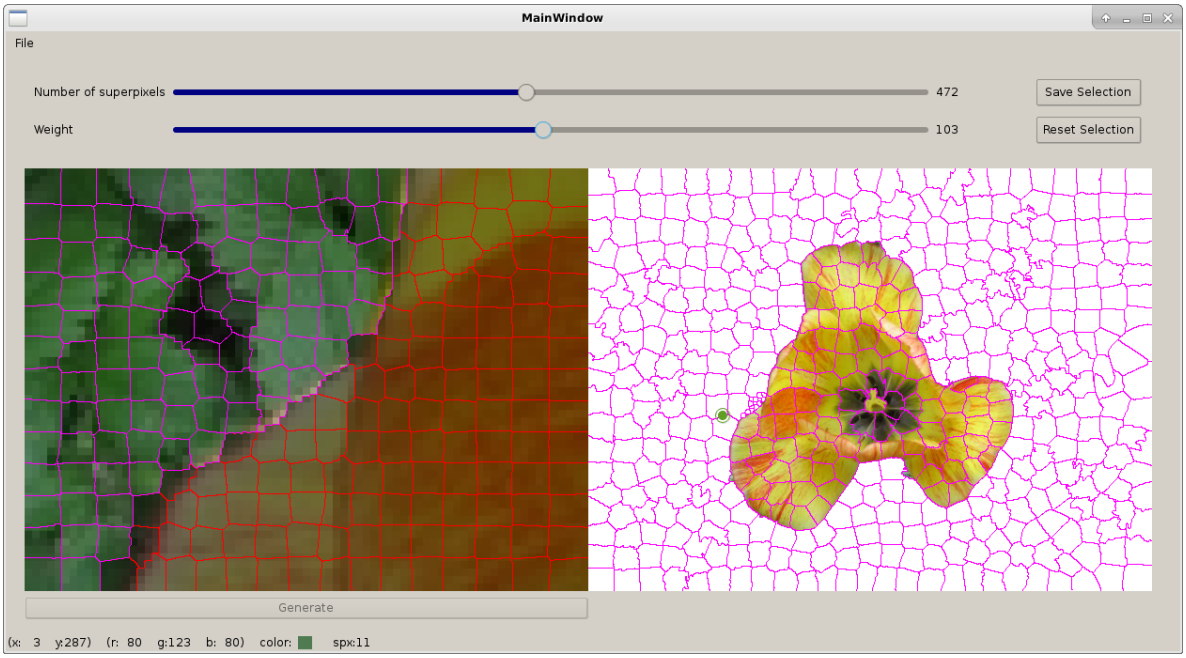

FIGURE 2.5 – Dé-sélection des superpixels (à gauche) et fusion des cartes de superpixels (à droite)

Enfin, lorsque toutes les zones ont été raffinées, le résultat est sauvegardé en appuyant sur `Save Selection`. La Figure 2.6 montre le résultat avant et après le détourage de la fleur.

FIGURE 2.6 – Image de départ (gauche) et image sauvegardée après détourage (droite)

3 Architecture et implémentation

L'architecture de notre logiciel s'articule autour de trois classes principales. Une classe `MainWindow` contient l'interface homme-machine (IHM) et reçoit les interactions de l'utilisateur. En fonction de ces interactions (sélection, dé-sélection, zoom), une classe `ClickableLabel` héritant de la classe `QLabel` de Qt et dont une instance appartient à la classe `MainWindow` a pour rôle l'affichage et la mise à jour des images traitées. Enfin, une classe appelée par `ClickabelLabel` contient l'implémentation de tous les algorithmes de traitement effectués sur les images, dont ceux fournis par les clients, et correspond au noyau algorithmique : la classe `Slic`.
Les différents liens d'appartenance et interactions entre chacune de ces classes seront illustrés par la suite à l'aide de diagrammes.

3.1 Pattern MVC

Notre logiciel se base sur une architecture Modèle-vue-contrôleur (MVC). Cette implémentation permet de séparer les données, la vue utilisateur et la gestion des évènements pour faciliter la maintenance et les évolutions futures.

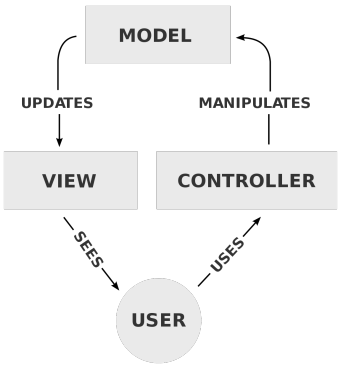

FIGURE 3.1 – Modèle architecture MVC

Modèle [`slic.cpp, slic.h`]

Notre modèle repose sur la structure de données `Slic()`, qui contient toutes les données relatives aux superpixels (Lien entre pixels de l'image et clusters, labels des clusters, clusters sélectionnés).

Vue [`mainwindow.cpp, mainwindow.h, mainwindow.ui`]

La vue est gérée par la classe `MainWindow`, elle va permettre d'afficher les données du modèle et de recevoir les actions de l'utilisateur (ouverture d'image, sliders des paramètres de l'algorithme Slic, boutons de l'interface).

Contrôleur [`clickablelabel.cpp, clickablelabel.h`]

Le contrôleur correspond à la classe `ClickableLabel` et va permettre la synchronisation du modèle et de la vue, en appliquant des modifications sur les données et en demandant la mise à jour de la vue. Son fonctionnement est détaillé dans la partie 3.3.1.

3.2 Diagrammes UML

3.2.1 Diagramme de classes

Le diagramme de classes (Figure 3.2) permet de montrer la structure interne du logiciel, notamment les interactions entre les différents objets du système.
La partie `Vue` de notre logiciel, représentée par `MainWindow`, hérite de la classe `QMainWindow`, qui comprend un framework pour la création de l'application. `MainWindow` est donc une classe composite de `ClickableLabel`, car la destruction de `MainWindow` entraîne la destruction de toutes les parties du logiciel.
La partie `Contrôleur` correspond à la classe `ClickableLabel`, celle-ci héritant de la classe Qt `QLabel` qui permet de contenir les images. Ses données et fonctionnalités sont détaillées dans la partie 3.3.1. `ClickableLabel` est une classe composite de `Slic`, car la destruction d'une l'image entraîne la destruction de la carte de segmentation associée.
La partie `Modèle` correspond à la classe `Slic` et contient toutes les données et fonctions liées à la segmentation de l'image en superpixels.
Chaque classe de notre logiciel contient des matrices représentants les images et les superpixels, et est donc une classe composite de la bibliothèque `OpenCV`. Il aurait été intéressant de limiter les dépendances dans notre logiciel, notamment en se passant de la bibliothèque `OpenCV` pour le stockage des matrices, mais du fait des dépendances déjà présentes vers cette bibliothèque dans la base de code de `Slic` et de la courte durée du projet, nous n'avons pas accordé une forte priorité à cette tâche.

