Université de Strasbourg

Master 2 Informatique - Spécialité ISI

Rapport final du Projet 150h

Logiciel d'expérimentation pour les textures multi-échelles

Auteur :
Haehnel Jonathan

Tuteurs :
Kenneth Vanhoey
Basile Sauvage
Jean-Michel Dischler

16 décembre 2013

Table des matières

Chapitre 1

Contexte

1.1 Présentation de l'équipe

J'ai effectué mon travail d'étude et de recherche au laboratoire *ICube*. C'est un groupement de quatre laboratoires de recherche multi-disciplinaire fédéré par l'imagerie. Les grandes disciplines qui y sont représentées sont l'informatique, le traitement du signal, la robotique, la télédétection, la biophysique, la mécanique et l'électronique. Dans tous ces domaines, l'image joue un rôle majeur. En effet, c'est un type de données complexe privilégié dans les travaux sur l'algorithmique, la programmation, la classification, la fouille de données et l'asservissement visuel.

J'ai été affecté dans l'*équipe "Informatique Géométrique et Graphique"* composée de 18 permanents et de deux contractuels. L'équipe IGG concentre ses activités de recherche autour de la modélisation géométrique. L'objectif principal est de définir des modèles géométriques efficaces, prenant en compte la nature variée des données considérées (contraintes, images médicales, numérisation, capture du mouvement), pour concevoir, reproduire la forme, l'apparence et le mouvement des objets 3D afin de les visualiser et d'interagir avec eux de manière précise.

1.2 La méthode : *On-the-Fly Multi-Scale Infinite Texturing*

L'équipe IGG a développé une méthode [2] permettant de texturer des scènes très détaillées en temps réel. En effet, elle permet de répéter une texture d'entrée de manière infinie sur l'ensemble de la scène. Pour cela, un ensemble de motifs présents dans la texture sont pré-calculés et remplacés à la volée durant l'exécution. De ce fait, les effets de bords sont résorbés et la scène devient plus réaliste.

Cette méthode implémente également la notion de texture multi-échelle qui est une texture dont l'aspect est un mélange de deux images de référence, représentant chacune une échelle différente. En fonction de la distance où la scène est visualisée par l'utilisateur, le mélange entre les deux textures d'entrée est modifié. Alors qu'un mélange naïf génère du flou ou un effet de transparence peu naturel, cette méthode établie un mélange astucieux, respectueux des motifs de chacune des images et s'ajustant à leurs contours.

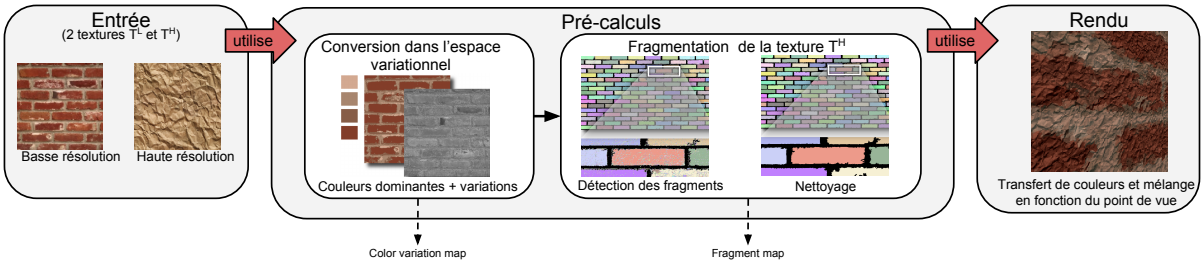

FIGURE 1.1 – Chaîne de traitement permettant de générer une texture multi-échelle comportant trois phases séquentielles : la segmentation des deux textures d'entrée, la fragmentation de la texture haute-résolution et enfin, le rendu de la texture multi-échelle

La figure 1.1 définit la suite de traitement à effectuer pour générer une texture multi-échelle, on peut regrouper ces différents traitements en trois phases. Par ailleurs, il est important de savoir que tous ces traitements s'effectueront de manière séquentielle et que les deux premières phases pourront être directement pré-calculées.

