Rapport de stage de DUT

Développement de MINT-builder

Bastien Oudot

Du 31-03-14 au 06-06-14

REMERCIEMENTS

Avant de commencer le développement du rapport, je souhaitais adresser mes remerciements à plusieurs personnes, sans qui mon stage ne se serait pas aussi bien passé, ou tout simplement n'aurait pas eu lieu.

Au CReSTIC et à M. Laurent Lucas, qui ont accepté de m'accueillir au laboratoire en tant que stagiaire.

À M. Antoine Jonquet, mon tuteur académique.

À M. Nicolas Passat, mon responsable administratif.

Et tout particulièrement à M. Romain Guillemot, mon encadrant, qui a eu la patience de répondre à mes nombreuses questions et de m'apporter les outils nécessaires à la validation de mes objectifs de stage.

Enfin, je tenais également à remercier toute l'équipe du laboratoire qui, à travers sa bonne humeur quotidienne, a permis à mon stage de se dérouler dans les meilleures conditions possibles.

Ce stage a été financé par le CReSTIC, via un Projet Incitatif Amont de l'Université de Reims Champagne-Ardenne (appel 2014)

SOMMAIRE

TABLE DES FIGURES

TABLE DES SIGLES ET DES ABBRÉVIATIONS

CReSTIC – Centre de Recherche en Sciences et Technologies de l'Information et de la Communication.

API – Application Programming Interface ou Interface de programmation en français, ensemble normalisé de classes, de méthodes ou de fonctions par lequel un logiciel offre des services à d'autres logiciels.

OpenGL – Open Graphics Library, interface de programmation qui regroupe des fonctions de calcul utilisées principalement dans le domaine de la géométrie tridimensionnelle.

MINT – Multi View Insight Platform. Il s'agit d'un environnement développé par le CReSTIC orienté sur la visualisation de données (plus particulièrement des données médicales 3D, acquises en scanner X ou IRM) composé d'une API (MINT-toolkit), d'une base de greffons et d'une couche logicielle (MINT-viewer).

URCA – Université de Reims Champagne-Ardenne.

EDI – Environnement de Développement Intégré. Il s'agit d'un logiciel qui regroupe un ensemble de fonctionnalités utiles pour développer dans un langage particulier (éditeur de texte avec coloration syntaxique, compilateur, etc.).

1 Introduction

Afin de valider mon DUT Informatique, je devais, à l'issue d'un an et demi d'études, accomplir un stage de dix semaines en entreprise. J'ai effectué ce stage au CReSTIC, laboratoire de recherche où travaille la plupart des enseignants-chercheurs qui m'ont suivi durant mon cursus. Lors de mon projet informatique du troisième semestre, j'avais eu l'occasion de travailler avec MINT, une plateforme logicielle développée par une équipe de ce laboratoire. Mon projet consistait à développer une série de scripts permettant de générer des modèles 3D de formes géométriques plus ou moins complexes (pyramides, polycylindres, etc.). A l'issu de ce projet, M. Laurent Lucas, responsable de l'un des groupes du laboratoire, et tuteur de mon groupe sur ce projet, m'a proposé ce stage. Très séduit par l'optique de travailler à nouveau dans le domaine de la 3D et plus particulièrement avec MINT, j'ai tout de suite accepté cette proposition.

Ce stage fut pour moi l'occasion de mettre en pratique une grande partie des différentes notions d'analyse et de programmation qui m'avaient été enseignées durant mon cursus, mais aussi d'appliquer des notions plus spécifiques que j'avais acquises au cours de certains de mes modules (réalité virtuelle, traitement d'image, mathématiques appliquées à l'imagerie numérique, etc.) ou lors de mon projet du troisième semestre (principalement la géométrie tridimensionnelle), et également de m'apporter une première expérience professionnelle dans les domaines de l'informatique que je préfère, à savoir le développement logiciel et l'imagerie 3D. Ce stage s'est de fait parfaitement inscrit dans la continuité de mon cursus.

L'objectif principal du stage a été de développer un logiciel, MINT-builder, au sein de la plateforme MINT. Après avoir explicité le contexte du stage en proposant une présentation du CReSTIC, des conditions de travail ainsi que du projet MINT en lui-même, j'expliquerai en détail la mission qui m'a été confiée, puis je décrirai les premiers travaux que j'ai réalisés, avant de développer le déroulement de ma mission principale.

2 Contexte du stage

2.1 Présentation du laboratoire

2.1.1 Historique du CReSTIC

Le CReSTIC est un laboratoire de recherche universitaire dirigé par M. Michaël Krajecki, situé au sein de l'URCA. Il a été créé en 2004 par fusion du LAM (Laboratoire d'Automatique et de Microélectronique) et du LERI (Laboratoire d'Études et de Recherches Informatiques) et est actuellement structuré autour de trois groupes de recherche :

- AUTO (Automatique et Systèmes Hybrides) ;
- SIC (Signal, Image et Connaissance), groupe auquel j'étais rattaché ;
- SYSCOM (Systèmes Communicants).

Il fédère au sein de l'URCA l'essentiel des efforts de recherche dans le domaine des STIC. Il est également le laboratoire d'appui d'une offre de formation cohérente dans le secteur STIC allant du DUT au Master. Le CReSTIC est rattaché à l'École Doctorale « Sciences, Technologies, Santé ».

Le laboratoire est membre de plusieurs GdR (Groupements de Recherche), comme les GdR MACS (Modélisation, Analyse, Conduite, des Systèmes dynamiques), ASR (Architecture, Systèmes, Réseau), IG (Informatique Graphique) et ISIS (Information, Signal, Image, Vision).

La production scientifique du CReSTIC est conséquente. Elle se caractérise sur la dernière période de référence par plusieurs dizaines de thèses ou HDR (Habilitation à Diriger des Recherches) soutenues par plusieurs centaines de publications et/ou communications dans des congrès et revues internationales, avec comités de sélection, et enfin par de nombreuses participations à des projets de recherche ou de transfert technologique sur le plan régional, national et international.

2.1.2 Activité de recherche du groupe SIC

Les activités du groupe SIC recouvrent un panel de compétences larges et cohérentes lui permettant de viser la collaboration étroite de l'instrumentation intelligente, du traitement du signal et des images, de l'intelligence artificielle et de la synthèse d'images. Il est dirigé par M. Laurent Lucas, assisté d'un conseil élu de 6 couples titulaire/suppléant qui représente aussi le groupe au conseil de laboratoire. Par ailleurs, le groupe est pluridisciplinaire, multi-sites, et multi-composantes au sein de l'URCA.

Le groupe SIC, entretient de nombreuses collaborations académiques comme industrielles aux plans local, national et international. Son activité consiste aussi en de nombreuses participations à des comités de programmes et/ou éditoriaux de conférences et revues internationales. Il contribue aux activités des GdR en participants aux évènements organisés par ces derniers. Et enfin il prend part aux Conseils d'Administration de plusieurs associations comme l'AFIG (Association Française d'Informatique Graphique) ou l'EGFR (Eurographics France) et a participé à plusieurs expertises pour l'ANR (Agence Nationale de la Recherche) et l'entreprise OSEO.

Le groupe SIC a mené une quinzaine de projets de recherche depuis 2006, et notamment le projet ICOS (Immersive Computational Surgery). Pendant les opérations, les chirurgiens ont besoin d'accéder aux images préopératoires. Cependant, pour des raisons d'hygiène, ils ne peuvent toucher un clavier ni une souris pour manipuler un logiciel et doivent donc faire appel à une tierce personne, impliquant des problématiques de communication. Le projet ICOS a pour volonté de fournir un accès à l'information direct pour les neurochirurgiens tout en respectant les contraintes d'hygiène, en proposant des solutions informatiques de pilotage d'imagerie médicale sans contact (utilisation de la voix, des gestes pour piloter). De plus, le projet veut offrir un accès à l'information plus immersif par un affichage en relief à ultra haute définition ne nécessitant pas le port de lunettes (écrans auto-stéréoscopiques). Le développement du projet MINT, dans lequel s'inscrit mon stage, constitue le volet logiciel du projet ICOS.

2.2 Conditions de travail

2.2.1 Environnement

J'ai travaillé, durant toute la durée de mon stage, dans un plateau ouvert (aussi appelé *open space*), c'est-à-dire un espace de travail où les bureaux ne sont pas séparés, côtoyant alors une douzaine de personnes. Cette configuration offre beaucoup d'avantages, comme le fait de faciliter grandement la communication, ce qui induit une grande cohésion au sein de l'équipe. De cette façon, j'ai pu poser beaucoup de questions techniques aux gens qui travaillaient avec moi.

J'ai travaillé seul sur les objectifs que l'on m'a confiés. Cependant, j'étais encadré par M. Romain Guillemot, ingénieur de recherche affecté à MINT, qui venait très souvent discuter avec moi de mon travail, répondant à mes questions et me guidant face aux problèmes que je pouvais rencontrer.

M. Laurent Lucas est souvent venu me voir également afin d'observer l'avancement du projet et de me faire part de ses remarques et de ses critiques constructives.

2.2.2 Matériel

A mon arrivée au laboratoire, tout avait été préparé pour que je puisse commencer à travailler dans les meilleures conditions possibles. Un ordinateur m'a été prêté, avec tous les logiciels dont j'avais besoin déjà installés et configurés.

2.2.3 Logiciels et bibliothèques utilisés

La plateforme MINT est développée en C++. Pour les développements de mon stage, j'ai donc naturellement utilisé ce langage.

C++ est un langage de programmation populaire, car il est disponible sur beaucoup de systèmes d'exploitation différents, et de nombreuses bibliothèques (collections de fonctions pouvant être utilisées dans des programmes) ont été conçues pour ce langage. J'avais commencé à étudier le langage C++ quelques années avant d'être arrivé en DUT Informatique, puis en deuxième année, nous avions eu des cours de C++, ce qui m'a permis d'acquérir de bonnes bases et de pouvoir appréhender certains mécanismes du langage assez rapidement.

Ce stage m'a permis de découvrir *Microsoft Visual Studio*, un EDI C++ dont je ne m'étais jamais servi, qui avait été installé et configuré sur ma machine.

J'ai également appris à me servir de *CMake*, logiciel permettant d'automatiser le processus de compilation d'un programme. En effet, afin de compiler un programme, le compilateur a besoin de connaître divers paramètres dépendant de la machine sur laquelle on veut compiler (notamment son architecture, 32 ou 64 bits, son système d'exploitation,

etc.). CMake permet donc de réaliser cette tâche de façon automatique. Je m'en suis servi pour compiler le code de MINT, le premier jour de mon stage, afin de générer la plateforme en adéquation avec l'architecture de ma machine.

D'autre part, afin de faciliter le travail en groupe, le logiciel *Git* a été installé et configuré sur ma machine. *Git* est ce que l'on appelle un logiciel de gestion de versions décentralisé. Il permet à différents programmeurs qui travaillent sur le même projet d'envoyer leurs versions de codes sur un serveur, ce qui permet à tout le monde de pouvoir mettre à jour son code, mais également de pouvoir revenir en arrière en cas de problème.

L'utilisation de *Git* a été particulièrement utile pour permettre aux personnes qui travaillaient sur la plateforme MINT, comme moi, d'en posséder une version la plus à jour possible. En effet, MINT étant encore en développement, il n'était pas rare que quelqu'un ajoutât une nouvelle fonctionnalité sur un des blocs de la plateforme, ou encore détectât et corrigeât un bug. Ainsi, grâce à *Git*, une fois le code modifié, il était possible pour chaque développeur de recharger une version de MINT à jour.