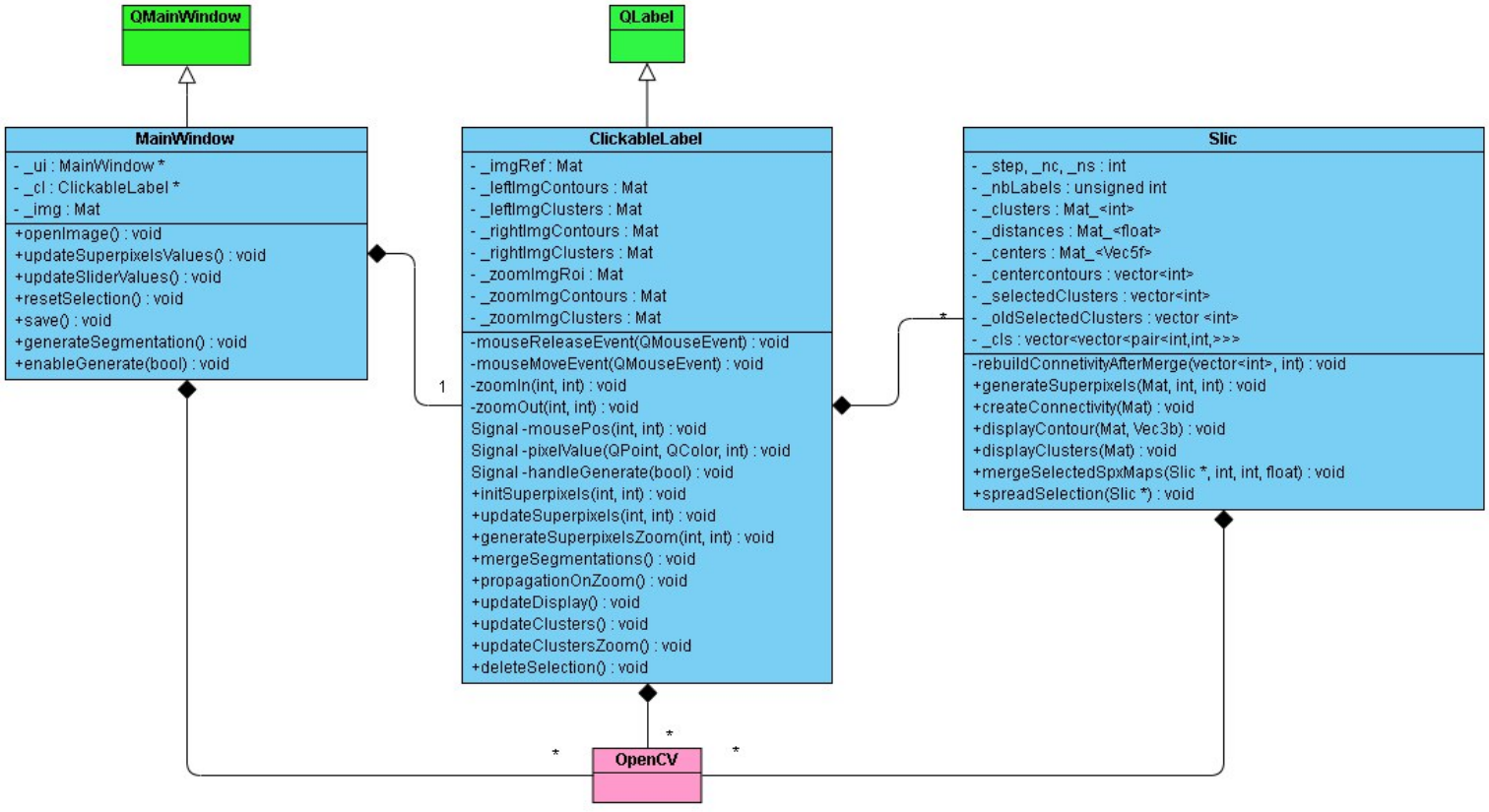

FIGURE 3.2 – Diagramme de classes

3.2.2 Diagramme de séquence

Le diagramme de séquence (Figure 3.3) modélise les interactions possibles avec l'utilisateur. Nous avons choisi de ne montrer que les fonctionnalités principales du logiciel, à savoir la sélection de superpixels, le zoom, le re-segmentation et le changement des paramètres de `Slic` avec les sliders.

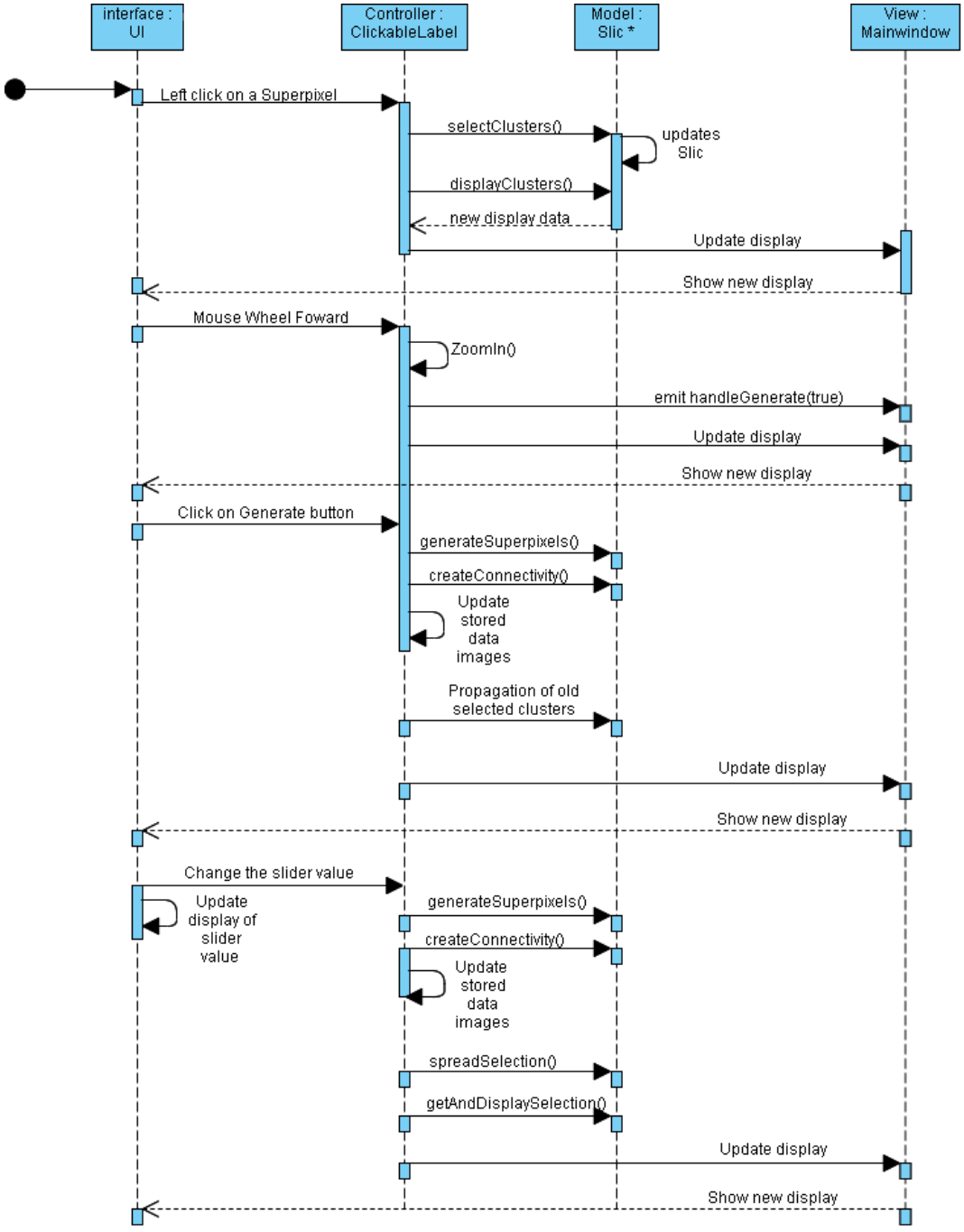

FIGURE 3.3 – Diagramme de séquence

3.3 Implémentation

3.3.1 ClickableLabel

La classe `ClickableLabel` a pour objectif principal de faire le lien entre les interactions utilisateur sur les images et les algorithmes de segmentation en superpixels de la classe `Slic`. De plus, la classe contient les deux cartes de superpixels (à gauche l'image segmentée pouvant être sélectionnée et zoomée et à droite la carte de segmentation avec la sélection actuelle). Il est donc nécessaire de pouvoir communiquer entre ces deux parties pour permettre un seul calcul de superpixels commun et une optimisation du temps de calcul.
Le logiciel contient une seule instance de `ClickableLabel` qui contient plusieurs instances de la classe `Slic()`, pour permettre à la fois la segmentation sur l'image initiale (`_slic`) et la re-segmentation lors d'un zoom (`_slicZoom`). Lors d'un raffinage, la fusion des cartes se fait alors entre la segmentation initiale (`_slic`) et la nouvelle segmentation qui contient les changements de sélection ou dé-sélection (`_slicZoomSelection`).
Pour éviter de devoir recalculer les cartes de segmentation lors d'une sélection/dé-sélection, la classe `ClickableLabel` possède en attribut l'image initiale (qui permet l'initialisation des superpixels au lancement de logiciel ou aux changements des paramètres du `Slic` avec les sliders), les images de gauche et droite avec les contours des superpixels (lors d'une sélection/désélection, il suffit alors de remplir ces images sans passer par un recalcul des superpixels), ainsi que l'image rognée avec et sans les contours superpixels, correspondant au zoom actuel.

3.3.2 Propagation de la sélection

La propagation des superpixels est une fonctionnalité implémentée permettant de garder la sélection précédente après avoir zoomé et re-segmenté une partie de l'image, pour ainsi éviter à l'utilisateur de devoir sélectionner à nouveau les superpixels pendant un raffinage.
Cet algorithme est appelé au moment de la re-segmentation sur l'image zoomée (i.e le clic sur le bouton `Generate`). Il consiste à parcourir les pixels contenus dans les superpixels sélectionnés dans l'espace initial, puis chercher les pixels correspondants dans l'image zoomée et sélectionner les superpixels dans lesquels ils se trouvent.
La correspondance entre pixels de l'image initiale et pixels de l'image zoomée se fait actuellement grâce au niveau de zoom (`_zoom`) et aux coordonnées du pixel situé tout en haut à gauche de l'image zoomée dans l'espace initial (`_xRoi` et `_yRoi`).