Durant la première phase, l'utilisateur fournit deux images représentant la même texture visualisée à une haute et une basse résolution. Ensuite, on va transformer ces deux textures de l'espace couleur RGB classique à un autre espace de couleur nommé variationnel, où, chaque pixel de l'image est défini par une couleur dominante et par une coordonnée variationnelle qui définit la variation locale de couleur. A terme, cette conversion dans ce nouvel espace de couleur sera utilisée dans la mélange final des deux textures d'entrée, il était donc primordial de l'intégrer initialement dans le logiciel.

Dans un second temps, on va découper la texture haute-résolution en fragments à l'aide d'un algorithme de croissance de région. Ces différents fragments seront utilisés lors de la dernière phase, notamment dans le transfert de couleur.

Enfin, une dernière phase est nécessaire, c'est la génération de la texture finale par un mécanisme de transfert de couleurs entre les différentes textures d'entrée permettant de respecter les fragments trouvés précédemment. La figure 1.2 permet d'illustrer ce transfert de couleur. Enfin, un mélange astucieux entre la couleur de la texture T^H modifiée par le transfert de couleur et la couleur de la texture T^L est effectué en fonction de la distance à laquelle la texture est visualisée. L'image 3.1 permet de se rendre compte des résultats que l'on pourrait obtenir après chaque phase.

1.3 Problématique et objectifs

Actuellement, il existe une application permettant de visualiser le résultat final de notre méthode, néanmoins, toutes les phases de la chaîne de traitement ne sont pas toutes directement intégrées dans ce logiciel. Il faut donc sans cesse faire des allers-retours et des imports-exports dans les différentes applications et les différentes interfaces, ce n'est vraiment pas facile dans ses conditions, de tester notre méthode.

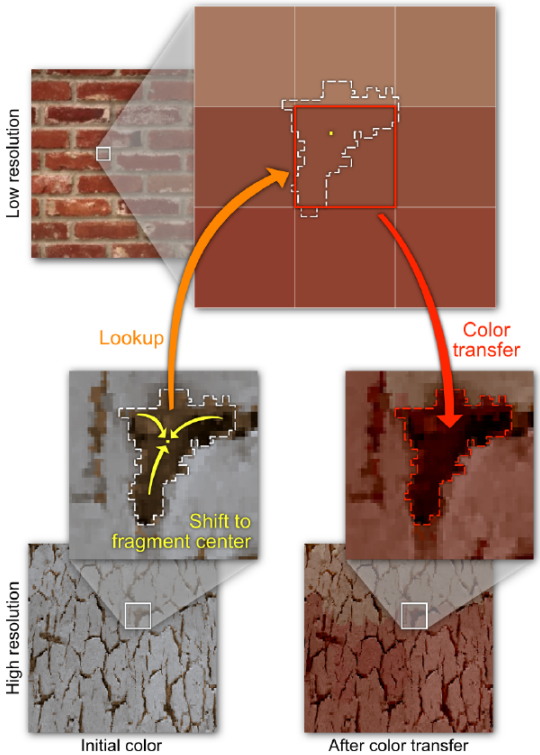

FIGURE 1.2 – Transfert de couleur de la texture T^H selon la texture T^L durant la phase de rendu. La couleur transférée $T^H(p)$ d'un pixel p est en fait, la somme de la couleur $T^H(p)$ originale et de la couleur $T^L(q)$ du pixel q qui est le centre du fragment du pixel p auquel l'on soustrait la couleur moyenne $\overline{T^H}$ de la texture T^H.

Néanmoins, le cadre de mon application est plus restreint, on s'intéressera uniquement à l'aspect multi-échelle dans l'application finale. En effet, il n'est pas demandé d'intégrer l'aspect infinie [1] de la texture présent dans l'article [2].