En plus de ces trois logiciels, je me suis servi, au cours de mon développement, de deux bibliothèques informatiques principales :

- Qt (version 5.2), qui est une bibliothèque C++ permettant de développer des interfaces graphiques, et proposant également des classes haut niveau pour simplifier très largement la mise en place de certains mécanismes.
- OpenGL, dont j'ai dû me servir à plusieurs reprises lorsque j'ai développé certaines fonctionnalités propres à l'affichage 3D.

Comme je ne connaissais que moyennement Qt et quasiment pas OpenGL, cela fut pour moi l'occasion de me documenter sur ces bibliothèques et d'apprendre à me servir de ces nouveaux outils par moi-même.

2.3 La plateforme MINT

MINT est une plateforme composée de trois blocs principaux (voir figure 1).

- MINT-toolkit, qui correspond à une API proposant un système de pipeline de traitement et de visualisation commun à tous les éléments de la plateforme (voir figure 2) ;
- Une base d'une cinquantaine de greffons, qui permet d'effectuer différents traitements sur des données (segmentation, édition, fusion, etc.) ;

- MINT-viewer, un logiciel de visualisation (avec gestion de la vision en relief stéréoscopique et multiscopique) et de traitement d'image (principalement issues de l'imagerie médicale), reliant les différents greffons de la plateforme avec le pipeline de traitement et de visualisation du toolkit.

Comme expliqué précédemment, MINT se place dans le contexte du projet ICOS puisqu'il permet la visualisation relief sans lunette de données médicales. De plus, des fonctionnalités de navigation sans contact sont actuellement en développement pour être ajoutées à MINT-viewer, de façon à éviter des problèmes d'asepsie pour les chirurgiens qui utiliseraient le logiciel.

Lors de mon projet informatique du troisième semestre, j'avais eu l'occasion de travailler avec MINT-viewer, et plus particulièrement avec une de ses fonctionnalités qui permet d'interpréter du JavaScript pour modéliser des formes en trois dimensions.

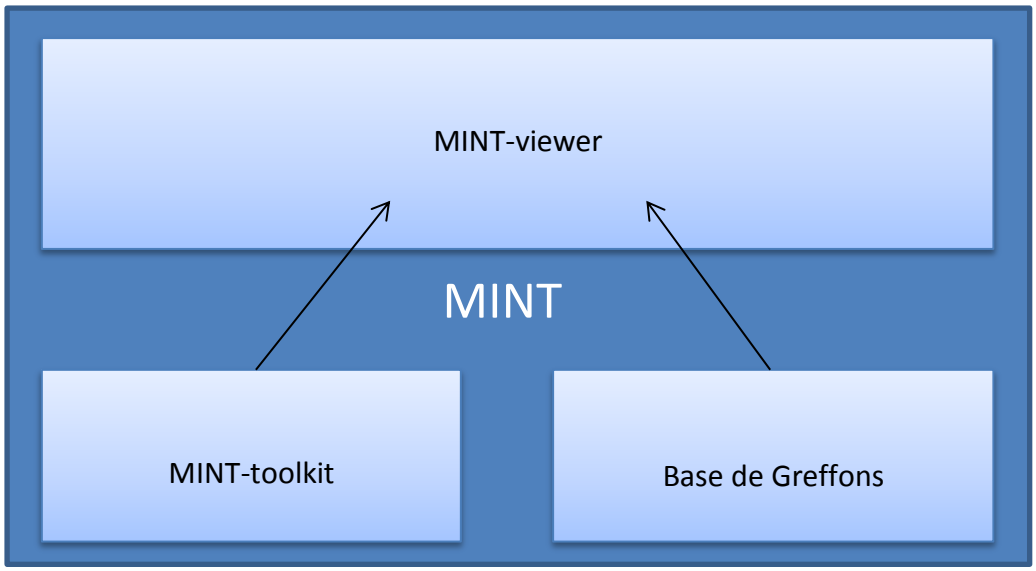

Figure 1 - La plateforme MINT, ses trois composantes et leurs liens.

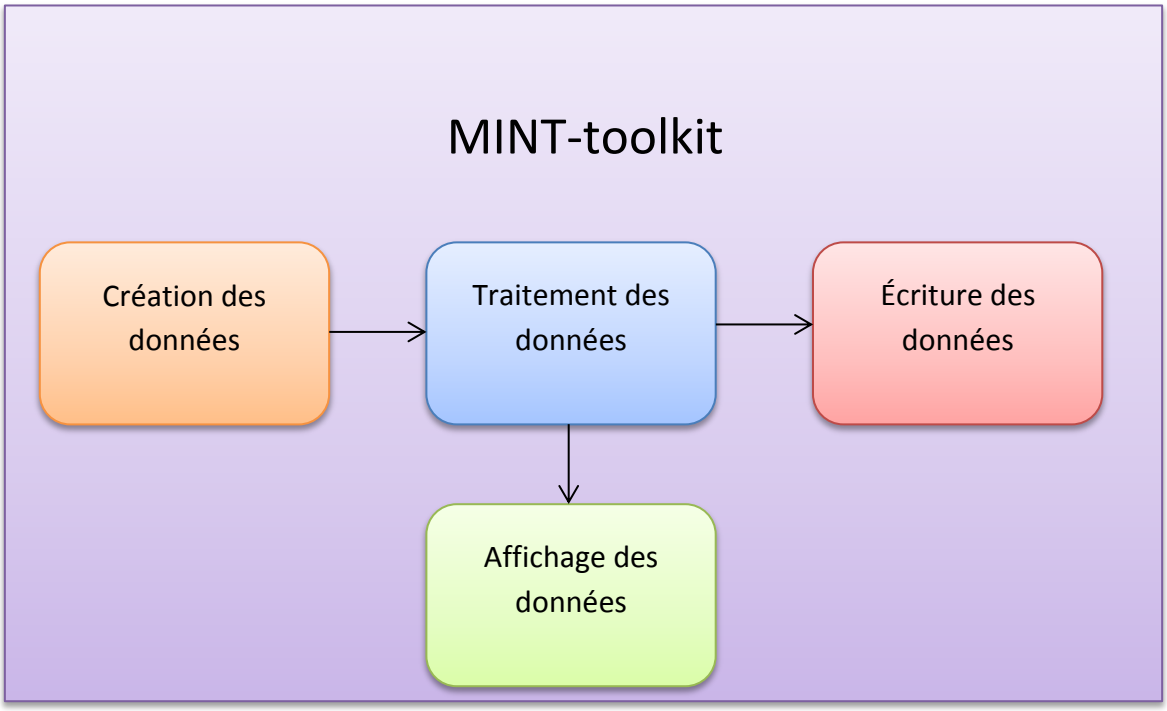

Figure 2 - Le pipeline graphique de MINT-toolkit s'articule autour de quatre briques : la création de données, leur traitement, leur écriture et leur affichage.

2.3.1 Greffons et données

Les données manipulées dans MINT, et en particulier à travers les différents greffons, sont de différentes natures.

- Image, qui sert principalement à représenter des images DICOM (Digital Imaging and Communications in Medicine), à savoir les données issues de l'imagerie médicales (IRM, scanner X, etc.) ;
- Nuage de point, qui permet de figurer un modèle par un nuage de points ;
- Glyphe 2D, dont on se sert pour l'instant pour afficher du texte à l'écran ;
- Glyphe 3D, type de données que j'ai manipulé durant mon projet de semestre 3, permettant de représenter des formes géométriques en trois dimensions.

Cette liste de base constitue les quatre types de données de bases manipulables sur ma version de la plateforme. En plus de cela, il existe deux autres types de données pour MINT que je n'ai pas manipulés, car dépendant de bibliothèques externes non installées sur mon poste de travail, le type maillage (qui représente des modèles tridimensionnels de forme « soupe de polygone ») et protéine (qui permet de représenter des protéines issues d'une grande base de données).

Ajoutons à cela que MINT-toolkit offre la possibilité aux développeurs de créer et d'intégrer à la plateforme leurs propres types de données. C'est une caractéristique que j'ai exploitée lors de mon stage (voir le chapitre 4.2.3.1).

Ces données sont exploitées par les greffons. Ceux-ci sont repartis en différents groupes mais possèdent toutefois des caractéristiques communes.

- Le greffon possède une donnée en entrée. Cette donnée est appelée « input ».
- Le greffon exécute différents traitements qui lui sont propres sur son « input ».
- Une fois ces instructions terminées, le greffon enregistre les nouvelles données obtenues et génère ainsi une nouvelle donnée appelée « output ».

En outre, les greffons de MINT possèdent des caractéristiques propres au groupe auquel ils appartiennent :

- *Type reader* : ce sont les greffons qui permettent d'importer des données. Ils n'ont pas d'« input », mais prennent en paramètre un chemin vers un fichier. Le greffon va ensuite lire le fichier et essayer de construire une donnée avec (par exemple, une image en couleur). Cette donnée construite correspond à son « output » et est ainsi transmise à MINT. Il est à noter que certains readers spécifiques permettent de générer plusieurs données (ils possèdent plusieurs « output »).
- *Type writer* : les greffons writer consistent à exporter une donnée qui leur est transmise en « input ». Ils n'ont pas d'« output » mais prennent en paramètre un

chemin vers un fichier, qui sera le fichier correspondant à l'écriture de la donnée. MINT peut ensuite charger ce fichier, grâce au reader approprié.

- *Type filter* : un greffon filter a pour rôle d'effectuer des traitements divers sur une donnée (changer les couleurs d'une image, agrandir un modèle 3D, isoler une partie de l'image, …). Son « input » correspond à la donnée d'entrée tandis que son « output » correspond tout simplement à cette donnée modifiée par le filtre. Les filtres peuvent présenter plusieurs entrées et/ou plusieurs sorties, en fonction de leurs natures et de leurs mécanismes.
- *Type starter* : ce type de greffon ne possède pas de donnée en « input » puisque son but est de générer, à partir de quelques paramètres spécifiés par l'utilisateur, une donnée (générer une forme de base, une image vide, etc.). Son « output » est alors la donnée générée.
- *Type renderer* : le greffon renderer, aussi appelé greffon de rendu, est chargé d'analyser sa donnée en « input » et d'en proposer une représentation graphique à l'écran, à l'aide de l'API OpenGL (version 3.2 et supérieure, Core Profile). Ce type de greffon étant destiné à l'affichage uniquement, il ne possède pas de donnée en « output ».

2.3.2 Adaptation dans MINT-viewer

MINT-viewer exploite le mécanisme des greffons en les intégrant aux briques constitutives du pipeline de traitement et de visualisation de MINT-toolkit (voir figure 3). Les greffons de type reader et starter ont pour rôle de créer une donnée, puis les greffons filter traitent cette donnée, avant de l'envoyer en écriture dans un greffon writer ou de l'afficher à travers un greffon renderer. Il est à noter que, en plus de cela, MINT-viewer implémente[1] la notion de chaîne de traitement en série, ce qui signifie que la donnée d'entrée d'un greffon (son « input ») peut être la donnée de sortie (l' « output ») du greffon précédent ; à son tour, la donnée de sortie de ce greffon pourra être la donnée d'entrée du greffon suivant, et ainsi de suite.