3.3.3 Fusion de deux segmentations

De par le fait que notre outil met à disposition un moyen d'agrandir des images ou des zones d'une image pour les segmenter en superpixels à différentes échelles, nous avons dû implémenter un algorithme fusionnant deux segmentations. Prenons comme exemple une image de fleur visible en Figure 3.4. A partir de cette image, notre outil déduit une segmentation en un nombre de superpixels choisi par l'utilisateur. L'utilisateur l'utilise afin de détourer l'objet qui l'intéresse : la fleur. On obtient une première segmentation $\mathscr{S}_1$ de la Figure 3.5. Cette segmentation pouvant manquer de précision, l'utilisateur peut choisir d'agrandir l'image de départ, de générer une nouvelle segmentation $\mathscr{S}_2$ sur celle-ci, Figure 3.6, puis de définir une zone de sélection plus précise, soit, comme c'est le cas dans la Figure 3.7, en sélectionnant de nouveaux superpixels, soit en dé-sélectionnant les zones n'appartenant pas à la fleur.

FIGURE 3.4 – Image d'exemple de départ

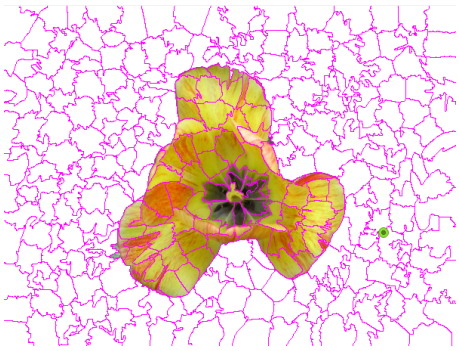

FIGURE 3.5 – Première sélection sur l'image de départ

FIGURE 3.6 – Zoom sur la partie à resegmenter

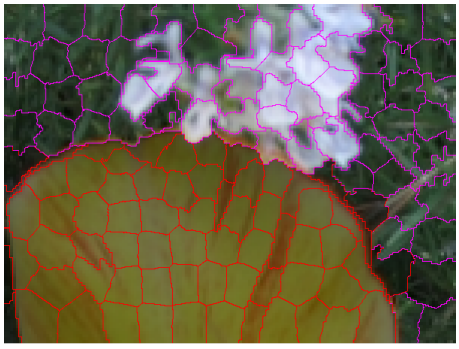

FIGURE 3.7 – Deuxième sélection sur l'image zoomée

On souhaiterait maintenant que la segmentation $\mathscr{S}_2$ restreinte à la sélection de la fleur jaune, visible en Figure 3.7 soit appliquée à la segmentation $\mathscr{S}_1$. On obtiendrait alors un détourage plus précis de la fleur, visible en Figure 3.8 et Figure 3.9.

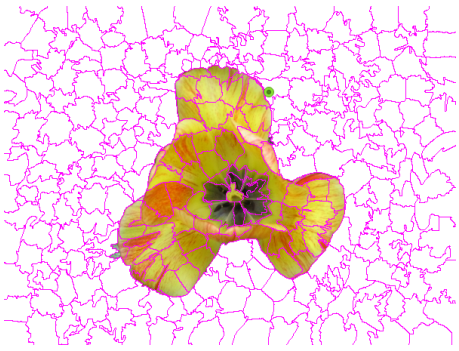

FIGURE 3.8 – Résultat de l'opération de fusion

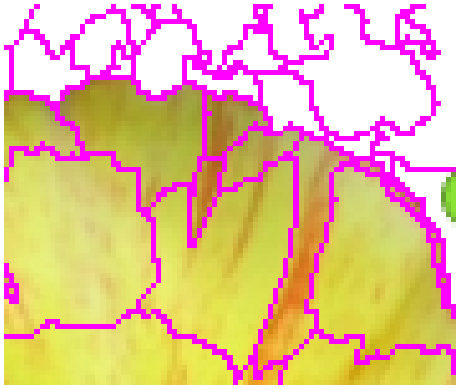

FIGURE 3.9 – Zoom sur la zone fusionnée de la fleur

Nous avons implémenté cette fonctionnalité en une méthode `mergeSelectedSpxMaps()` de la classe SLIC. Cette méthode prend en paramètre une instance de Slic, sous forme d'un pointeur constant. Nous avons choisi de procéder ainsi car la classe SLIC contient plusieurs structures stockant la même information de différentes manières et, selon l'opération à effectuer, utiliser une structure plutôt qu'une autre permet d'accélérer les temps de calculs. De plus, grâce au paramètre SLIC, il est notamment possible de récupérer la liste des superpixels sélectionnés de $\mathscr{S}_2$.

Notre idée est de mettre à jour les contours de la segmentation $\mathscr{S}_1$ en préférant les contours de la nouvelle segmentation à ceux de l'ancienne pour la zone sélectionnée en Figure 3.7. Il est à noter que lorsque nous évoquons la zone sélectionnée en Figure 3.7, nous ne parlons uniquement que des superpixels sélectionnés en mode zoom. Pour cela, nous parcourons la liste de superpixels sélectionnés en Figure 3.7, et, pour chacun de leurs pixels, nous les supprimons de l'ancienne segmentation avant de les ajouter dans un nouveau superpixel. En procédant ainsi, nous évitons des problèmes d'intersection entre les contours des première et seconde sélections si un superpixel de la nouvelle segmentation chevauche deux "anciens" superpixels. L'algorithme est décrit ci-dessous en pseudo-code :

```
foreach(superpixel in selection S2){
    create new superpixel new_spx in S1
    foreach(pixel in superpixel){
        add pixel to new_spx
        find pixel in cls structure and remove it
        update the superpixel label of pixel in the cls structure
    }
}
```

Trois éléments supplémentaires sont à prendre en compte. Premièrement, la sous-image agrandie n'est pas nécessairement la sous-image commençant au pixel (0, 0) de l'image de départ. De ce fait, puisque la seconde sélection à la souris se fait dans le repère de la sous-image, il faut prendre en compte le décalage et le facteur de zoom dans les calculs du pixel dans l'espace de l'image de départ, pour trouver notamment à quel superpixel il appartient dans $\mathscr{S}_1$, et le supprimer.

Deuxièmement, il est possible que l'intégralité des pixels d'un "ancien" superpixel soit sélectionnée et donc qu'ils soient supprimés de ce superpixel, le rendant vide. Nous avons décidé de ne pas supprimer les potentiels "anciens" superpixels vides afin d'éviter de créer des problèmes dans le décalage des labels.

Enfin, notre algorithme de fusion de deux segmentations peut séparer certains superpixels en plusieurs composantes non-connexes, par exemple lorsque les superpixels sélectionnés découpent un "ancien" superpixel en deux. Pour gérer ces cas, nous avons implémenté une fonction afin de reconstruire la connexité des superpixels : `rebuildConnectivityAfterMerge()`, appartenant à la classe SLIC et qui est appelée à la fin de notre algorithme de fusion. L'idée de cette fonction est de déterminer les composantes connexes des anciens superpixels rendus non-connexes par la fusion, pour les transformer chacune en superpixels indépendants. Pour cela, on réalise un parcours du voisinage d'un pixel en largeur (par analogie avec les structures arborescentes). Il est à noter que nous prenons en compte le cas où l'utilisateur s'aventurerait en dehors de la zone de l'ancienne sélection et fusionnerait des pixels n'appartenant pas à l'ancienne sélection. Il serait donc possible qu'un superpixel se trouvant dans le voisinage de la sélection soit également rendu non connexe après la fusion. Aussi, nous appliquons l'algorithme de connexité sur les "anciens" superpixels sélectionnés mais également sur leurs voisins.

L'algorithme est décrit ci-dessous en pseudo-code :

```
foreach(superpixel in selection S1){
    while(all pixels in superpixel have not been visited){
        create new superpixel new_spx in S1
        add first pixel in superpixel to queue q
        while(q is not empty){
            pixel = q.front()
            q.pop()
            foreach(neighbours of pixel){
                if(neighbour not visited yet & belong to the same spx than pixel){
                    add neighbour to q
                    set neighbour state to visited
                }
            }
        }
    }
}
```

4 Tests

4.1 Non-fonctionnements

Notre logiciel a été testé de façon itérative afin d'être en mesure de traiter et corriger les non-fonctionnements de notre projet au fur et à mesure qu'il avançait. Toutefois, quelques non-fonctionnements demeurent dans notre logiciel.

Dé-sélection de superpixels après re-segmentation

Tout d'abord, pour visualiser ce défaut, il faut avoir zoomé au moins une fois dans l'image de gauche. La dé-sélection continue de superpixels fusionnés n'est parfois pas fonctionnelle. Ce défaut se traduit par le fait que certains superpixels dé-sélectionnés dans l'image de gauche restent affichés dans l'image de droite.

Avant d'expliquer d'où semble venir le problème, il faut préciser un point. Sachant que nous utilisons les méthodes de détection de la souris de Qt et notamment `MouseMoveEvent()`, afin d'éviter de vouloir dé-sélectionner l'ensemble des pixels d'un superpixel à répétition dès que le curseur se déplace au sein d'un même superpixel, nous vérifions qu'il ne l'est pas déjà, et ne le dé-sélectionnons que la première fois.