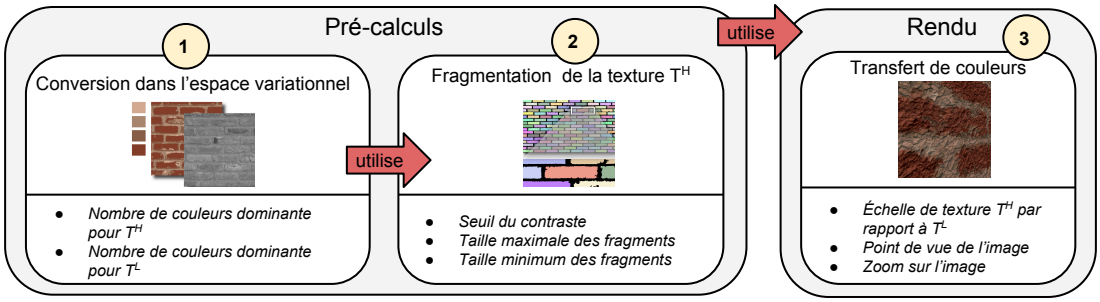

FIGURE 1.3 – Liste des paramètres pour chacune des trois phases du projet.

Le but de l'application est de paramétrer nos deux textures d'entrée et de visualiser la texture générée dans une seule application et interface uniformisée et ergonomique. Sachant que par la suite, l'application sera potentiellement modifiée par d'autres personnes, elle doit être suffisamment modulable, compréhensible et évolutive.

1. Répétition aléatoire de régions dans la texture pour éviter les effets de bords dans la scène

Comme on peut le voir sur la figure 1.3, pour chacune des trois phases, il existe un ensemble important de paramètres à définir. Il faudra pouvoir modifier ses paramètres facilement dans notre application.

1.4 L'existant et organisation

Afin de ne pas perdre trop de temps dans l'implantation de la méthode proprement dite, je n'ai pas eu besoin de re-coder l'ensemble des phases de la chaîne de traitement, en effet, j'ai reçu un ensemble de code à utiliser sous forme de boite noire et de manière transparente. Cela m'a permis de me concentrer plus sur l'aspect d'interopérabilité des différentes phases et sur la mise en place d'une interface graphique ergonomique et unique.

Le code effectuant la quantification initiale [2] de l'image et le code permettant d'effectuer la fragmentation de la texture en régions m'ont été donné. J'ai tout de même effectué l'implantation de la conversion de l'image en espace variationnel (par un algorithme de type K-means, où chacun des clusters est une couleur dominante) et le transfert de couleur final.

Les réunions, presque toutes les semaines, m'ont permis de discuter de l'avancement du projet avec mes tuteurs et d'avoir des retours réguliers et d'avancer efficacement dans les phases critiques du projet. On peut assimiler cette démarche à l'utilisation d'une méthodologie itérative ou agile.

Le tableau 1.1 permet de se rendre compte de l'organisation globale de mon travail tout au long du projet. On constate qu'un grand nombre de tâches ont été mise en oeuvre en parallèle, du fait, d'un lien étroit entre l'interface graphique et le coeur fonctionnel de l'application.

Date	**Description des tâches**
10 octobre 2013	Prise de contact avec les encadrants et analyse du sujet
14 octobre 2013	Début du développement
15 octobre 2013	Mise en place des différentes structures de données (matrice générique, matrice variationnelle, carte des fragments)
20 octobre 2013	Interface graphique : Première page permettant de charger les deux images d'entrée
26 octobre 2013	Mise en place d'un algorithme de type K-means
4 novembre 2013	Interface graphique : Seconde page permettant de configurer la fragmentation de la texture haute-résolution
10 novembre 2013	Seconde phase opérationnelle : Fragmentation de l'image
16 novembre 2013	Interface graphique : Troisième page permettant d'effectuer le rendu final
27 novembre 2013	Troisième phase presque opérationnelle : Rendu et transfert de couleur
4 décembre 2013	Débug important de la page de rendu et génération de la documentation *Doxygen*
8 décembre 2013	Rédaction du rapport
13 décembre 2013	Rédaction de la présentation en vue de la soutenance

TABLE 1.1 – Déroulement du projet [3]

2. Détection de X couleurs dominantes par un algorithme de fast octree quantification
3. Informations récupérées sur le dépôt SVN du projet

Chapitre 2

Mes choix d'implémentations

Comme les boites noires fournies ont été implémenté en $C++$, j'ai donc, logiquement utilisé ce langage dans mon projet. Pour l'interface graphique, j'ai utilisé la librairie Qt étant donné que cette librairie m'était déjà très familière.