[1] Implémente : le mot tout à fait correct serait « implante » car « implémenter » est considéré comme un anglicisme. Cependant, il est très utilisé dans le jargon informatique et a été officiellement adopté en France en 2007, avec une définition qui couvre la majorité des opérations réalisées par rapport à un système ou un projet.

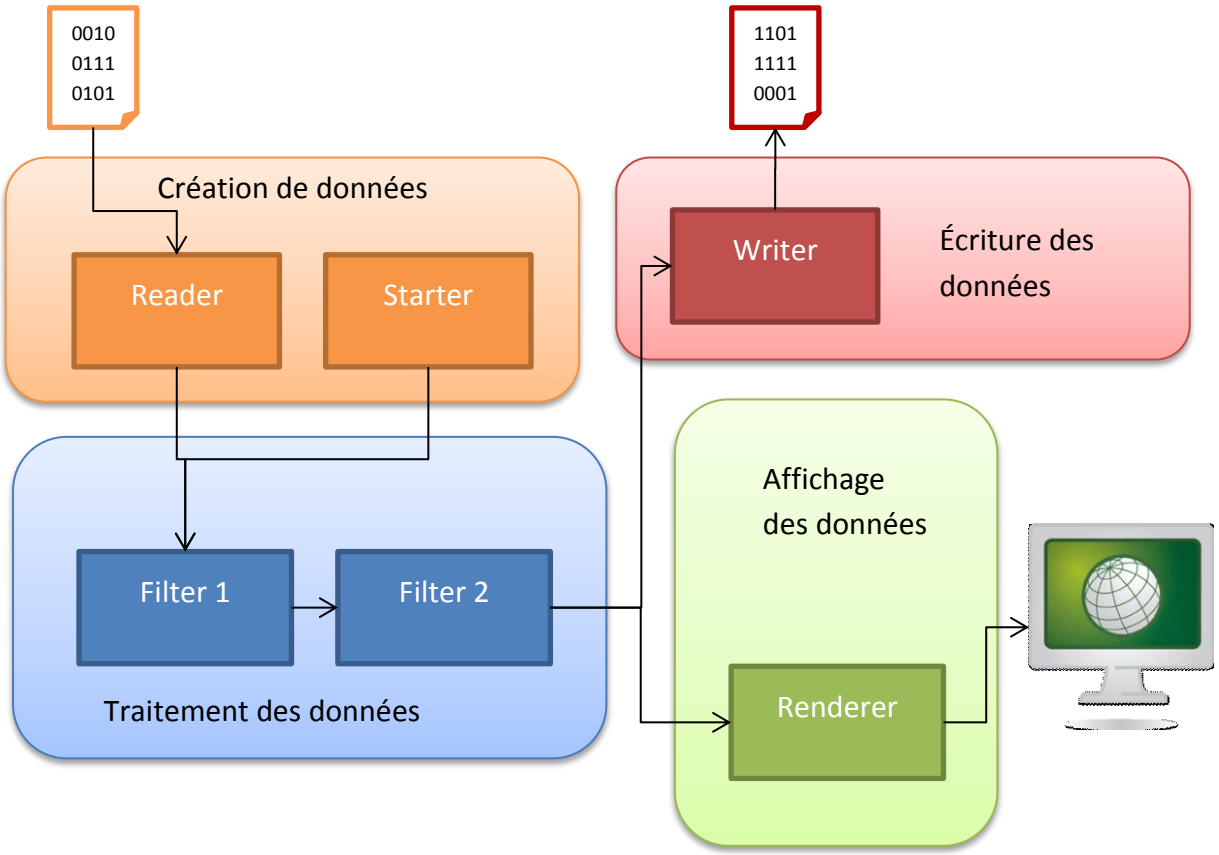

Figure 3 - MINT-viewer adapte le système de greffon au pipeline de traitement et de visualisation de MINT-toolkit. Ainsi, les différents types de greffons sont répartis dans les quatre briques du pipeline, en fonction de leurs natures et de leurs fonctions.

Le logiciel propose une interface graphique, présentée sur la figure 4, composée de différents blocs pour représenter ce pipeline. Un bloc contient la liste des données importées ou crées ; face à lui se trouve une zone où la donnée sélectionnée est affichée. Enfin, il est possible d'appliquer un traitement (ou une chaîne de traitements) à une donnée, à travers une boîte de dialogue. Le fonctionnement de cette chaîne est simple : on ajoute les greffons de traitement les uns à la suite des autres, puis on configure leurs paramètres à travers une interface graphique présente dans le quart en bas à droite de la boîte de dialogue. Enfin, lorsque les greffons sont configurés et que l'utilisateur clique sur le bouton « Ok », le pipeline de traitement s'exécute : la donnée qui a été sélectionnée est envoyée en « input » du premier greffon. Puis la première sortie de ce greffon sert d'entrée au greffon suivant, et ainsi de suite. La première sortie du dernier greffon de la chaîne de traitement est ensuite envoyée au greffon de rendu pour permettre son affichage à l'écran.

Le choix d'implémentation de la partie de traitement du pipeline graphique est efficace mais présente toutefois des limitations, la rendant incomplète aux yeux des utilisateurs :

- Lors de l'exécution des greffons de traitement, seule la première sortie d'un greffon est envoyée en entrée au suivant. Cela signifie qu'on ne pourrait exploiter que la première sortie d'un greffon qui en présenterait plusieurs ;
- La chaîne de traitement ne permet pas de réaliser des traitements parallèles (voir figure 5), qui pourraient servir à exécuter deux traitements différents à partir d'une donnée prétraitée ;

Ce sont ces principales raisons qui ont participé au besoin de posséder un nouvel outil, au sein de MINT, permettant de gérer des chaînes de traitement plus flexibles.

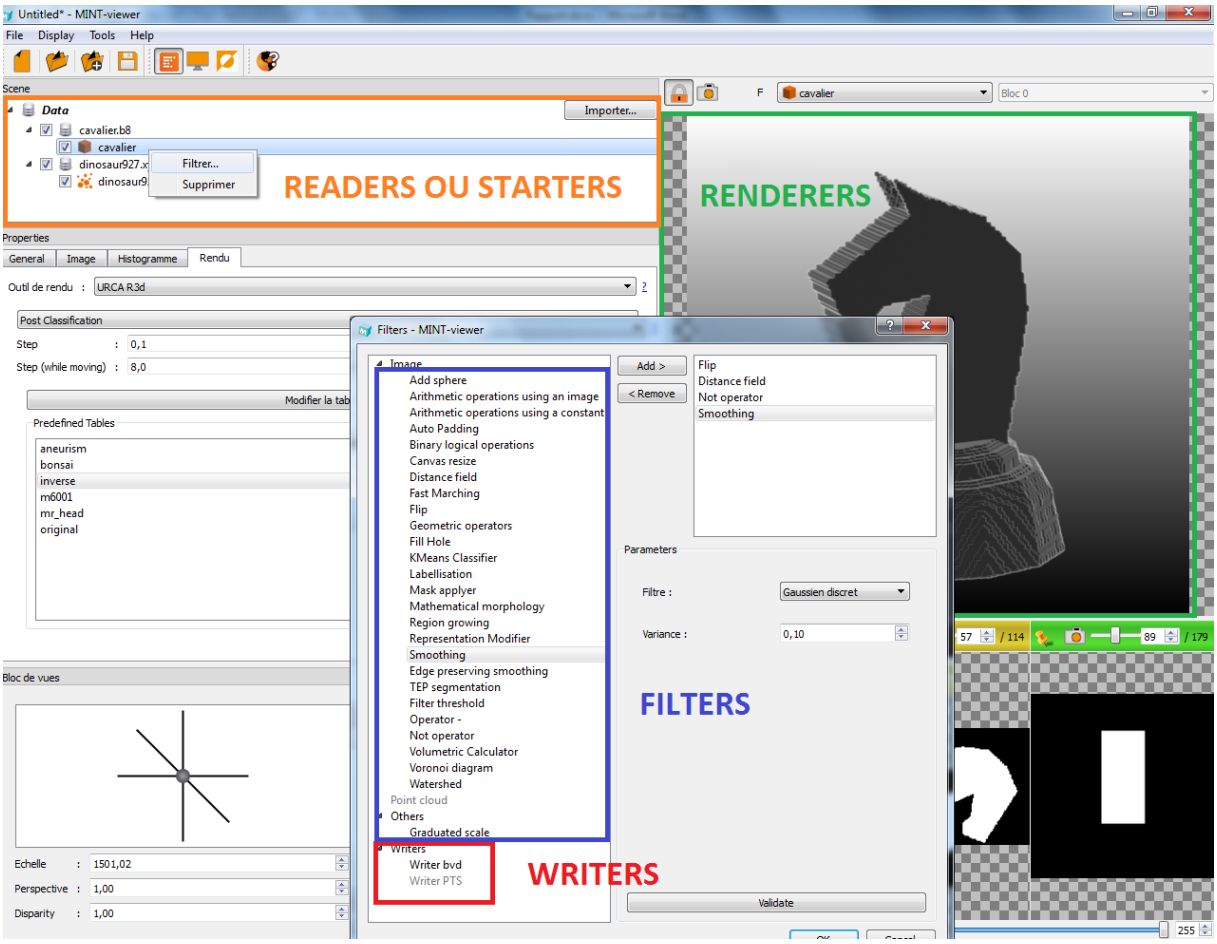

Figure 4 - Interface graphique de MINT-viewer avec mise en évidence des différents types de greffons.

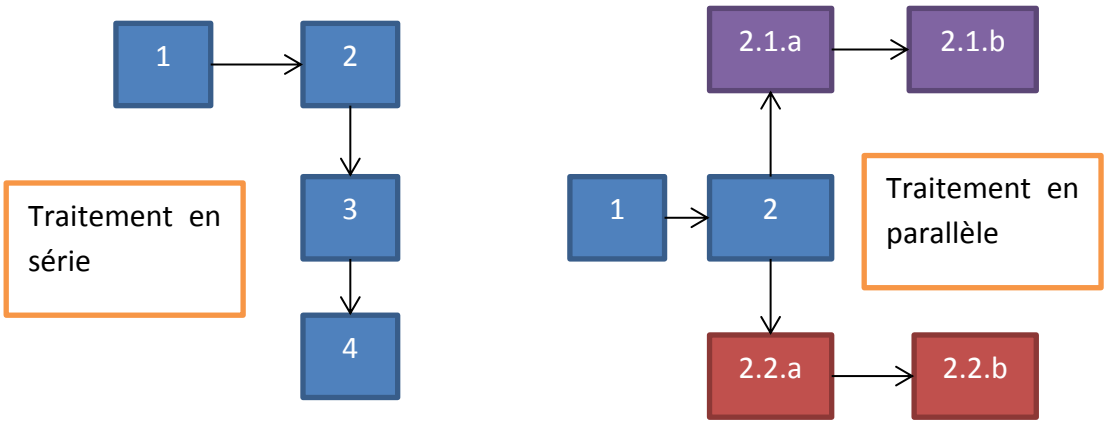

Figure 5 - Traitement en série versus traitement en parallèle

3 Description de la mission confiée

3.1 Objectif principal

L'objectif principal du stage était d'amorcer le développement d'un nouveau logiciel qui s'intégrerait à la plateforme MINT. Ce logiciel, de la même façon que MINT-viewer, devait proposer une utilisation des greffons au sein du pipeline de traitement et de visualisation décrit par MINT-toolkit. Cependant, son but était différent : alors que MINT-viewer était surtout axé sur la visualisation des données et peu sur leur traitement (notamment à cause des limitations de son système, mentionnées plus haut), ce nouveau logiciel, appelé MINT-builder, devait principalement proposer un système de traitement de données complet et flexible, accompagné d'un système de visualisation simple (permettant toutefois une vision en relief, en fonction du type de rendu choisi).