Ainsi, si lors de la détection d'un nouveau superpixel à dé-sélectionner, le curseur de la souris est placé sur une des frontières d'un superpixel, il n'y a aucune certitude que ces frontières soient les mêmes entre avant et après la fusion des segmentations et celles-ci peuvent avoir été décalées de quelques pixels. C'est pourquoi, si un pixel appartenant visuellement à un nouveau superpixel est traversé en continu, il est possible qu'avec l'imprécision du zoom, ce pixel soit considéré comme appartenant au superpixel voisin déjà dé-sélectionné, ce qui entraînera le problème mentionné.

Une solution serait de stocker la correspondance pixel/étiquette dès lors que des superpixels sont modifiés par la fusion. Ainsi, pour un pixel de l'image zoomé, récupérer son étiquette correspondante dans l'image de référence ne nécessiterait plus d'approximation des coordonnées et serait instantané. Nous avions entamé la correction de ce défaut en utilisant une `Map` pour stocker les correspondances mais l'intégration fonctionnelle de cette nouvelle structure à notre code nécessitait un certain temps que nous n'avions malheureusement pas.

Zoom après propagation de la sélection

Le deuxième défaut de notre logiciel apparaît dans un cas précis de la propagation (cf Partie 3.3.2) ; Après avoir sélectionné des superpixels, si l'utilisateur zoome sur une partie de l'image sélectionnée et re-segmente cette partie avec le bouton `Generate`, les superpixels vont être propagés sur la nouvelle segmentation. Mais, si au lieu de sélectionner/dé-sélectionner des superpixels pour permettre le raffinage, l'utilisateur décide de re-zoomer sur cette partie de l'image et de sélectionner des superpixels sans regénérer la segmentation, un conflit va se produire entre les différentes cartes de segmentation (relatives à chaque niveau de zoom), ce qui va avoir pour effet de revenir au niveau de zoom précédent au lieu d'interagir avec le niveau de zoom actuel.

Ce conflit a pour origine notre implémentation du zoom. En effet, celui-ci ne dépend jamais de la vue précédente, mais se réfère à chaque fois à l'image initiale en multipliant par deux le facteur de zoom. Pour corriger ce problème, une solution serait d'implémenter une autre méthode pour le zoom, pour permettre de zoomer directement sur la vue précédente (i.e la nouvelle carte de segmentation), sans passer par l'image initiale (i.e l'ancienne carte de segmentation).

Interface des algorithmes de segmentation

Une des extensions de notre logiciel, souhaitée par les clients, est la capacité à comparer divers algorithmes de segmentation de façon efficace. Cette fonctionnalité peut être considérée comme étant non-fonctionnelle, non pas parce qu'elle induit des non-fonctionnements dans notre logiciel, mais parce qu'elle n'a pas été implémentée.

En effet, nous avons réfléchi aux modifications à apporter dans le code en vue de rendre possible la comparaison d'algorithmes. Selon nous, une solution serait d'interfacer les algorithmes de segmentation dans le but de masquer les appels à `Slic`, ou plutôt d'avoir une classe abstraite dont hériterait chacune des classes correspondantes à un algorithme de segmentation. Toutes les fonctions d'affichage et de traitement des superpixels (sélection, fusion) seraient communes à tous les algorithmes. Une seule méthode serait à redéfinir pour chacun d'eux : `generateSuperpixels()`. De cette façon, la classe `ClickableLabel` n'aurait non pas un attribut `slic` mais un attribut `segAlgo` par exemple, qui serait initialisé à l'algorithme souhaité.

Cependant, jugeant la réalisation et finition des différents tests primordiales, nous avons été contraints de diminuer la priorité de réalisation de cette fonctionnalité et n'avons pas eu le temps de l'implémenter.

4.2 Tests

Nous en venons maintenant à la partie des tests réalisés, que l'on peut découper dans notre cas en trois catégories : les tests unitaires, les tests de performances ainsi que les tests que l'on associera aux tests d'intégration. Les deux premières catégories sont davantage orientées sur l'algorithme qui est au coeur de notre logiciel : l'algorithme SLIC. Ces tests ont été écrits dans un répertoire séparé, nommé "./qt_tests" directement à la racine du projet. Dans ce même répertoire se trouve un fichier README contenant les informations relatives à l'exécution des tests et l'accès à leurs résultats.

4.2.1 Tests unitaires

Notre logiciel étant relativement simple quant à sa structure globale, on peut bien distinguer l'interface du noyau algorithmique qu'est SLIC. La partie des tests unitaires ne s'est intéressée qu'au noyau, l'interface faisant l'objet d'autres types de tests.

Concernant l'algorithme SLIC, deux principaux opérateurs étaient à tester. Premièrement, il s'agit de la segmentation d'une image en superpixels. Deuxièmement, il s'agit de la fusion de deux cartes de superpixels. Des tests supplémentaires ont été écrits afin de s'assurer de la cohérence des structures utilisées pour stocker les relations entre pixels et superpixels d'une image à tout moment où un objet SLIC est dans un état stable, c'est-à-dire avant et après chaque méthode de modification, mais pas pendant.

Afin de ne pas surcharger le rapport, nous ne décrirons que certains des tests unitaires réalisés sous le format suivant : Nom du test / Scénario / Etat / Résultats / Conclusion. Il est à noter que le texte du scénario de chacun des tests est en anglais étant donné qu'ils proviennent des fichiers générés par l'exécution des tests. Les scénarios et états des tests unitaires ne figurant pas dans le rapport sont visibles dans le fichier "slictest.txt" du répertoire "./results".

Génération de superpixels

Par construction, le nombre d'éléments de la structure `_cls` d'un objet SLIC équivaut au nombre de superpixels créés stocké dans la variable `_nbLabels` du même objet SLIC. Ainsi, pour chacun des tests relatifs à la génération de superpixels sur une image, nous ne vérifions l'état que d'un de ces deux éléments et avons choisi de vérifier l'état de la variable `_nbLabels`. De plus, étant donné que dans l'initialisation des tests unitaires (`SlicTest::init()`), la génération de cent superpixels sur une image non vide est réalisé, l'état de la variable `_nbLabels` doit impérativement respecter les deux conditions suivantes : la stricte positivité et la stricte infériorité au nombre de pixels de l'image. De fait, ces deux conditions sont vérifiées dans chacun des tests relatifs à la génération de superpixels.

Nous avons testé l'opérateur de segmentation d'une image en superpixels aux cas limites : génération de zéro superpixel, d'un superpixel puis d'un nombre de superpixels égal au nombre de pixels de l'image et enfin d'un nombre de superpixels supérieur au nombre de pixels de l'image. Le test générant un nombre de superpixels égal au nombre de pixels de l'image est décrit ci-dessous :

```
Nom : SlicTest::generateNbPxSpx()
Scenario : Test of Slic::generateSuperpixels().
    The number of superpixels is nbPixels, the weight is fixed.
    The test is applied on "flower.jpg".
    The number of superpixels (corresponding to "nbLabels") is tested :
      - nbLabels should be between 1 and nbPixels.
Etat : PASS: [SlicTest::generateNbPxSpx()]
```

- Résultats : La génération, sur une image quelconque, d'un nombre de superpixels égal au nombre de pixels de l'image est, selon nous, un cas limite à tester. Cependant, générer autant de superpixels qu'il y a de pixels dans l'image n'amène pas au résultat auquel nous pourrions nous attendre qui est : chaque superpixel ne contient qu'un pixel. Cela ne révèle pas le disfonctionnement d'une méthode mais relève du fait de la façon dont les auteurs de l'algorithme Slic ont choisi d'implémenter leurs fonctions `generateSuperpixels()` et `createConnectivity()`. En effet, étant donné qu'un des avantages des superpixels est de réduire le nombre d'éléments à traiter, la génération de "trop petits" superpixels (nombre de pixels par superpixel inférieur à un nombre limite) perd tout son sens. Ainsi, aucune condition spécifique n'est à vérifier hormis celles communes à tous les tests relatifs à la génération de superpixels.
- Conclusion : Le nombre de superpixels générés est bien strictement positif et strictement inférieur au nombre de pixels de l'image.