2.1 Structures de données

De l'ouverture d'une image d'entrée à la fin de la phase de pré-calcul, un grand flot d'informations est généré par les différents algorithmes [1]. Ces données étant utilisées tout au long de la chaîne de traitements, il était important de trouver des structures de données cohérentes pour les stocker. En effet, une texture est stockée sous forme :

- D'une matrice RGB, où chaque pixel est une triplet $Rouge, Vert, Bleu$. On obtient cette représentation à l'ouverture de l'image.
- D'une matrice variationnelle, où chaque pixel pointe sur une couleur dominante d_i et variation locale v_i de couleur. On peut retrouver la couleur originale en utilisant une table des couleurs dominantes (D_i, V_i). On obtient cette représentation après avoir segmenté la texture, autrement dit, dès qu'on a terminé la première phase.
- D'une carte des fragments, où chaque pixel contient un vecteur à deux dimensions des coordonnées vers le centre du fragment. On obtient cette représentation après avoir fragmenté la texture, autrement dit, dès qu'on a terminé la seconde phase.

Étant donné que le code devait être le plus modulable possible, j'ai mis en place une classe matrice « générique »permettant d'implémenter, soit une matrice de pixels sous forme variationnel (couple di, vi), soit sous forme de fragment (vecteur de décalage au centre du fragment). Les différents types de pixels ont été implémenté par de simples structures ce qui permettent une certaine souplesse et évolutivité dans le temps.

Afin d'accéder à toutes les représentations de la texture, j'ai créé une classe regroupant une matrice RGB, une matrice variationnelle et une carte de fragments. Les instances de cette dernière sont accessibles n'importe où dans le code, à l'aide d'un objet singleton. Pour plus d'informations, consultez le diagramme UML fourni en Annexe A.1.

1. D'une part, l'algorithme de segmentation de l'image en couleurs dominantes, et d'autre part, l'algorithme de fragmentation de l'image.

2.2 Interface graphique

Comme toute l'application fonctionne sous forme d'étapes dépendantes les unes des autres dans la chaîne de traitement, j'ai décidé de dessiner mon interface comme un « installeur de programme ». En effet, on peut passer à étape suivante à l'aide d'un bouton « Suivant [2] »et à l'étape précédente à l'aide d'un bouton « Précédent ».

Dans la pratique, chaque phase est modélisée sous forme d'un onglet qui hérite des fonctionnalités d'un onglet générique. La figure 2.1 illustre ce système d'onglet. Cela permet un ajout très facile de nouveaux modules et une bonne évolutivité dans le temps. Pour plus d'informations, consultez le diagramme UML fourni en Annexe A.2.