3.2 Cahier des charges

J'avais pour me guider dans mon développement un document, faisant office de cahier des charges, me présentant les différentes caractéristiques, fonctionnalités et contraintes que devait prendre en compte le logiciel :

- Le logiciel devait proposer une interface permettant de concevoir des schémas graphiques afin de construire des traitements complexes à partir des greffons de MINT ;
- Les greffons, listés dans le logiciel, devaient pouvoir être ajoutés dans un schéma par un système de glisser-déposer ;
- Les greffons ainsi déposés pouvaient posséder zéro ou plusieurs entrées, zéro ou plusieurs sorties, zéro ou un bouton de configuration de paramètre, un bouton de parcours de fichier (pour les readers et les writers), un bouton de visualisation (pour les renderers) ;
- Une sortie pouvait être connectée à plusieurs entrées si les types étaient « compatibles » ;
- Une entrée ne pouvait être reliée qu'à une seule sortie ;
- Un schéma faisant intervenir une boucle infinie ne pouvait être accepté/exécuté ;
- Un schéma devait pouvoir être sauvegardé (dans un format XML ou fondé sur XML) dans un fichier pour pouvoir être rechargé plus tard ;
- Un schéma enregistré devait pouvoir être inséré dans un autre schéma ;
- L'interface devait proposer un environnement multi-fenêtré ;
- L'interface devait proposer des fonctions de copier/coller d'une fenêtre vers une autre.

Cependant, avant de me lancer dans l'analyse de ce document et dans le développement de MINT-builder, j'ai été amené à développer une série de greffons durant les trois premières semaines de mon stage, d'une part afin de bien comprendre la structure d'un greffon et pouvoir ainsi produire une meilleure analyse de MINT-builder, et d'autre part pour acquérir de bonnes habitudes de développement, apprendre et connaître les conventions de nommage, bien indenter le code et le commenter de façon précise et pertinente, dans un souci de lisibilité et de maintenance.

4 Premiers travaux

4.1 Compilation de MINT-viewer

Les greffons que j'allais écrire seraient testés avec le logiciel MINT-viewer. De fait, la première tâche que j'ai accomplie fut de compiler le logiciel MINT-viewer afin d'en posséder une version compatible avec l'architecture de l'ordinateur sur lequel je travaillais. Pour cela, j'ai dû utiliser CMake pour la première fois (voir le chapitre 2.2.3 : Logiciels et bibliothèques utilisés), et ainsi en apprendre un peu plus sur ce à quoi correspondait vraiment l'étape de compilation d'un programme. L'installation terminée, je possédais tous les fichiers nécessaires aux tâches de développement qui m'avaient été assignées.

4.2 Écriture des greffons

L'étape préalable d'analyse et de compréhension de l'architecture du système de greffons ainsi que les nombreuses réponses que m'a apportées mon encadrant m'ont permis de commencer à écrire mes propres greffons. J'ai au total écrit cinq greffons (un de chaque type).

4.2.1 Lecture et écriture du format BVD

Le format BVD (Binary Volume Data) est un format de fichier conçu par l'équipe de développement de MINT permettant de stocker, comme son nom le suggère, des données binaires correspondant à un volume, une image en trois dimensions.

Ce format est structuré comme suit :

- Un entête de 36 octets comprenant :
  - En premier un nombre de quatre octets représentant la version du format ;
  - Ensuite un nombre de quatre octets représentant le type de format des pixels (RVB, RVBA, niveaux de gris, etc.) ;
  - Au neuvième, treizième et dix-septième octet, on trouve les informations de résolution du volume (sa longueur, sa largeur et sa hauteur) représentée chacune par un nombre de quatre octets ;
  - Au vingt-et-unième, vingt-cinquième et vingt-neuvième octet se trouvent les informations d'échelles des voxels (un voxel étant un pixel en trois dimensions), représentées par trois nombres de quatre octets ;
  - Enfin, un nombre de quatre octets qui représente le format des données.
- Un paquet de données binaires représentant les intensités des pixels.

Quand je suis arrivé en stage, il existait déjà un greffon reader bvd. Cependant, le format était considéré comme incomplet puisqu'il ne prenait pas en compte le format des données DICOM (à savoir Hounsfield, SUV, MRI ou autres, qui correspondent à des unités de mesure particulières dans le milieu médical). Il m'a donc été demandé de réécrire le reader existant pour prendre en compte cette nouvelle composante. Néanmoins, certains fichiers exploités par l'équipe de MINT étaient des fichiers bvd de la « première version ». Ainsi, après discussion avec mon encadrant, j'ai décidé de ne pas remplacer le greffon déjà existant mais de lui ajouter une nouvelle méthode, de sorte que l'on puisse aussi bien l'utiliser pour lire des anciennes données bvd que des nouvelles.

En ce qui concerne son fonctionnement, le reader procédait comme suit :

- Le greffon ouvre le fichier spécifié et lit le premier caractère. Ce caractère correspond à la version du format bvd (pour le moment, soit 1 qui correspond à

l'ancien, soit 2, qui correspond au nouveau format prenant en compte le format des données) ;

- En fonction de cette version, le greffon appelle l'ancienne ou la nouvelle méthode de lecture ;
- La méthode de lecture alloue une structure de 32 ou 36 octets (en fonction de la version) et y place les informations de l'entête ;
- Le greffon crée une donnée qui a les caractéristiques de l'entête, puis y injecte les données contenues dans le fichier ;
- Le greffon copie cette donnée créée dans son « output ».

Ensuite, j'ai développé le greffon writer bvd, qui n'existait pas encore. Il fonctionnait à l'inverse du reader ; le greffon injectait dans une structure les différentes informations de la donnée en cours de traitement, puis les enregistrait dans un fichier sur le disque dur.

Le fait d'avoir développé ces deux greffons en même temps m'a permis de pouvoir immédiatement les tester : J'importais une donnée dans MINT-viewer, je l'enregistrais avec le writer bvd puis j'importais ce nouveau fichier directement après avec le reader bvd.

4.2.2 Filtre de modification de représentation

Dans MINT, les données possèdent des informations d'entête, précédemment décrites dans le chapitre 4.2.1. Ces données ont un impact sur la visualisation de l'objet ; en effet, comme le montre la figure 6, un même objet peut être rendu de façon complètement différente selon son entête.

Aussi, il m'a été demandé de développer un greffon de traitement qui avait pour but de modifier les informations d'entête d'une donnée à travers une interface graphique simple car il n'existait pas de moyen clair et rapide pour réaliser ce genre d'opération (un moyen aurait été de modifier directement les octets du fichier de la donnée, par exemple).

Image	Taille d'un voxel en x	Taille d'un voxel en y	Taille d'un voxel en z	Format de la donnée
	1m	1m	1m	Standard
	1m	1m	1m	MRI
	0.963m	0.247m	0.536m	Standard

Figure 6 – Une image de base (ligne 1) rendue avec deux entêtes différents : la deuxième image est considérée par le renderer comme une image tirée d'un IRM, la troisième image a subi un changement d'échelle de ses voxels (c'est-à-dire que ces voxels ne sont pas des cubes de $1m^3$, mais des pavés droits de 963mm*247mm*536mm).

Le greffon que j'ai développé offrait donc une interface graphique qui permettait de modifier certains paramètres de l'entête (à savoir le format de la donnée, l'échelle de mesure des voxels et l'échelle de valeur des voxels).

4.2.3 Intégration d'une image de fond

L'écran de visualisation des données de MINT-viewer, c'est-à-dire la zone où sont affichées les données, est composée de deux parties.

- La partie en arrière-plan, où est affiché un fond de couleur unie, ou un fond avec un dégradé de couleur ;
- La partie en premier plan où se trouve la donnée en elle-même.

Cependant, dans certains cas, il peut être intéressant d'afficher la donnée non pas sur un fond uni ou dégradé, mais sur une image, un motif (par exemple, on peut imaginer qu'on ait besoin d'afficher un volume représentant un organe par-dessus un motif fait de points, où chaque point serait espacé de 1cm des autres points. Cela permettrait de mesurer assez simplement certaines régions de l'organe). Le logiciel ne permettait pas encore cette possibilité. On pourrait se dire qu'il suffit d'importer un modèle 3D plat, qui ne serait en fait qu'une texture. Mais cette solution ne conviendrait pas : d'abord, le motif importé ne ferait pas forcément toute la taille de l'écran, de plus, il serait affecté par les différentes actions de l'utilisateur (rotations, zooms) alors qu'on souhaite justement que ce motif reste fixe. C'est pourquoi le besoin d'offrir aux utilisateurs de MINT-viewer cette possibilité s'est imposé.

Le développement de cet ensemble de fonctionnalités s'est déroulé en trois étapes :

- Elaboration d'un nouveau type de donnée ;
- Développement d'un greffon starter pour charger le motif ;
- Développement d'un greffon de rendu pour afficher l'image.

4.2.3.1 Création d'un nouveau type de données

Afin de gérer cette nouvelle fonctionnalité et comme MINT le permettait, j'ai créé un nouveau type de données, c'est-à-dire que j'ai ajouté au toolkit une nouvelle classe pour représenter la donnée d'image de fond dont j'allais me servir ensuite.

Cette donnée possédait un seul attribut : une chaîne de caractères représentant le chemin vers une image du disque dur.

La prochaine étape était d'écrire un greffon starter de façon à spécifier la façon dont était créée et se comportait ma nouvelle donnée.

4.2.3.2 Écriture du starter

Le greffon starter devait permettre à l'utilisateur de construire une nouvelle donnée de type « image background » en choisissant un fichier image de son disque dur. Je me suis donc tout naturellement orienté vers l'utilisation d'une classe de Qt, *QFileDialog*, qui permet d'introduire dans un logiciel des fenêtres utilitaires de type chargement de fichier ou sauvegarde de fichier. Ainsi, le greffon starter de création d'image de fond stockait dans une

chaîne de caractères le chemin d'un fichier image choisi par l'utilisateur à travers une fenêtre de *QFileDialog*. Le greffon créait ensuite une donnée de type image background et injectait ce chemin de fichier dans la donnée.

4.2.3.3 Écriture du renderer

L'écriture du greffon de rendu des images de fond fut la tâche la plus compliquée de ces trois premières semaines de stage. En effet, le fonctionnement de ce greffon reposait sur des notions techniques (rendu 3D et OpenGL) et mathématiques (géométrie dans l'espace) que je ne maîtrisais pas ou peu. Il m'a donc fallu un long temps de préparation avant de commencer à pouvoir le développer, d'abord pour appréhender de nombreuses nouvelles notions et ensuite pour analyser les mécanismes qui rentraient en jeu dans ce greffon de rendu.

Ainsi, aidé par mon encadrant, j'ai pu, à la lecture de certains points précis de la documentation d'OpenGL, à l'analyse de morceaux de codes proposant des solutions assez proches de ce dont j'avais besoin, et à l'étude de certains objets mathématiques, tels que les quaternions, intégrer les notions utiles et nécessaires au développement du greffon de rendu.