Fusion de deux cartes de segmentation

Concernant l'opérateur de fusion de deux cartes de segmentation, nous l'avons testé sur une image vide, sur l'image "flower.jpg" et avons testé le fait de fusionner zéro superpixels. Le test fusionnant deux cartes de superpixels sur l'image "flower.jpg" est décrit ci-dessous :

```
Nom : SlicTest::mergeOnFlowerImage()
Scenario : Test of Slic::mergeSelectedSpxMaps(Slic, int, int, float).
    The test is applied on "flower.jpg".
    The number of superpixels and the weight are fixed for slic and slic_zoom.
    _slicZoom contains twice the number of superpixels of _slic.
    _slic has two selected clusters.
    _slicZoom has four selected clusters.
    The merge of two segmentations (_slic and _slicZoom) should not add more
    ↪  clusters to _slic than the number of selected clusters of _slicZoom.
Etat : PASS: [SlicTest::mergeOnFlowerImage()]
```

- Résultats : Après la fusion des superpixels sélectionnés de `_slicZoom` avec ceux de `_slic`, le nombre de superpixels contenus dans `_slic` doit à minima être augmenté du nombre de superpixels à fusionner (nombre de superpixels sélectionnés de `_slicZoom`). Nous ne testons pas la stricte égalité car, comme expliqué dans la partie 3.3.1, nous traitons également les superpixels voisins à ceux sélectionnés qui sont donc eux aussi potentiellement recréer en superpixels connexes et ajoutés à `_slic`.
- Conclusion : Après fusion des cartes de segmentation de `_slic` et `_slicZoom`, le nombre de superpixels de `_slic` est bien au moins égal au nombre de superpixels de `_slic` avant fusion additionné au nombre de superpixels à fusionner.

Ensuite, toujours concernant l'algorithme de fusion de deux cartes de superpixels, nous avons testé la méthode qui recrée la connexité des superpixels traités lors de la fusion : `Slic::rebuildConnectivity()`. En voici sa description :

```
Nom : SlicTest::rebuildConnectivity()
Scenario : Test of Slic::rebuildConnectivityAfterMerge(vector<int>, int).
    The test is applied on a black 1000x1000 image to have a regular grid.
    The rebuild connectivity operation should create new superpixels for each non
    ↪  connected subset of superpixel.
Etat : PASS: [SlicTest::rebuildConnectivity()]
```

- Résultats : Nous avons sélectionné, à la main, certains superpixels à la fois pour `_slic` et pour `_slicZoom` de sorte que l'opération de fusion, appliquée à ces derniers, sépare le premier superpixel *S0* de `_slic` (`_cls[0]`) en deux superpixels non connexes. L'opération de connexité est donc nécessaire. Deux nouveaux superpixels sont alors créés dans `_slic` et *S0* ne contient désormais aucun pixel.
- Conclusion : Le premier superpixel de `_slic` ne contient bien aucun pixel après l'opération de connexité effectuée et un nombre positif de superpixels a bien été créé et ajouté à `_slic`.

Sélection des superpixels

Enfin, nous avons également testé l'opérateur de sélection des superpixels correspondant à la méthode `Slic::selectCluster(Point2i)`. Tout d'abord, le fonctionnement de ce dernier a été vérifié sur une image sur laquelle aucun superpixel n'a été généré. Puis, nous avons testé la sélection en dehors de l'image à sélectionner. Enfin, nous avons vérifié le comportement de cette méthode dans le cas où le superpixel à sélectionner l'est déjà.

4.2.2 Tests de validation

Au sujet des tests de validation, nous avons également mis en place des tests de performances sur le module SLIC que nous avons complété. Étant donné que nous ne nous sommes pas contenté d'ajouter des éléments dans ce module SLIC, mais que nous avons modifié des méthodes, les rendant plus robustes notamment, nous avons établi, comme pour les tests unitaires, des tests de performances pour les méthodes les plus critiques de cette classe SLIC.

Parmi les algorithmes de cette classe, nous en avons ciblé trois principalement qui peuvent s'avérer les plus coûteux, à savoir ceux implémentés dans les méthodes : `SLIC::generateSuperpixels()`, `SLIC::createConnectivity()` et `SLIC::mergeSelectedSpxMaps()`. Pour chacune de ces méthodes, nous avons testé divers paramètres, comme par exemple différentes tailles d'images, différents nombre de superpixels générés, ou bien le poids de la distance colorimétrique dans le calcul de la carte de superpixels. Les résultats de ces tests sont représentés sur les graphiques ci-dessous. Les durées arrondies au centième de seconde sont lisibles dans les fichiers avec l'extension csv. Ces durées sont mesurées en mode "CPU-time" et non en temps réel. Enfin, sur les différents graphiques illustrés ci-dessous, la courbe de référence à 1 seconde est à prendre davantage comme un point de repère, plutôt que comme un objectif à atteindre.

Génération de superpixels

Dans un premier temps, on cherche à déterminer par l'expérience si la taille des images a bien une influence sur le temps d'exécution de cet algorithme. C'est donc ce paramètre que l'on fait varier, avec des images noires. Notons que cet algorithme ne vient pas de nous, mais qu'il nous revient de savoir s'il est efficace ou non.

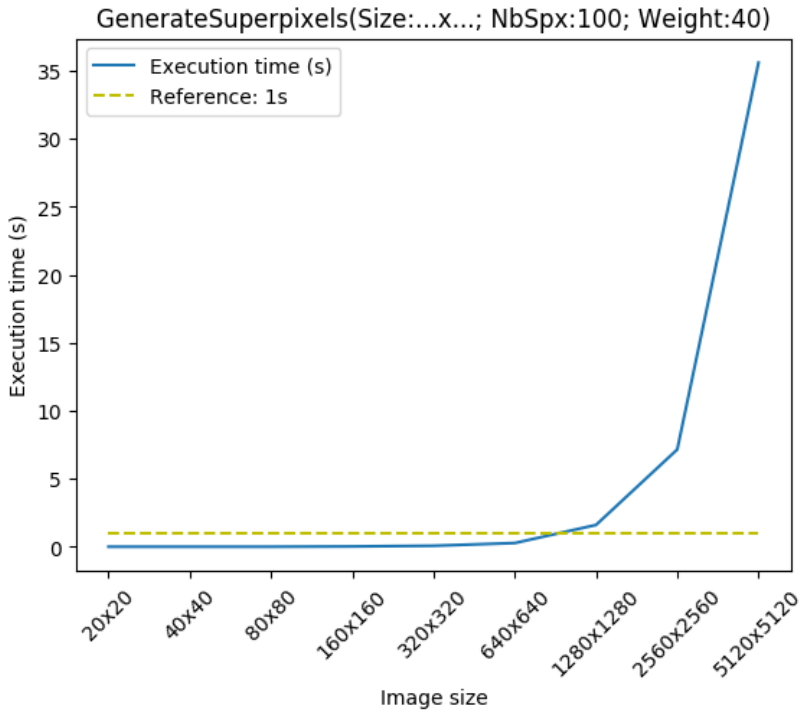

FIGURE 4.1 – Evaluation de la méthode generateSuperpixels de Slic

Ce graphique estime le temps d'exécution de la méthode `generateSuperpixels()` de la classe `Slic` en fonction de la taille d'image. Notons que puisque l'on utilise la classe `Mat` d'OpenCV pour stocker les images, on estime qu'il n'y a pas de variation de performances pour deux images n'ayant pas le même rapport longueur/largeur mais contenant le même nombre de pixels. Sachant que les mesures en abscisses ne sont pas linéaires, on observe qu'expérimentalement parlant, la courbe est quasi linéaire. De plus, on estime raisonnable le temps d'exécution pour des images de taille 1280x1280 au maximum, qui est de 1.6 seconde. On peut tolérer des durées de l'ordre de 5 secondes pour cet algorithme sachant qu'il ne s'exécute qu'une fois par image en moyenne, au début, et par conséquent, tolérer des images de tailles 2500x2500 environ. Les exemples fournis par nos clients sont tous valables en terme de taille.

Il reste à vérifier la durée d'exécution de cette fonction lorsque l'on fait varier le nombre de superpixels ou bien le poids des distances. Les courbes suivantes étant peu lisibles sur un seul graphique, nous les avons séparés en plusieurs graphiques. Chaque graphique représente une taille d'image testée en faisant varier les paramètres sus-mentionnés. Pour chaque figure ci-dessous, le graphique de gauche représente une variation du nombre de superpixels, tandis que le graphique de droite représente une variation du poids des distances.