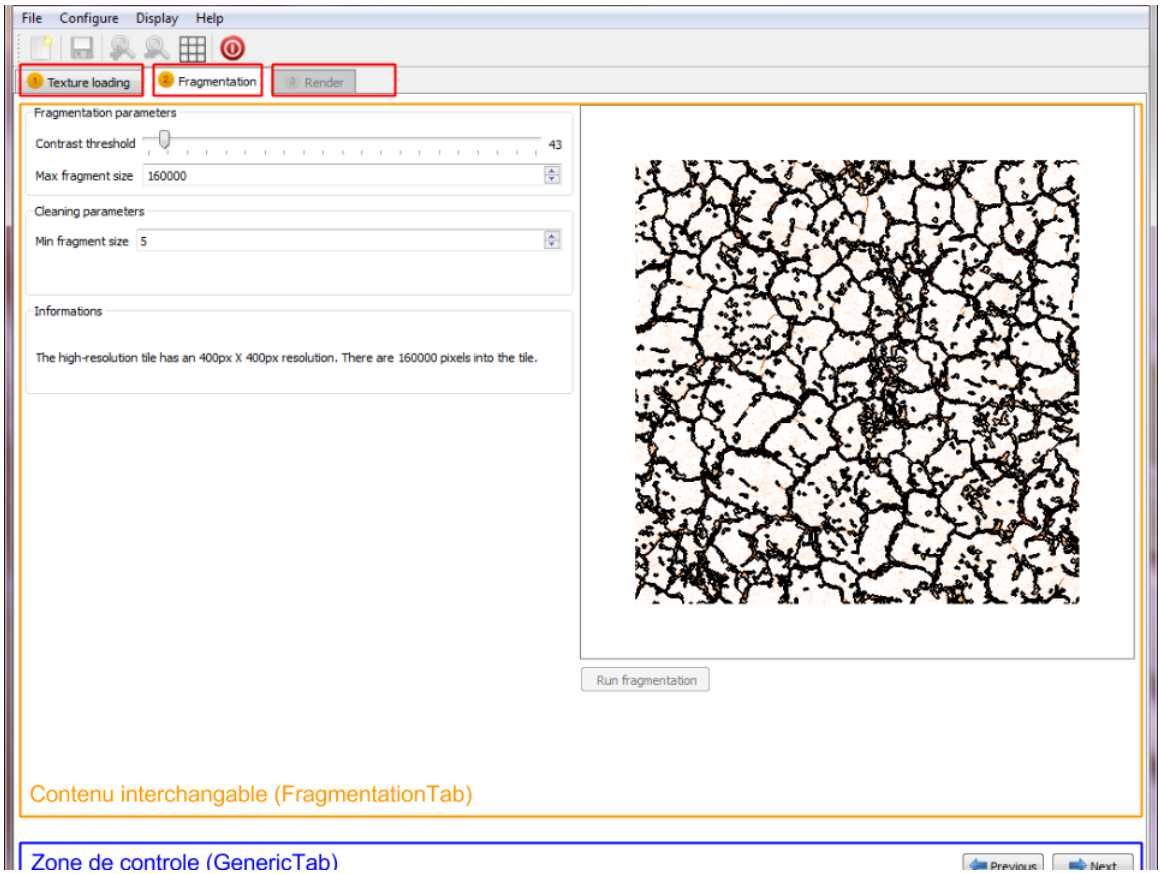

FIGURE 2.1 – Capture d'écran de l'interface permettant d'effectuer la phase de fragmentation de la texture haute-résolution

2. Il faut avoir rempli certaines conditions pré-définies pour que le bouton devienne actif.

Chapitre 3

Mes résultats

3.1 Des résultats encourageants

Globalement, le projet est arrivé à terme. Le produit final respecte les spécifications demandés dans le sujet du projet. Comme illustré dans la figure en annexe B.1, les différentes phases de pré-calculs et de rendu évoluent dans un interfaçage unique et ergonomique, ce qui rend l'utilisation plus facile.

Je pense avoir réussi à mettre en place une application modulable et évolutive dans le temps. Pour rendre plus facile la reprise du code, j'ai également généré une documentation *Doxygen* avec de nombreux diagrammes UML et un petit rapport technique.