Le développement du greffon a représenté environ un tiers du temps de travail total que j'ai passé sur les greffons (à savoir une semaine sur les trois). Enfin, après diverses corrections à la fois dans mon greffon et dans la gestion de la caméra du toolkit, le greffon, dont un aperçu final est visible en figure 7, était terminé.

Le développement de ces cinq greffons a été très bénéfique pour moi. Il m'a d'abord appris à respecter rigoureusement des normes de codage. Ensuite, il m'a fait progresser de façon notable sur le plan technique. De plus, cela m'a forcé à faire preuve d'initiative et à porter une analyse attentive sur les problèmes et leurs solutions possibles.

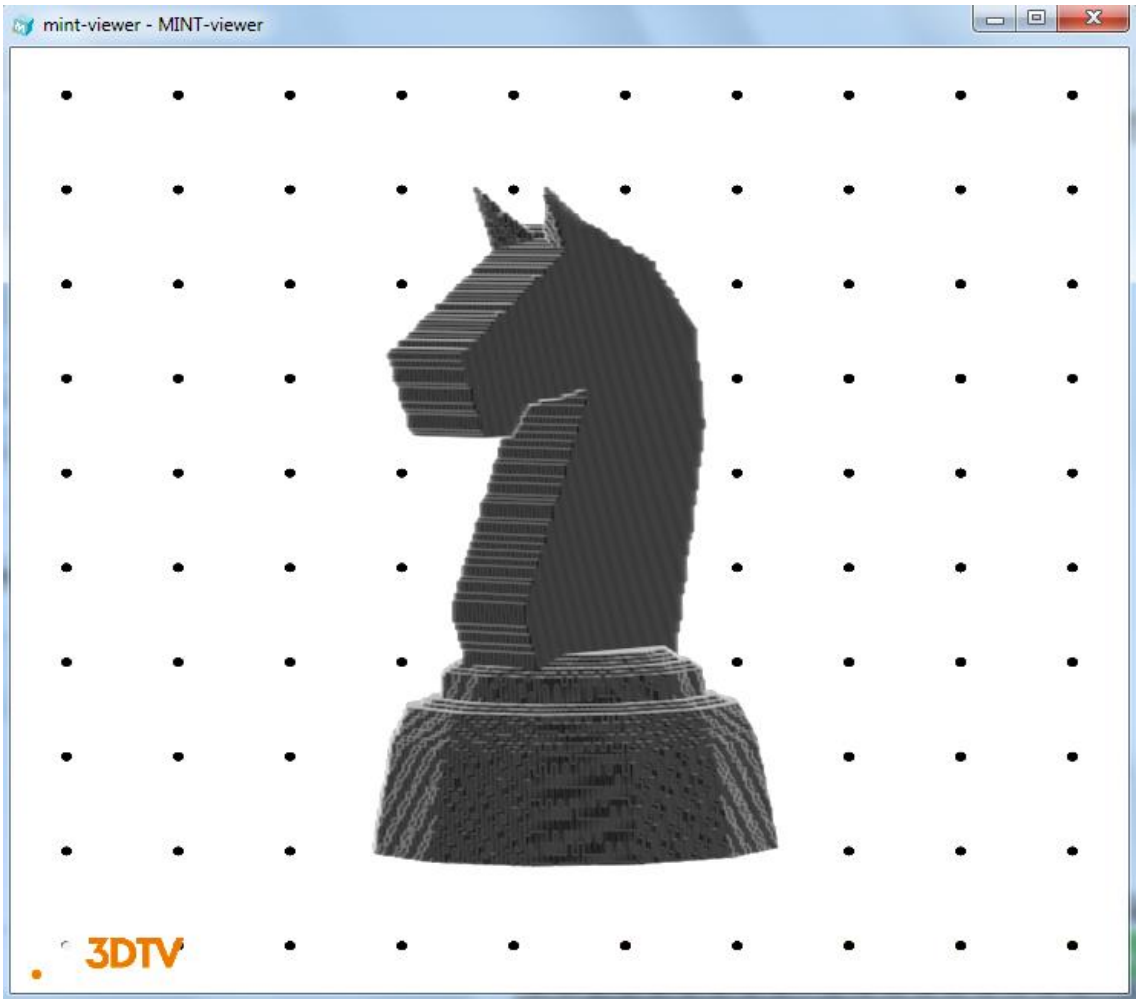

Figure 7 – Rendu final de l'ensemble des fonctionnalités d'image de fond de MINT-viewer.

Enfin, c'est grâce à tout ce travail d'analyse et de compréhension du toolkit que j'ai pu aborder sereinement l'objectif principal de mon stage, à savoir le développement de MINT-builder.

5 Développement de MINT-builder

Avant de pouvoir convenablement démarrer la conception de MINT-builder, j'ai effectué une phase d'analyse.

Après analyse du cahier des charges (dont les idées principales ont été évoquées au chapitre 3.2), j'ai pu isoler deux aspects principaux du logiciel :

- L'aspect graphique, qui englobe toutes les caractéristiques et contraintes en rapport avec l'interface graphique ;
- L'aspect moteur, qui regroupe les fonctionnalités, les mécanismes du logiciel.

5.1 Analyse et développement de la partie graphique

5.1.1 Éléments graphiques

L'analyse des caractéristiques graphiques du logiciel m'a permis de dégager des idées que j'ai pu ensuite corréler avec des concepts programmatiques.

- Un schéma correspond à une zone de travail dans lequel l'utilisateur ajoute et déplace des éléments graphiques ; cette définition correspondait en tout point aux mécanismes proposés pas la classe *QGraphicsView* de Qt (il s'avérait en fait, après une étude approfondie, qu'il n'y avait pas d'autre solution envisageable à moins de développer un système à partir de zéro qui copierait le mécanisme du *QGraphicsView*) ;
- Un greffon doit avoir une représentation graphique qui permette de rendre compte du nombre de ses entrées et sorties, et d'autres éventuelles caractéristiques (paramètres, choix de fichier, etc.), j'avais donc choisi de représenter le greffon par un *QGraphicsItem*, qui est une classe de Qt permettant de construire une figure, avant de m'apercevoir, après discussion avec mon encadrant, qu'il était plus judicieux de le représenter par un *QGraphicsRectItem*, ayant ceci de particulier qu'il correspondait de base à une figure rectangulaire ;
- Les caractéristiques d'un greffon (ses entrées, sorties, boutons de paramétrage, etc.) pouvaient toutes être considérées comme des boutons. Ainsi, si Qt possède une classe *QPushButton* pour représenter un bouton, il était plus judicieux de définir un bouton comme étant un élément graphique carré ou rectangulaire, et donc comme une entité représentée par un *QGraphicsRectItem*, en particulier parce qu'un greffon était représenté par un *QGraphicsRectItem*, et qu'il est possible (et très simple) d'ajouter une figure graphique, comme un autre

QGraphicsRectItem, dans un *QGraphicsRectItem*, mais que cela n'est pas possible avec un élément d'interface graphique comme le *QPushButton* ;

- On devait pouvoir lier des greffons entre eux, il était donc important de considérer un lien comme un élément graphique, afin de pouvoir l'intégrer au schéma. Qt proposait une classe *QGraphicsLine* qui me semblait être une bonne solution pour représenter un lien. Cependant, M. Laurent Lucas, responsable du développement de MINT, m'a fait remarquer que représenter un lien par une ligne risquerait de masquer les greffons dans certaines configurations (voir figure 8), ce qui n'était pas souhaitable. Après intégration de cette nouvelle contrainte, j'ai décidé d'utiliser plutôt la classe *QPolygon*, qui peut être considéré comme un conteneur de points, afin de pouvoir dessiner des liens « intelligents ».
- L'interface est découpée en deux zones :
    - Une zone contenant la liste des greffons. Cette liste constituant un ensemble de données ayant pour vocation d'être traitée de façon textuelle plus que graphique, avec une dimension taxinomique (il y a des « groupes » de greffons, tels que les renderers, les filters ou bien les writers, et ces groupes contiennent des membres, comme par exemple, dans le cas du groupe writer, le writer-bx, le writer-bvd, ou encore le writer-xyz), la bibliothèque Qt m'offrait trois choix d'implémentations pour gérer cet ensemble :
        - Le *QTableWidget* implémentant un modèle tabulaire, qui était un modèle intéressant du fait que les greffons peuvent être considérés comme des entrées d'un tableau (avec un nom, un auteur, un type d'entrée et un type de sortie) mais présentant le désavantage de ne pas offrir de structure arborescente ;
        - Le *QListWidget* représentant une simple liste, que j'ai écarté très vite du fait de son manque à la fois de la dimension tabulaire et de la dimension taxinomique ;
        - Le *QTreeWidget*, modélisant une liste structurée sous forme d'arbre avec colonnes, permettant alors une conception de classification et de représentation tabulaire. C'est donc vers ce troisième modèle de représentation que je me suis tourné, le jugeant être le mieux adapté de tous.
    - Une zone contenant un environnement multi-fenêtré, chaque sous fenêtre contenant un schéma. Trois mécanismes existaient pour gérer ce concept :
        - Le SDI (Single Document Interface), consistant à ouvrir une nouvelle fenêtre à chaque fois qu'un nouveau document est créé ou chargé. Il s'agissait d'une méthode trop intrusive et qui aurait gravement nuit à l'ordonnance de l'espace de travail. De plus, Qt

ne gérait pas nativement ce système. Pour toutes ces raisons, je n'ai pas retenu cette solution ;

- Le TDI (Tabbed Document Interface), géré par Qt à travers la classe *QTabWidget*, permettant de structurer son espace de travail avec des onglets, solution beaucoup plus ergonomique que la précédente mais qui présentait le défaut majeur de ne montrer le contenu que de l'onglet courant, ce qui pouvait être gênant si l'utilisateur avait souhaité, par exemple, comparer deux schémas ;
- Le MDI (Multiple Document Interface), géré par Qt à travers la classe *QMdiArea*, système permettant de créer plusieurs fenêtres et de les placer dans une zone principale, et de soit maximiser la fenêtre courante, soit afficher toutes les sous-fenêtres côte à côte ; la solution du MDI m'a donc semblée être la plus adaptée au problème.

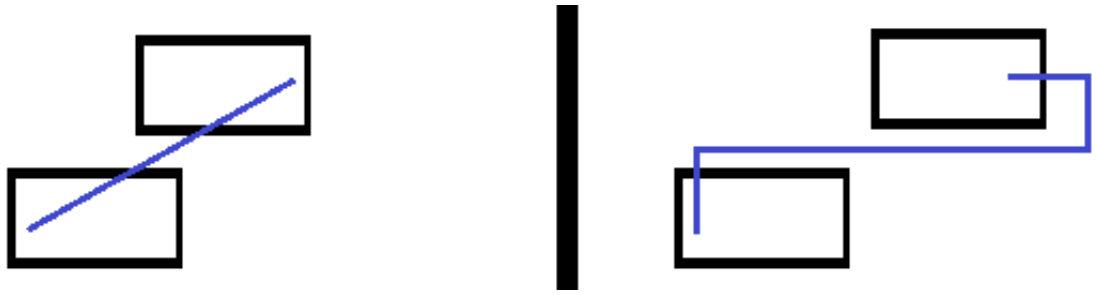

Figure 8 – Schéma d'un lien en simple ligne (à gauche) versus un lien « intelligent » (à droite)

5.1.2 Menu et barre d'outils

Deux éléments graphiques dont il n'avait pas été question dans le cahier des charges mais qui se trouvent être des éléments indispensables à toute interface graphique sont la barre de menu et la barre d'outil.