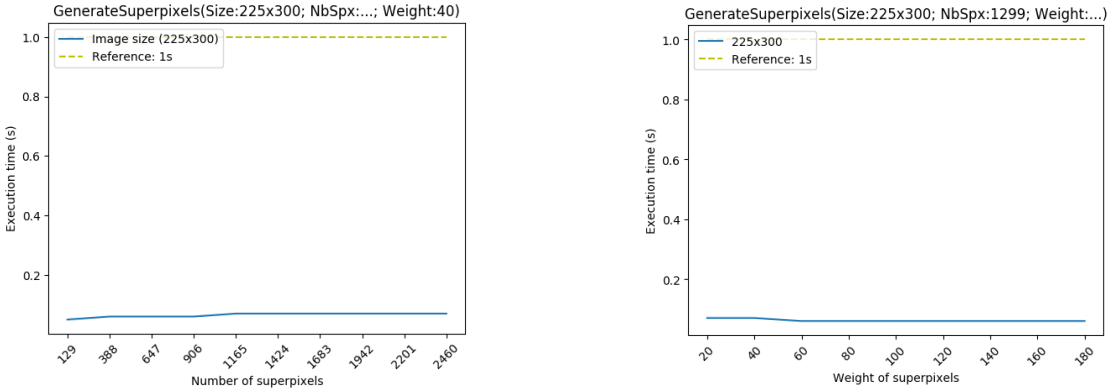

FIGURE 4.2 – Performances de la méthode generateSuperpixels() de Slic en fonction du nombre de superpixels à gauche et du poids des distances à droite pour une image de taille 225x300

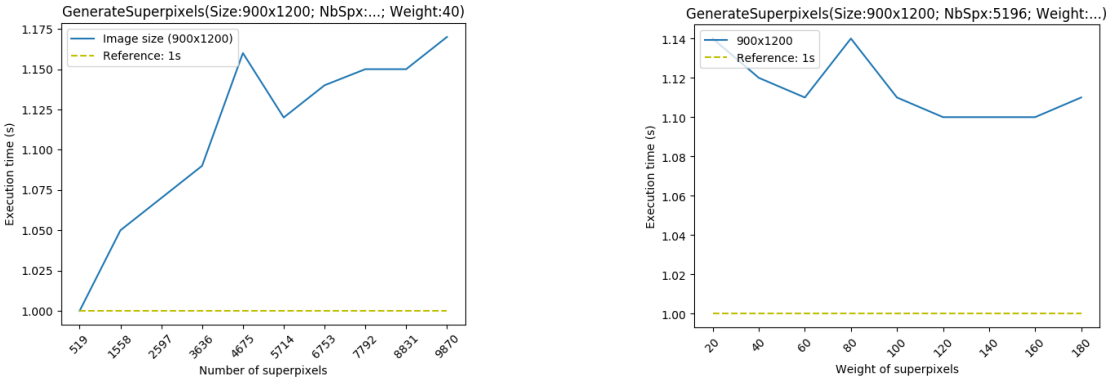

FIGURE 4.3 – Performances de la méthode generateSuperpixels() de Slic en fonction du nombre de superpixels à gauche et du poids des distances à droite pour une image de taille 900x1200

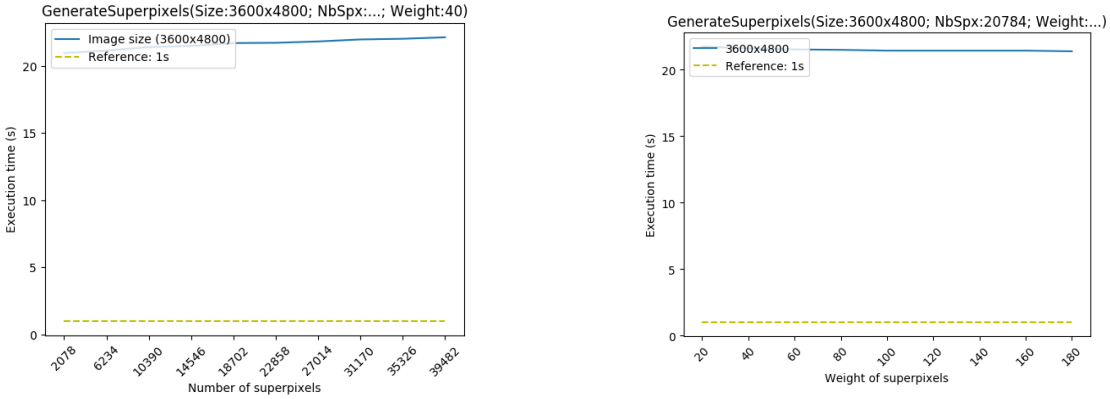

FIGURE 4.4 – Performances de la méthode generateSuperpixels() de Slic en fonction du nombre de superpixels à gauche et du poids des distances à droite pour une image de taille 3600x4800

D'après les graphiques précédents, une tendance ressort à savoir que les courbes sont globalement constantes voire légèrement croissantes. Il est important de noter que la Figure 4.3 n'illustre pas une anomalie, car les échelles sont beaucoup plus précises que sur les autres graphiques. Quant aux bornes des valeurs des superpixels, nous les avons fixées avec nos clients à savoir :

$$[min = \frac{1}{2}\sqrt{longueur * largeur}\ ;\ max = 10\sqrt{longueur * largeur}] \tag{4.1}$$

Comme dans le cas précédent, la limite raisonnable s'arrête aux images de dimensions 1800x2400, pour un traitement d'une durée de 5 secondes. Ce résultat s'applique autant pour le nombre de superpixels que pour les poids. Pour les échantillons d'images proposés par le client, et les paramètres demandés (poids de 20 à 180 environ pour les distances, les valeurs supérieures à 180 n'ayant pas beaucoup d'impact sur la plupart des images), les résultats de cette méthode `generateSuperpixels()` de `Slic` sont tout à fait convenables.

Dans le code, une méthode suit la génération de superpixels sur une image. Il s'agit de la méthode `createConnectivity()`. Celle-ci ajoute aux superpixels une propriété de connexité qui n'est pas garantie à la génération. Ce point justifie d'ailleurs le fait que le nombre de superpixels demandé par l'utilisateur n'est pas nécessairement celui qui apparaît à la fin sur l'image.

Cette méthode n'a pas échappé aux tests de performances. Nous avons mesuré l'impact de deux paramètres différents à savoir la taille des images d'entrée ainsi que le nombre de superpixels souhaité. Afin de ne pas surcharger ce rapport, les courbes ne seront pas illustrées ici mais il reste possible de les générer en exécutant les tests comme décrit en partie 4.2.4. Ceux-ci ont été réalisés sur l'image de la fleur (voir Figure 3.4). Nous avons mesuré 1.36 seconde pour établir la connexité sur cette image redimensionnée à 3600x4800, pour des poids de 40 pour les distances. Ce résultat répond aux exigences : moins d'une seconde et demie pour des images haute définition. De plus, nous avons remarqué que l'impact du nombre de superpixels et des poids est pratiquement inexistant. Pour l'image de la fleur, la variation du nombre de superpixels de 260 à 4930 ne change pas la durée d'exécution de la fonction qui reste à 0.02 seconde.

Enfin, il y a encore un dernier point à discuter quant aux tests de performances. Il s'agit des deux méthodes `mergeSelectedSpxMaps()` et `rebuildConnectivityAfterMerge()`. Afin de limiter au maximum les potentiels effets de bord, nous avons décidé de laisser la méthode `rebuildConnectivityAfterMerge()` avec une visibilité privée, celle-ci n'étant appelée que dans `mergeSelectedSpxMaps()`. Pour

tester cette méthode, il aurait fallu modifier la visibilité de cette méthode en "protected" et créer une classe amie, seulement pour tester cette méthode. De plus, il n'est quoi qu'il en soit plus possible de tester uniquement les performances de la première méthode sans mesurer la seconde en même temps. Au vu du temps qu'il nous restait et des priorités, nous avons décidé de réaliser les mesures sur les deux méthodes en même temps. C'est pourquoi la méthode de test porte le nom : `mergeAndRebuildConnectivity()`.

Pour ce qui concerne cette méthode de test, les résultats semblent très satisfaisants car pour des images de taille 5120x5120, la méthode ne prend que 0.16 seconde pour s'exécuter. Cependant, il reste un point qu'il faudrait approfondir et que nous n'avons pas eu le temps de tester. En manipulant l'interface, nous avons remarqué que cet algorithme est fortement dépendant du nombre de superpixels sélectionnés. Il reste peu probable que 100% des images soit sélectionné en pratique, mais il peut tout de même y avoir beaucoup de superpixels sélectionnés avant la fusion des cartes de segmentation. Toutefois, cet algorithme reste théoriquement coûteux. Nous avons tenté de l'optimiser en réduisant le nombre de superpixels à traiter, mais cela produisait d'autres problèmes de non-connexité des superpixels. En conséquence, nous avons décidé de ne pas accorder de priorité à l'optimisation de cette fonction.

4.2.3 Tests d'intégration/d'interface

Monkey testing

Il ne nous est pas possible de détailler précisément les tests que l'on a réalisé sur l'interface. En revanche, à chaque itération de notre projet, nous avons adopté la technique du "Monkey testing", à savoir de tout tenter pour mettre le logiciel en défaut. Nous répétions un certain nombre de tests, notamment sur l'ensemble des opérations réinitialisant la sélection ou la segmentation d'une image, ce qui nous a permis de relever certains bugs. De plus, ces tests étaient réalisés sur plusieurs images (répertoire `/images/`) provenant de la base de données libre de droit `GrabCut image dataset`, qui constitue une référence pour les tests dans le domaine du traitement d'image.