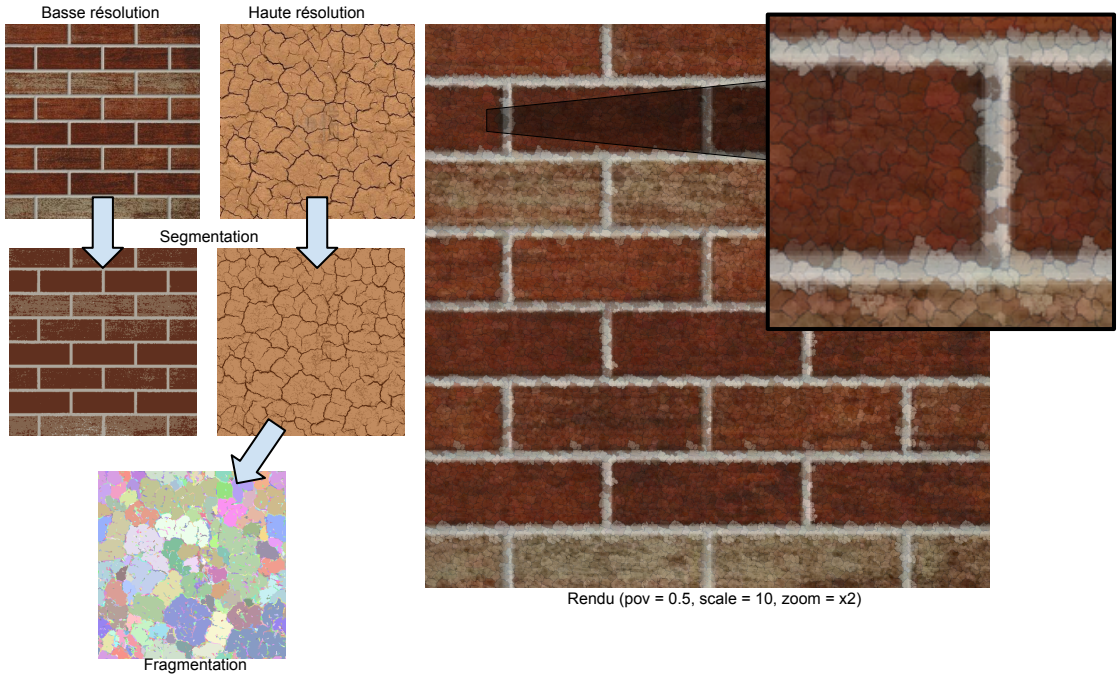

FIGURE 3.1 – Un exemple d'une texture générée par l'application, on peut également découvrir le résultat de la phase de segmentation des deux textures d'entrée et de la phase de fragmentation

3.2 Limitations et perspectives d'évolutions

Les délais du projet étant très serrés, j'ai du, dès le début, définir un ensemble de tâches prioritaires, afin d'obtenir un logiciel fonctionnel en fin de projet. Par exemple, je n'ai pas eu le temps d'implanter certaines fonctionnalités ou optimiser certains calculs notamment dans la fenêtre de rendu [1].

Par ailleurs, la taille des deux images d'entrée doit être identique, mais, on pourrait imaginer d'accepter des tailles différentes sans trop bouleverser le logiciel actuel. Cela permettra donc, d'avoir plus de flexibilité dans nos tests.

Pour finir, le rendu ne s'effectue pas en temps réel à cause du calcul du mélange de couleurs pour chaque pixel de la texture finale ? à chaque changement de point de vue. On pourrait imaginer dans une version future de passer en rendu GPU ce qui permettra de gagner en interactivité.

1. Par exemple, en utilisant la puissance de calcul du GPU via des techniques GPGPU comme *OpenCL*, *DirectX* or encore *GLSL*.