La barre de menu est une barre située en haut de la fenêtre de l'application et qui regroupe, suivant des mots-clés spécifiques, différentes actions réalisables par l'utilisateur. La barre d'outils, située généralement directement en dessous de la barre de menu, est une barre contenant des boutons figurant les actions les plus usuelles de la barre de menu.

J'ai donc en premier lieu réfléchi aux mots-clés correspondant aux actions possibles. J'ai choisi les rubriques classiques « Fichier », contenant les actions relatives au fichier courant tel que nouveau, ouvrir et sauvegarder, « Édition », proposant une liste d'actions relatives aux composants du fichier comme les fonctions de copier/coller ou de sélection, et « Aide »,

auxquelles j'ai ajouté les menus « Outil » qui permet d'avoir accès aux options du logiciel, «Exécution » contenant les actions d'exécution du schéma et « Fenêtre » gérant les différentes sous-fenêtres de la zone MDI.

J'ai ainsi créé toutes les actions du menu et de la barre d'outils, de façon graphique. Le développement de leurs fonctionnalités sera abordé dans le chapitre 5.3.

5.1.3 Mise en place des composants graphiques

Avec la bibliothèque Qt, la façon courante de développer une interface graphique pour une application est d'abord de créer une classe héritant de *QMainWindow*, qui est la classe de base pour représenter une fenêtre principale. Ensuite, il faut créer et ajouter le menu et la barre d'outils à la fenêtre.

Mon encadrant de stage, M. Romain Guillemot, avait développé pour la création de l'application MINT-viewer une série de classes qu'il a regroupées sous la forme de bibliothèques personnelles. Ces bibliothèques ne sont pas fournies aux développeurs de plugins, cependant, j'ai pu m'en servir afin de faciliter le développement de mon interface graphique.

En effet, une des branches de ces bibliothèques propose une classe abstraite, *CAbstractMainWindow*, *spécialisable*[2], permettant la création d'une fenêtre avec barre de menu et barre d'outils très rapide (voir figure 9). De plus, héritant de la classe Qt *QMainWindow*, cette classe en propose les mécanismes de base, implémentant notamment des méthodes directes pour ajouter des widgets dans la fenêtre à des emplacements spécifiques.

Plutôt que d'écrire le code complet pour générer l'interface graphique de MINT-builder, j'ai donc choisi d'utiliser les outils offerts par MINT-toolkit. J'ai ainsi pu regrouper tous les composants graphiques d'interface (le *QTreeWidget*, le *QMdiArea,* ainsi que le *QMenuBar* (pour le menu) et la *QToolBar* (pour la barre d'outils) qui étaient déjà gérés par la classe du toolkit), comme le montre la figure 10.

[2] Spécialisable : bien que courant dans les écrits techniques, ce mot n'apparaît pas dans la plupart des dictionnaires français et semble de fait être un néologisme.

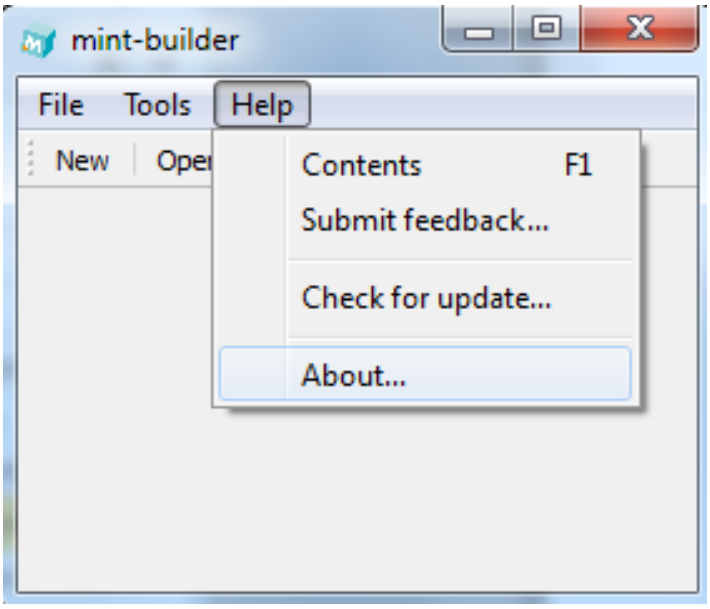

Figure 9 – Fenêtre graphique construite par la classe *CAbstractMainWindow* du toolkit.

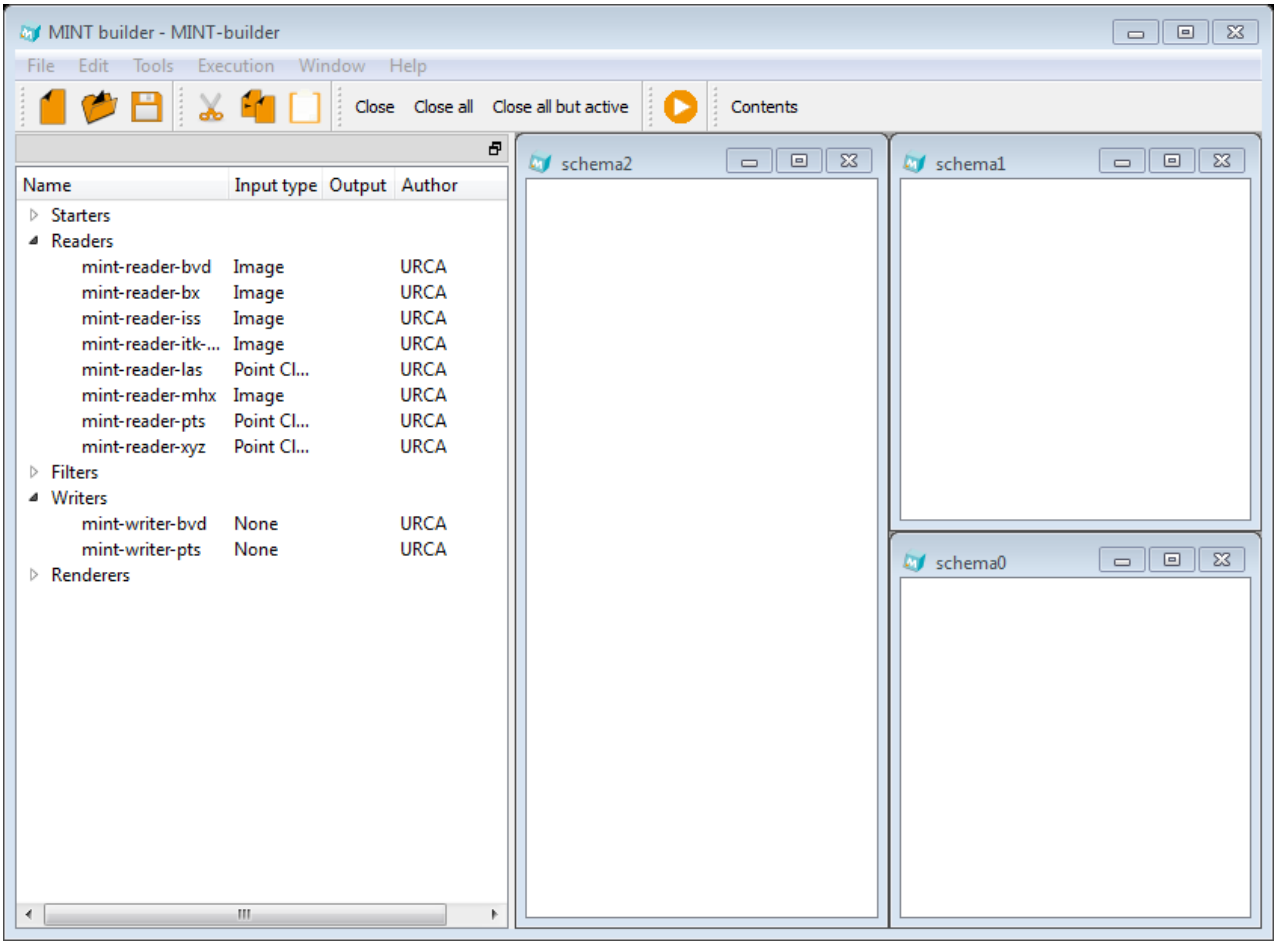

Figure 10 - L'interface graphique de MINT-builder complétée

5.2 Analyse et développement du moteur

Après avoir dégagé tout l'aspect graphique du logiciel, j'ai pu me pencher plus en détail sur l'analyse de ses fonctionnalités.

Il s'est avéré que le fonctionnement global de MINT-builder se rapprochait beaucoup d'une autre application, Toolbox Image (voir figure 11), qui avait été développée par M. Philippe Vautrot, maître de conférences et membre du groupe SIC du CReSTIC. Cette application, dont le développement a commencé en 2003, est une application regroupant différents greffons de traitement d'image 2D, et offrant la possibilité pour un utilisateur d'écrire ses propres greffons. Toolbox Image a été conçue pour être utilisée dans un cadre pédagogique, et j'ai moi-même eu l'occasion de m'en servir pendant mes cours de Traitement Numérique de l'Image.

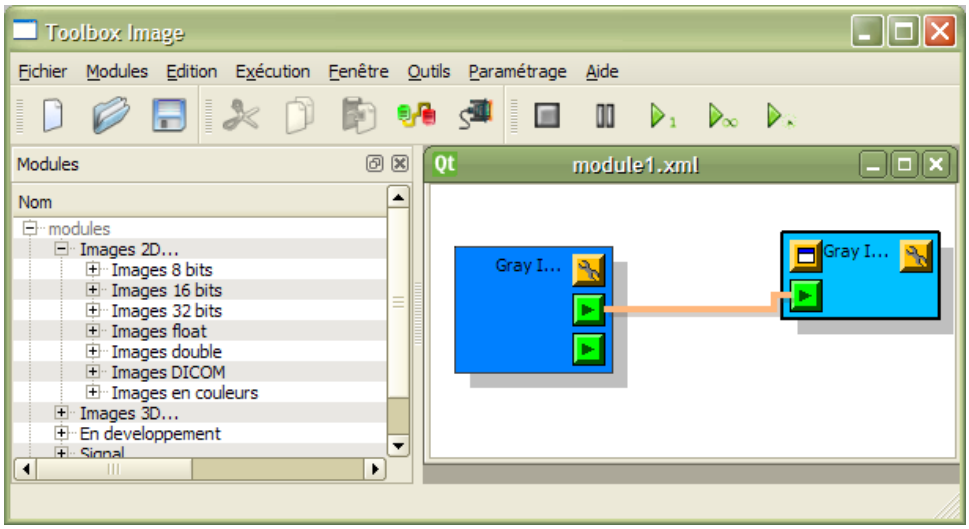

Figure 11 – MINT-builder s'inspire d'un logiciel conçu par M. Philippe Vautrot, Toolbox Image. On remarque la même conception de l'interface avec une partie en arbre à gauche et une scène graphique à droite. De plus, Toolbox Image permet d'intégrer des greffons à un schéma puis de les relier.