Durée d'utilisation du logiciel

Nous avons également fait des tests d'utilisation du logiciel, en mesurant la durée pour réaliser un détourage "précis" d'un objet sur une image, en gardant les paramètres de base. Ces résultats dépendent de la rapidité de l'utilisateur et de la précision du résultat attendu, mais permettent de quantifier approximativement le temps nécessaire pour extraire un objet d'une image.
Sur une image "simple" à détourer (ici, `images/flower.jpg`), le détourage se fait en environ 1 minute lorsque l'outil est maîtrisé, avec environ 3-4 raffinages nécessaires (un raffinage correspondant à l'ensemble des étapes `zoom` → `generate` → `selection/deselection` → `merge segmentations`). Sur une image relativement "difficile" à détourer (ici, `images/llama.bmp`), le détourage se fait en environ 3 minutes, avec une dizaine de raffinages nécessaires.

4.2.4 Automatisation

Afin de faciliter l'exécution de ces tests à chacune de nos itérations, nous avons établi un processus d'automatisation prenant en entrée les fichiers de tests et générant en sortie, les résultats dans un sous-répertoire "./results".

Pour les exécuter, il faut lancer le script bash nommé "autotests.sh". C'est dans ce script ainsi que dans le CMakeLists qu'ont été préalablement ajoutés les fichiers de tests à compiler. Les binaires générés sont exécutés et la sortie des tests est redirigée vers des fichiers de texte dans le sous-répertoire "./results", et dont les noms sont associés aux noms des fichiers de tests.

Une fois cette étape terminée, une étape supplémentaire est réalisée, à savoir la génération automatique de graphiques illustrant les performances des zones critiques de l'algorithme SLIC. Cette fois-ci, un script en python (exécutable avec la commande python3) a été utilisé. Il s'agit du script "make-perfplots.py". En effet, il est très simple d'appeler un script en python depuis un script bash, et il est aussi très facile de générer automatiquement des graphiques grâce à la bibliothèque "matplotlib". Il suffit pour cela de fournir en entrée un format de données relativement simple à écrire d'un côté et lire de l'autre. Nous avons choisi ici le format CSV, très simple à générer dans notre cas, et bien adapté au langage python, pour stocker les résultats des tests de performances avant de les afficher sous la forme de graphiques. Ces derniers sont enregistrés dans des images au format PNG, et peuvent être consultés dans le répertoire "./results".

5 Suivi de projet

5.1 Organisation des tâches

Le projet étant de courte durée, nous avons utilisé des outils pour faciliter le suivi du projet, l'organisation des tâches ainsi que l'optimisation du temps de travail, en utilisant des méthodes agiles et en fonctionnant par itérations. Ces outils sont constitués d'un gestionnaire de version `Git`, d'un Kanban (Trello) permettant de classer les tâches par priorité et de les assigner à différents membres, et d'un outil de communication type Messenger (Telegram).

5.2 Adaptation au client

5.2.1 Prototype OpenCV

Pour nous familiariser avec les outils et le code fournis, nous avons commencé par élaborer un prototype se basant sur une interface OpenCV. Celui-ci nous a permis de bien comprendre les demandes du client et de commencer à réfléchir à des fonctionnalités précises (sliders pour les paramètres Slic, sélection des superpixels, zoom et re-segmentation des superpixels), avant de travailler sur une interface Qt plus robuste et exhaustive.
La dernière version du prototype (cf Figure 5.1 ci-dessous) est divisée en 2 fenêtres ; la fenêtre de gauche permet l'affichage de l'image, le choix des paramètres et la sélection des superpixels, et la deuxième fenêtre affiche les superpixels sélectionnés de l'image avec une re-segmentation en fonction du zoom (molette souris).

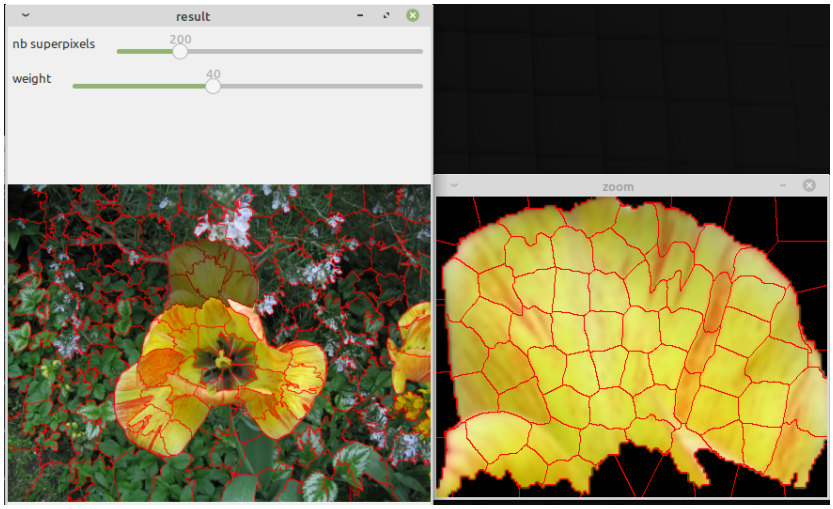

FIGURE 5.1 – Prototype avec interface OpenCV

5.2.2 Première version avec onglets

Dans un premier temps, nous avons fourni une version du projet avec une interface graphique similaire au prototype, en divisant cette fois en 2 onglets et 2 fenêtres pour permettre la séparation entre sélection et raffinage. Le premier onglet permet la sélection grossière des superpixels. Le 2ème onglet permet de sélectionner une partie de l'image qui s'ouvre dans une 2ème fenêtre, où le zoom avec molette est disponible. La carte de superpixels est alors regénérée à chaque zoom, et un bouton `Apply Segmentation` permet de fusionner les nouveaux superpixels sélectionnés dans la fenêtre de zoom avec la carte initiale.
Bien que fonctionnelle, cette version n'était pas assez ergonomique pour le client (nécessité de basculer entre les onglets et les fenêtres en permanence). Nous avons donc repensé notre interface pour permettre un zoom et une segmentation directement dans la fenêtre principale, à la demande du client.

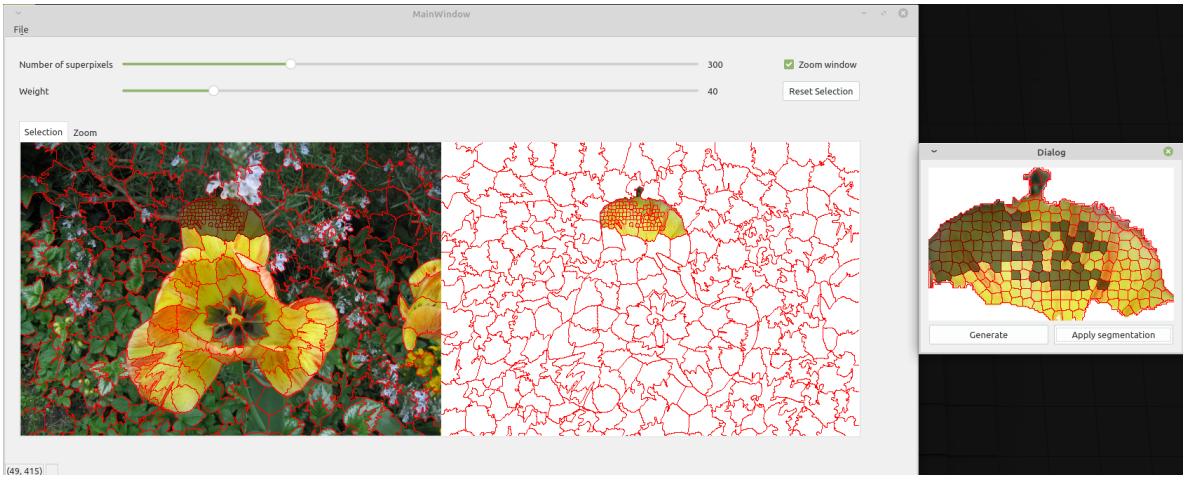

FIGURE 5.2 – Première version fonctionnelle avec onglets

5.2.3 Version avec segmentation hiérarchique

Un des objectifs du projet était de réfléchir à une sélection rapide des superpixels, avec par exemple la pose de label (Scribble). Après discussion avec les clients, il s'est avéré que la pose de label était trop difficile à implémenter avec une simple carte de segmentation basée superpixels, mais qu'une segmentation hiérarchique pourrait être intéressante à exploiter. Celle-ci reposerait sur la génération d'une segmentation très précise avec beaucoup de superpixels ($\sim$ 2000 sp), et un calcul de segmentation hiérarchique par groupement ascendant (chaque superpixel est regroupé progressivement en se basant sur une distance couleur et spatiale).
Pendant la dernière semaine de travail, la moitié du groupe s'est penchée sur cette version hiérarchique pendant que l'autre moitié terminait les tests sur la version existante.
En adaptant le code fourni par le client, nous avons fourni une ébauche d'approche qui, à partir d'une image en entrée, crée une segmentation hiérarchique (sous la forme d'une matrice) en une dizaine de secondes. Chaque ligne de la matrice stocke les labels des superpixels générés. La ligne 0 correspond aux 2000 superpixels initiaux, et chaque ligne suivante correspond à la fusion des 2 superpixels les plus proches. La ligne 1999 correspond donc à 1 superpixel. Cette méthode permet de précalculer les cartes de segmentation et ainsi d'éviter leur recalcul à chaque niveau de zoom.