Annexe A

Diagrammes UML des classes importantes de l'application

A.1 Structures de données

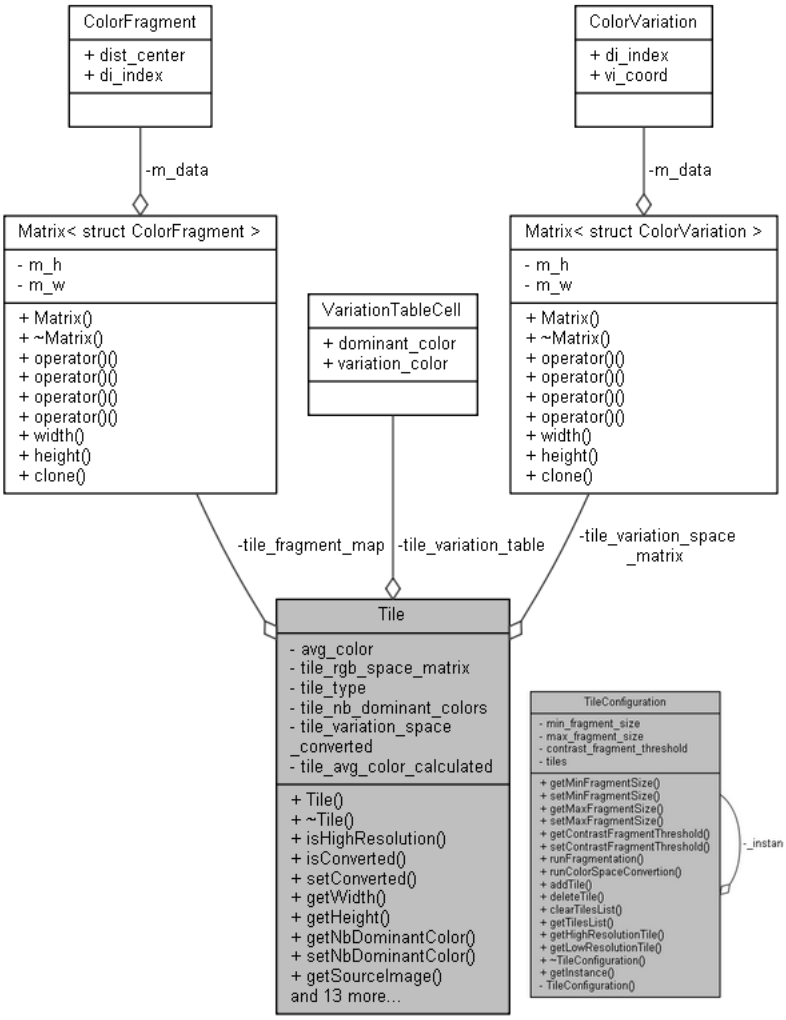

FIGURE A.1 – Diagramme UML des principales classes structurelles

A.2 Interface graphique

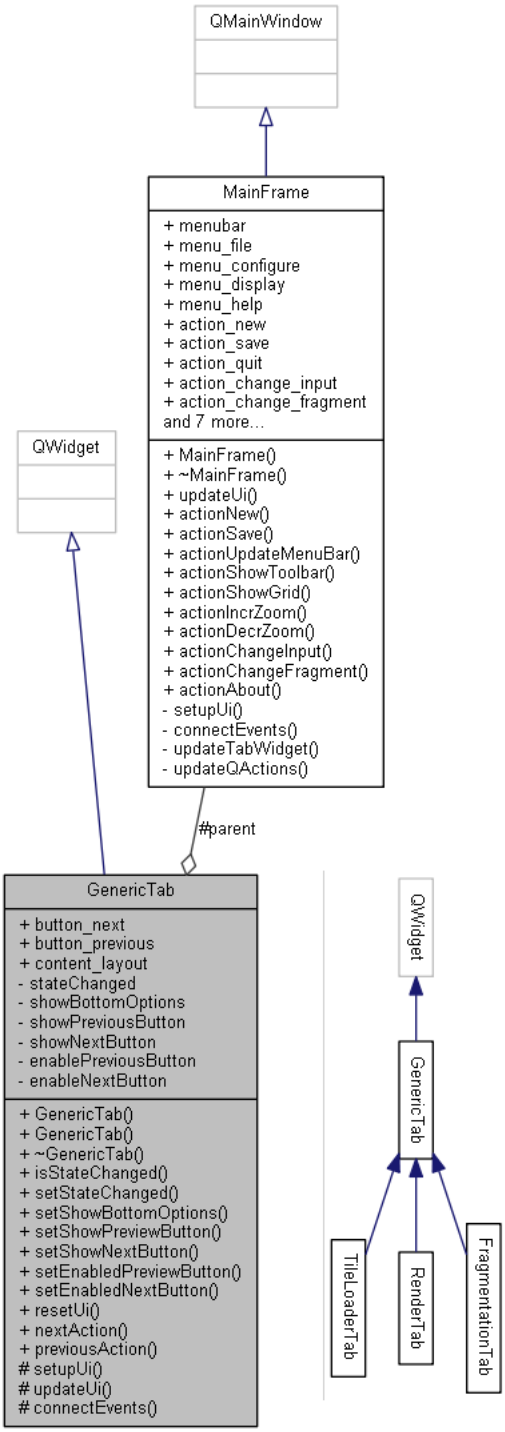

FIGURE A.2 – Diagramme UML des principales classes de l'interface graphique

Annexe B

Captures d'écran de l'application

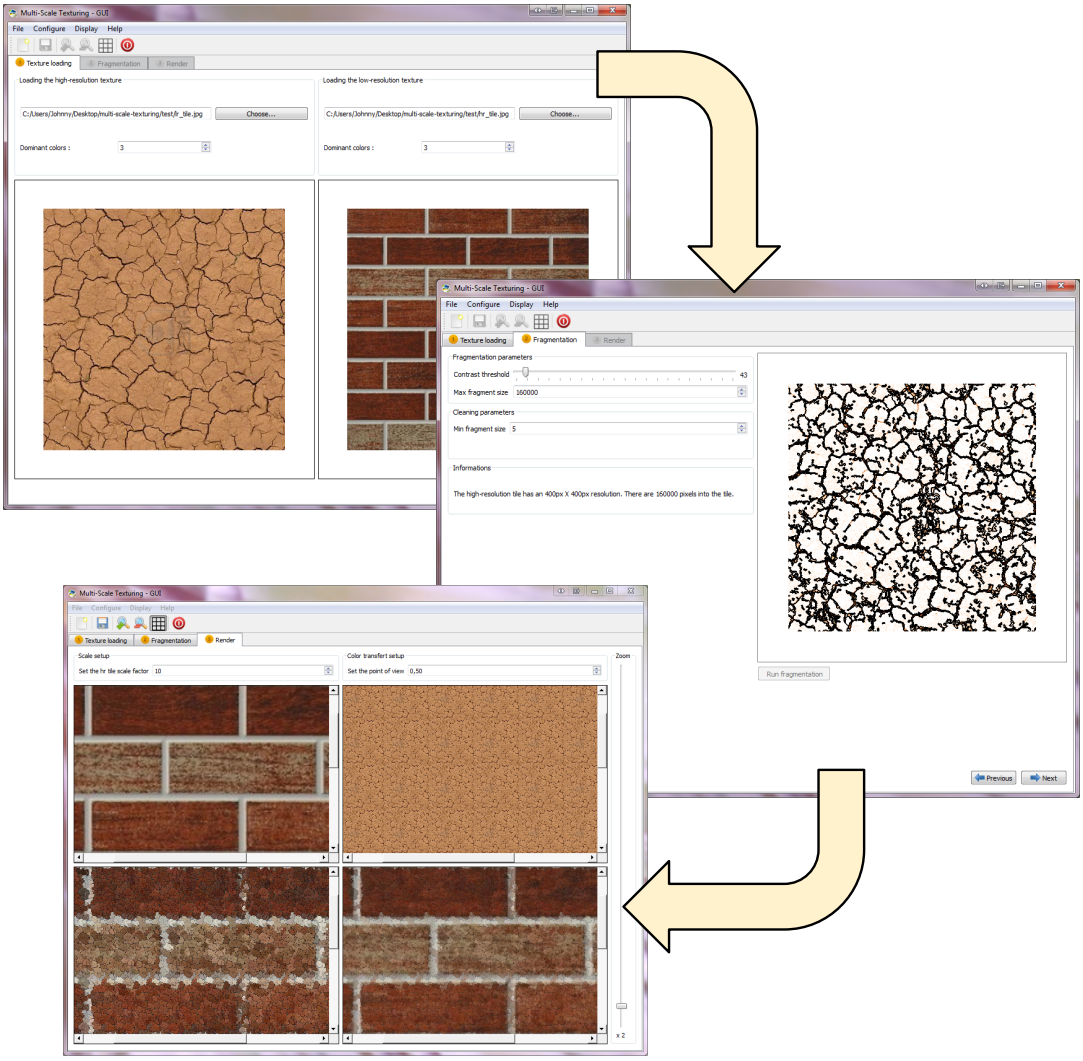

FIGURE B.1 – Captures d'écran de l'interface graphique de l'application. En haut, nous avons l'onglet permettant chargement des deux textures d'entrée et le paramétrage de la segmentation. Au milieu, on peut réaliser la fragmentation de la texture haute-résolution. Enfin, en bas, nous pouvons effectuer le rendu final.

Références

[1] Kenneth VANHOEY et al. *On-the-fly Multi-scale Infinite Texturing - Presentation video*. Nov. 2013. URL : https://www.youtube.com/watch?v=pC-QjjNpL8o.

[2] Kenneth VANHOEY et al. « On-the-fly Multi-scale Infinite Texturing from Example ». Dans : *ACM Trans. Graph.* 32.6 (nov. 2013), 208 :1–208 :10. ISSN : 0730-0301. DOI : 10.1145/2508363.2508383. URL : http://doi.acm.org/10.1145/2508363.2508383.