C'est pourquoi il m'a semblé judicieux de demander à M. Philippe Vautrot s'il était possible de me donner un accès au code source de son application afin de pouvoir étudier et reprendre certains mécanismes de l'application , ce qu'il a accepté.

5.2.1 Implémentation des greffons

L'étape centrale du développement de MINT-builder a été la conception des mécanismes des greffons.

5.2.1.1 Objet graphique greffon

De façon graphique, j'avais décidé de les représenter comme des boîtes qui contiendraient le nom du greffon, ainsi que des boutons figurant les entrées, sorties, et les caractéristiques particulières du greffon.

La première question que je me suis posée a été de savoir comment je pourrais relier mon objet graphique avec le greffon lui-même, c'est-à-dire avec le fichier physique du greffon, un fichier dll, contenant ses instructions. Là encore, c'est l'étude de MINT-toolkit qui m'apporta la solution. En effet, l'API proposait une série de méthodes, au sein de la classe *CMintPluginManager*, pour manipuler les greffons :

- getAvailableNames qui retournait, sous forme de chaînes de caractères, les noms des greffons disponibles ;
- createInstance qui, en prenant en paramètre le nom d'un greffon, nous retournait un objet de type *CMintAbstractPipelineObject*, classe représentant le greffon en lui-même ;
- releaseInstance, utilisée pour désallouer la mémoire utilisée par un greffon une fois son utilisation terminée.

Je pouvais donc, à partir du nom d'un greffon, en créer une instance et la manipuler. Cela était possible du fait qu'un greffon était identifié par son nom, c'est-à-dire que tous les greffons avaient un nom différent.

Après analyse de ces faits et discussion avec mon encadrant, j'ai conclu qu'un objet graphique greffon ne nécessitait pour être construit qu'une chaîne de caractères (son nom) et sa position sur le schéma.

Ensuite, depuis cette chaîne de caractères, j'ai pu récupérer l'objet greffon, analyser son type et ses informations afin d'ajouter les nœuds d'entrée et de sortie, et les boutons caractéristiques (pour les readers et les writers, un bouton de choix de ficher, cliquer sur ce bouton ouvre une fenêtre de choix de fichier et transmet le résultat au greffon ; pour les renderers un bouton de visualisation, dont le fonctionnement sera développé au chapitre 5.1.2.3, et enfin, pour les greffons qui possèdent des paramètres, comme on peut le voir sur la figure 12, un bouton permettant l'affichage de l'interface de modification de ces paramètres).

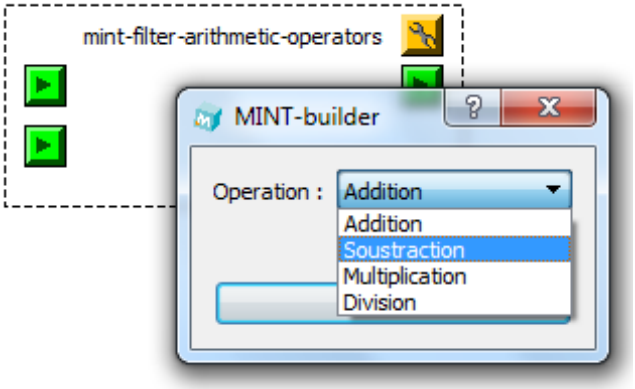

Figure 12 – L'appui sur le bouton de paramètre d'un greffon (en forme de clé, en haut à droite) déclenche l'ouverture d'une boîte de dialogue contenant les paramètres réglables du greffon.

Dans le code de la Toolbox Image, un greffon, en plus d'être représenté par un rectangle, possède une ombre (voir figure 11). Cela permettait de simuler l'existence d'un axe z. Cependant, en me fondant sur l'utilisation que j'avais faite de la Toolbox Image, je me suis souvenu que le fait que deux greffons pouvaient se chevaucher n'était pas très pratique et rendait souvent la lecture du schéma plus périlleuse. Après en avoir discuté avec M. Laurent Lucas, j'ai décidé d'implémenter une méthode de collision qui permettait d'empêcher deux objets graphiques greffons de se superposer, en me servant de leurs propriétés graphiques (notamment de leur *Bounding Box,* c'est-à-dire le rectangle, au sens mathématique du terme, qui englobe la forme graphique).

5.2.1.2 Nœuds d'entrée et nœuds de sortie

Chaque greffon possédait un certain nombre d'entrées et de sorties (qui pouvait être zéro). Ces entrées et sorties étaient représentées par des petits carrés, appelés nœuds. Puisque la gestion et l'affichage de ces nœuds étaient identiques conceptuellement dans MINT-builder et dans la Toolbox Image, j'ai pris le parti de reprendre le code de M. Philippe Vautrot et de l'intégrer directement au logiciel.

Ce faisant, il était important de noter la différence principale entre un nœud d'entrée et de sortie, qui semblaient conceptuellement identiques (un carré vert qui a pour caractéristique un type de données) : un nœud d'entrée possède en attribut un pointeur vers un nœud de sortie (qui représente le nœud de sortie vers lequel il est lié) alors qu'un nœud de sortie possède en attribut une liste de pointeurs de nœuds d'entrée (qui représente l'ensemble des nœuds d'entrée connectés à cette sortie). Cela s'explique du fait que, aussi bien dans la Toolbox Image que comme le spécifiait le cahier des charges (voir chapitre 3.2), une sortie peut être connectée à plusieurs entrées mais une entrée ne peut être reliée qu'à une seule sortie.

5.2.1.3 Bouton de visualisation

Le bouton de visualisation des greffons de rendu avait pour but d'ouvrir une fenêtre de rendu dans lequel était affichée la donnée liée. Cependant, cette fonctionnalité s'est révélée être plus compliquée que prévu.

MINT-builder était susceptible de manipuler des données très lourdes (des volumes). En outre, il était possible qu'une seule donnée en sortie d'un greffon soit reliée à deux greffons de rendu. Ainsi, il était préférable, le cas échéant, dans un souci d'optimisation, de partager les ressources.

Cela posait un problème majeur : OpenGL ne permettait pas le partage de tous les types d'objets. En particulier, les VAOs (Vertex Array Objects, des tableaux qui enregistrent des états OpenGL) et le FBOs (Frame Buffer Objects, servant à faire du rendu dans une texture), qui sont des éléments essentiels au rendu par OpenGL ne sont pas partageables.

M. Romain Guillemot, mon encadrant, qui est le principal développeur de MINT, avait déjà rencontré ce problème. Il m'a donc exposé la solution qu'il avait implémentée pour pallier ce problème, que nous avons ensuite adapté à MINT-builder, d'abord parce qu'il s'agissait d'une solution optimale, et ensuite parce qu'étant déjà implémentée dans MINT-viewer, nous avions tous les outils à portée de main pour la mettre en place.

Cette solution consistait à créer un contexte OpenGL global, et, pendant que ce contexte était actif, rendre l'image dans une texture, car une texture est une donnée partageable. Ensuite, il fallait transmettre cette texture aux renderers.

5.2.1.4 Liens

5.2.1.4.1 « Liens intelligents »

Afin de permettre un affichage correct, les liens ne devaient pas masquer les greffons, mais plutôt les contourner, le cas échéant (voir figure 8). J'ai repris l'algorithme permettant de gérer cette fonctionnalité à partir du code de la Toolbox Image, puisque nous voulions obtenir le même type de rendu.

Ensuite, j'ai appliqué cet algorithme à chaque fois que le lien était susceptible de bouger, donc à chaque fois qu'un greffon qui était lié était déplacé.

5.2.1.4.2 Problème de boucles infinies

Avec ce système, il était possible de relier une entrée d'un greffon avec une sortie de ce même greffon. En soi, puisque les greffons n'étaient pas exécutés, cela n'était pas gênant. Cependant, au moment où cela serait mis en place, une telle situation mènerait à une situation de boucle infinie, qui provoquerait, au bout d'un certain temps, la fermeture soudaine du programme. Pour éviter ce problème, je me suis intéressé à la théorie des

graphes et plus particulièrement aux algorithmes de détection de graphes cycliques. J'ai ainsi pu mettre en place un ensemble de fonctions qui s'assurent que l'utilisateur n'est pas en train de générer une boucle infinie, comme illustré dans la figure 13.

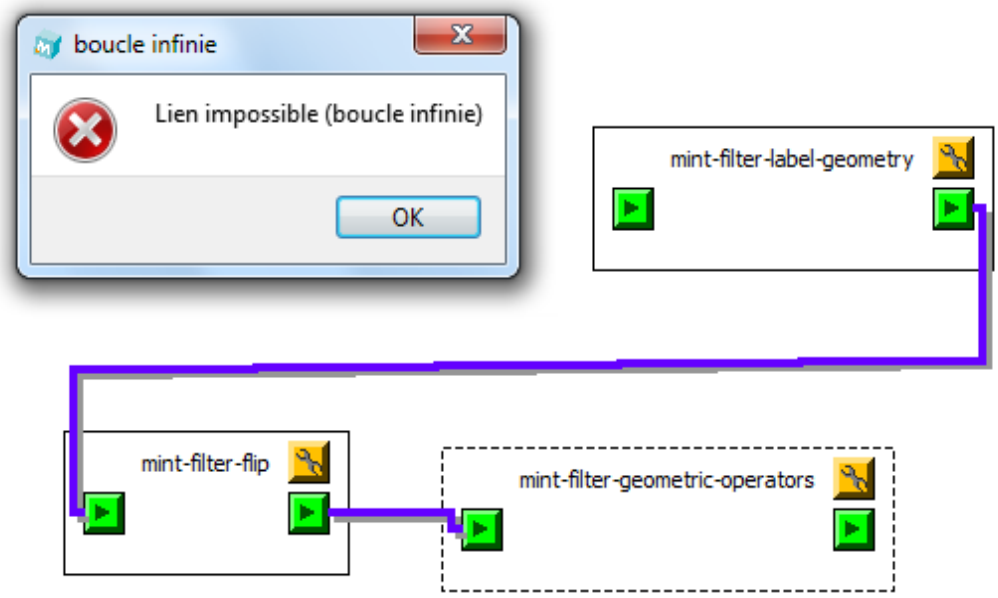

Figure 13 – Lorsqu'une boucle infinie est détectée (ici, si l'utilisateur essaye de relier la sortie du mint-filter-geometric-operators avec l'entrée du mint-filter-label-geometry), un message d'erreur s'affiche et le lien n'est pas créé.

5.2.2 Glisser-déposer

5.2.2.1 Déplacement d'un greffon vers un schéma

Le glisser-déposer[3] est une méthode consistant à utiliser une souris, pavé ou écran, pour déplacer d'un endroit à un autre de l'interface une information. Ainsi, cette méthode semblait être parfaitement adaptée au placement de greffons sur un schéma, puisque cela consistait au déplacement d'une information (un greffon) d'un endroit (le *QTreeWidget*) à un autre (le schéma) de l'interface.

Après quelques recherches dans la documentation de Qt, j'ai découvert que cette bibliothèque mettait en place des fonctionnalités pour gérer le glisser-déposer à travers différentes classes :

[3] Glisser-déposer (ou cliquer-glisser) : traduction recommandée par la DGLFLF (Délégation Générale à la Langue Française et aux Langues de France) et l'OQLF (Office Québécois de la Langue Française) pour l'expression anglaise « Drag and Drop ».