Cette version (disponible dans le dossier `/pfe_superpixels/qt_hierarchie/`) manque de quelques fonctionnalités (propagation de la sélection, actualisation des superpixels avec le slider) et n'a pas été testée mais permet d'avoir une idée concernant la segmentation hiérarchique et constitue une base pour la pose de label.

Voici l'affichage après ouverture de l'image :

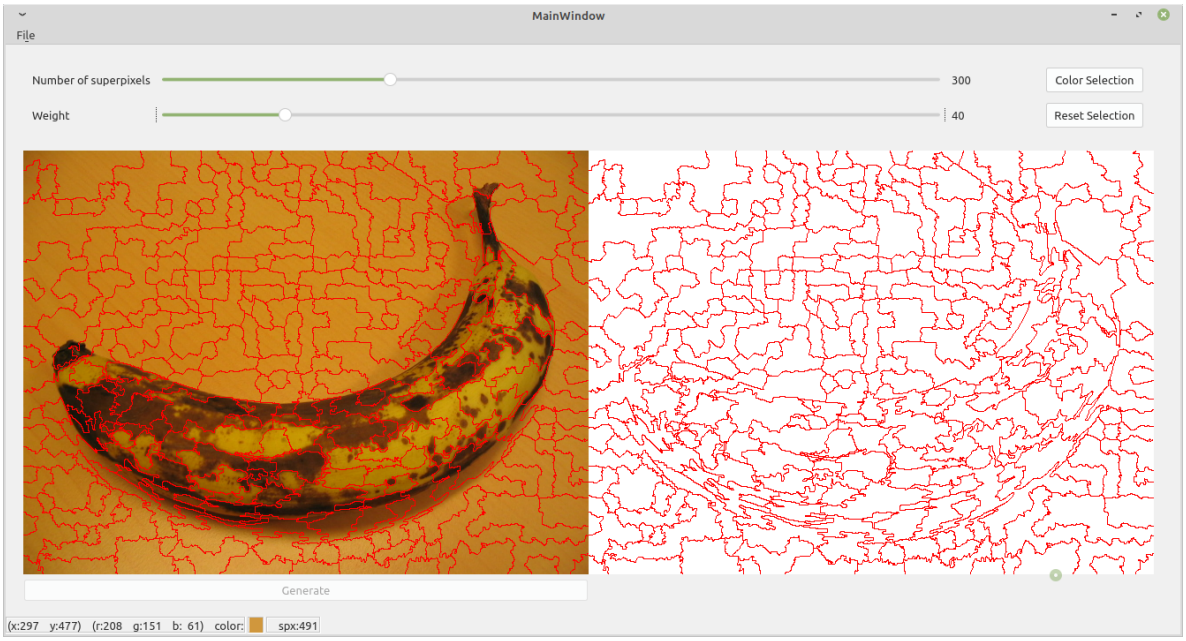

FIGURE 5.3 – Version avec segmentation hiérarchique, vue initiale

Puis en zoomant sur une partie de l'image (re-segmentation instantanée) :

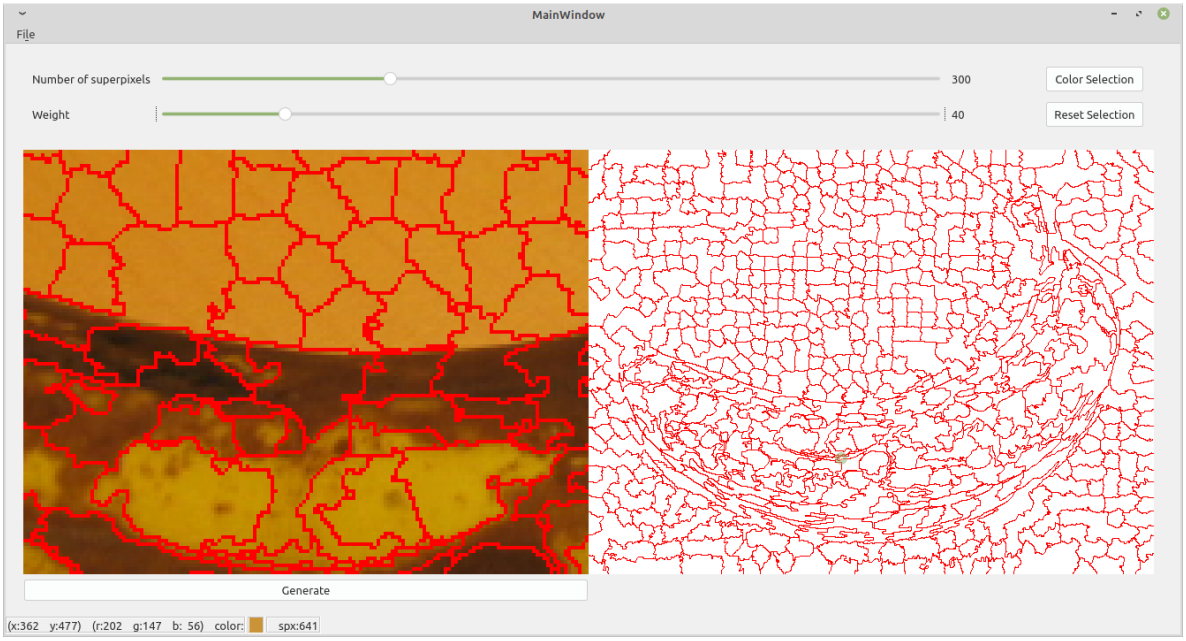

FIGURE 5.4 – Version avec segmentation hiérarchique, vue en zoomant sur le centre de l'image

6 Conclusion

Dans le vaste domaine du traitement d'image, la problématique qui nous a intéressés pour notre projet de fin d'études est la segmentation d'images en utilisant l'approche par superpixels, de manière à détourer des objets représentés.

En lieu et place de l'algorithme de "watersheding" utilisé dans l'outil déjà existant dont se servaient nos clients, il nous a été demandé d'utiliser l'algorithme SLIC (Simple Linear Iterative Clustering), en vue de futures comparaisons d'algorithmes.
Si plusieurs approches différentes étaient envisageables pour le développement de notre solution en terme de choix de langage, d'outil de test, etc, l'option que nous avons choisie est le développement d'une interface Qt en C++.

Bien que notre outil ait quelques défauts dus principalement à un manque de temps, nous avons tout de même répondu aux besoins principaux de nos clients avec un outil simple, capable de segmenter des images en utilisant une version paramétrable de l'algorithme SLIC et qui dispose d'un moyen de raffiner cette segmentation manuellement.
A propos de cette méthode de segmentation locale plus précise, il est à noter que son principal avantage comparé à d'autres méthodes reste un contrôle manuel important de la segmentation locale : il n'est pas nécessaire d'avoir une précision uniforme sur toute l'image, mais seulement dans les régions critiques, ce qui favorise les opérations de détourage dans une image.

Renouvelant l'exécution de l'ensemble des tests à chaque itération de notre projet en commençant par les tests unitaires, nous avons pu mettre en lumière un certain nombre de problèmes dans notre code que nous avons, pour la plupart corrigés.
Autant que possible, nous avons élaboré en parallèle des tests de performances, ce qui nous a permis de modifier les priorités de certaines tâches, d'optimiser davantage certains algorithmes centraux (implémentés dès les premières itérations du projet), ou même encore de se rendre compte rapidement que certaines fonctionnalités n'étaient en fait pas viables.

Toutefois, malgré une prise en main rapide des outils, une répartition efficace du travail, notre projet n'a pas été exempt de problèmes, notamment dans la planification des tâches prioritaires en fonction du temps restant. Nous avons dû renégocier la priorité de certaines tâches pour rendre un produit fini.

Selon nos prévisions, nous n'avons pas pu achever les dernières fonctionnalités attendues par nos clients concernant l'approche hiérarchique de la segmentation pour mettre en place la pose de labels qui favorise la sélection très rapide d'éléments d'une image. Malgré un prototype en partie fonctionnel, nous avons préféré ne pas l'intégrer directement au produit final, étant donné les circonstances exceptionnelles. Cette approche serait cependant intéressante à considérer pour une éventuelle poursuite du projet.

Bibliographie

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. SLIC Superpixels Compared to State-of-the-art Superpixel Methods. *TPAMI, 34(11) :2274–2282*, 2012.

[2] YutoUchimi. Slic-superpixels. `https://github.com/YutoUchimi/SLIC-Superpixels`, 2015.

[3] Jean-François Lalonde. Segmentation d'images en superpixels via slic. `http://vision.gel.ulaval.ca/~jflalonde/cours/4105/h17/tps/results/projet/111063028/index.html`, 2017.

[4] Benjamin Perret. Interactive segmentation with morphological hierarchies. `https://perso.esiee.fr/~perretb/ISeg/`, 2014.