- *QMousePressEvent*, classe d'évènements permettant de détecter le moment où l'utilisateur presse un bouton de la souris ;
- *QMimeData*, classe contenant les données que l'on souhaite déplacer ;
- *QDrag*, permettant le support du *QMimeData* pour le mécanisme du drag and drop ;
- *QDragEnterEvent*, classe d'évènements permettant de spécifier un traitement particulier lorsqu'un objet de type *QDrag* entre dans une zone spécifique, servant principalement à vérifier que le type de donnée contenu dans le QDrag est cohérent et peut-être déposé ;
- *QDropEvent*, classe d'évènements permettant de spécifier un traitement particulier lorsqu'un objet de type *QDrag* est déposé dans une zone spécifique.

Le schéma d'implémentation de cette fonctionnalité consistait donc, au moment où l'utilisateur presse un bouton de la souris (en général, le bouton gauche), à charger une donnée dans un *QMimeData*, injecter ce *QMimeData* dans un *QDrag*, puis implémenter les évènements d'entrée et de dépôt dans une zone particulière.

La première tâche fut donc de me pencher sur la classe *QMimeData* et de me demander de quelle façon il était possible de représenter un greffon afin de l'injecter dans un objet de type *QMimeData*.

La classe *QMimeData* permettait de gérer plusieurs types de données : le format couleur (RGBA), le format Image, le format html (pour stocker le code source d'une page internet), le format URL et enfin le format texte. Le nom d'un greffon lui servant d'identificateur, j'ai choisi de charger dans le *QMimeData* le nom du greffon en format texte, c'est-à-dire en tant que simple chaîne de caractères, sélectionné dans le *QTreeWidget* lors de l'évènement *QMousePressEvent*.

J'ai ensuite implémenté un *QDragEnterEvent* rattaché à chaque sous-fenêtre du *QMdiArea* qui avait pour rôle de s'assurer que le type de donnée tenue par le *QMimeData* du *QDrag* entrant était bien au format texte, puis de vérifier que la chaîne de caractères correspondait bien au nom d'un greffon. Si ces deux conditions étaient vérifiées, l'évènement était accepté, sinon, il était refusé.

La dernière étape a été l'implémentation du *QDropEvent* rattaché au *QMdiArea*. Si l'évènement de cliquer-déposer avait été refusé, rien ne se passait, sinon, un objet graphique greffon était créé avec pour paramètre la chaîne de caractères transportée par le *QMimeData* du *QDrag*, et positionné aux coordonnées de la souris.

5.2.2.2 Connexion d'une entrée vers une sortie

Pour connecter une entrée vers une sortie, j'avais imaginé plusieurs implémentations possibles.

D'abord, proposer un bouton, dans la barre d'outils, pour créer un lien entre deux greffons sélectionnés. J'ai dégagé de cette idée deux problèmes majeurs, d'abord, il aurait été complexe en terme d'algorithme de détecter, dans le cas de greffon multi-entrées et multi-sorties, quelle entrée devait être reliée à quelle sorite. Ensuite, le fait de lier deux greffons était probablement une des actions, si ce n'est l'action, que l'utilisateur accomplirait le plus souvent. Ainsi, je trouvais que faire passer cette action à travers deux étapes (sélectionner les greffons, puis cliquer sur un bouton) n'était pas assez ergonomique.

Ensuite, j'avais imaginé que l'utilisateur puisse cliquer droit sur un nœud, puis cliquer droit sur un autre nœud afin de les relier. Cette solution était bien plus ergonomique mais me gênait du fait que traditionnellement, le clic droit est réservé aux menus contextuels.

Enfin, j'ai voulu implémenter un système de cliquer-déposer. En effet, il paraissait très ergonomique de pouvoir créer un lien en effectuant une action de cliquer-glisser entre les deux nœuds qui nous intéressent. De plus, c'était la solution qu'avait choisie M. Philippe Vautrot pour son logiciel.

Cependant, je me suis heurté à un problème. Comment représenter la donnée d'un nœud ? En effet il ne pouvait s'agir ni d'un texte, ni d'une couleur, ni de toute autre information gérée par le *QMimeData*. En me penchant sur le code de la Toolbox, la solution que M. Vautrot avait choisie me paraissait complexe et discutable (il a choisi de convertir l'adresse du pointeur du nœud en nombre entier, qu'il pouvait passer au *QMimeData*, puis lors du *QDropEvent*, de transformer ce nombre en pointeur). J'ai donc décidé, après discussion avec mon encadrant et M. Vautrot lui-même, d'utiliser une autre méthode, beaucoup plus simple et beaucoup moins dangereuse. J'ai décidé de créer une classe héritant de *QMimeData*, et possédant en attribut un pointeur vers le nœud. Ainsi, en suivant le schéma du système de cliquer-déposer (*cf*. chapitre 5.2.2.1), je pouvais transporter directement le pointeur de mon nœud. Il n'y avait qu'un seul défaut : cette implémentation ne permettait pas de transporter ce pointeur à travers différentes applications, mais seulement au sein de MINT-builder. Cependant, cela n'était pas gênant dans le sens ou seul MINT-builder devait interpréter ce nœud.

5.3 Fonctionnalités de l'interface

J'avais, à l'aide de MINT-toolkit, géré convenablement la construction de la barre de menu et de la barre d'outils. Cependant, il ne s'agissait que d'une façade graphique, car aucun évènement n'était lié à ces actions.

5.3.1 Gestion des sous-fenêtres

MINT-builder proposait trois actions relatives aux sous-fenêtres de la *QMdiArea* :

- *Close*, pour fermer la fenêtre active ;
- *Close all*, pour fermer toutes les fenêtres ;
- *Close all but active*, pour fermer toutes les sous-fenêtres, excepté la fenêtre courante.

En plus de cela, la liste des fenêtres courantes est disponible dans le menu « Fenêtre » de la barre de menu afin de faciliter la navigation entre les fenêtres.

Ces différentes fonctions permettent de façon simple de gérer son espace de travail, ce qui se révèle être très fastidieux lorsqu'un utilisateur travaille sur une application de type MDI qui ne propose pas ce genre de fonctionnalités.

5.3.2 Fonctions de lecture et d'écriture

Comme le précisait mon cahier des charges, MINT-builder devait permettre la sauvegarde et le chargement de schémas en tant que fichiers XML.

La Toolbox Image utilisait déjà ce principe, de fait, j'ai analysé le code responsable de la sauvegarde et du chargement des schémas du logiciel de M. Vautrot. Cependant, j'ai remarqué que ce code utilisait des classes de Qt qui étaient, avec la version de la bibliothèque que j'utilisais, dépréciées. Qt 5 proposait, à la place, d'autres classes plus flexibles pour gérer l'écriture et la lecture XML, qui n'existait pas encore au moment où la Toolbox Image a été développée, à savoir la classe *QXmlStreamReader* et *QXmlStreamWriter*. Cependant, l'analyse du code de M. Vautrot m'a tout de même permis de comprendre comment étaient structurés les schémas, et ainsi j'ai gardé cette structure pour mes propres fonctions d'écriture et de lecture qui utilisaient les nouveaux flux XML proposés par Qt.

Cependant, au bout de quelques jours, mon encadrant et moi sommes arrivés à la conclusion que ce système était incomplet. En effet, il ne permettait pas de stocker puis de restituer ni les paramètres des greffons, ni les chemins de fichier des readers et des writers.

En étudiant la documentation du toolkit, j'ai remarqué que la classe abstraite dont héritent les données et les greffons, *CMintAbstractObject*, proposait un mécanisme de propriétés. C'est-à-dire qu'on pouvait affecter aux différents objets des propriétés (sous la forme de nombres entiers, de nombres à virgule, de chaînes de caractères, ou de tableaux de ces types) identifiées par un nom, puis les récupérer afin de les utiliser ultérieurement.

Malheureusement, cette solution, bien qu'implémentée, n'a pas été exploitée par les greffons. Il faudra donc attendre sa mise en place au sein de la partie des greffons de MINT pour pouvoir l'utiliser ensuite dans MINT-builder.

5.3.3 Copier / coller

Les mécanismes de copie et de collage pouvaient être gérés, grâce à Qt, à travers le *QClipboard*. Le *QClipboard* est une sorte d'entité globale, qu'on pourrait nommer « presse-papier » en français, qui contient des données et permet ainsi de les faire voyager d'un endroit à un autre. Le *QClipboard* de Qt contient des données sous la forme d'un *QMimeData*. Ainsi, afin de pouvoir copier/coller des greffons à travers les différents schémas, j'ai décidé de réécrire une classe dérivant de *QMimeData* qui pourrait stocker un conteneur de pointeurs sur les greffons sélectionnés. Encore une fois, le seul désavantage de cette solution, comme cité au chapitre 5.2.1.2, était qu'elle ne permettait pas le transfert de ces données à travers différentes applications. Cependant, puisque les objets graphiques greffons n'étaient exploités que par MINT-builder, j'ai estimé que ce désavantage ne desservirait en rien cette fonctionnalité.

Cependant, je me suis rendu compte que ce choix n'était pas très bon et pouvait provoquer des plantages[4]. En effet, si l'on copiait un greffon, qu'on le supprimait du schéma et qu'on le collait ensuite, le pointeur contenu dans le *QMimeData* devenait invalide, provoquant ainsi la fermeture brusque du programme.

La solution que j'ai choisie pour pallier ce problème a été non pas de stocker des pointeurs, mais plutôt de stocker les informations nécessaires à la reconstruction d'un objet graphique greffon identique, à savoir son nom et ses coordonnées de position dans le schéma. Ainsi, le système de copier/coller était fonctionnel, utilisait le même mécanisme mais était beaucoup plus stable.

Après réflexion, mon encadrant m'a fait remarquer que, tout comme les fonctions de lecture et d'écriture de schéma (voir chapitre précédent), cette implémentation ne permettait pas de stocker les informations de paramètre ou de chemin de fichier. Le mécanisme de copier/coller et de lecture/écriture étant donc, conceptuellement, assez proche, cette implémentation utilisant le *QClipboard* sera certainement remplacée plus tard par une exploitation du formatage XML.

[4] Plantage : ce terme peut paraître familier, cependant il s'agit là de la traduction recommandée par l'OQLF de l'expression « crash informatique ».

5.3.4 Exécution de la chaîne

Afin de rendre l'application effective, il fallait faire en sorte que les chaînes de traitement que nous créions puissent s'exécuter. La première étape a donc été de relier les entrées avec les sorties. Cela était déjà pris en compte de façon graphique, avec le système de lien, mais il fallait désormais spécifier aux greffons concernés que la sortie de l'un devenait l'entrée de l'autre. Cette fonctionnalité a pu être mise en place grâce à MINT-toolkit qui proposait des méthodes telles que *setInput*, *getInput*, *setOuput* ou *getOuput* par rapport aux greffons.

Ensuite, l'action « exécuter » parcourait la liste de tous les greffons du schéma et appelait la méthode *update*, du toolkit, sur chacun d'eux. Le fonctionnement de cette méthode et de la propagation des données au sein du pipeline graphique était ensuite délégué au toolkit.

6 Conclusion

En conclusion, ce stage au sein du groupe SIC du CReSTIC m'a été profitable en de nombreux points.

Tout d'abord, il m'a énormément fait progresser sur le plan technique, puisque j'ai dû aborder des notions programmatiques bien plus avancées que celles que j'ai pu apprendre pendant mon cursus, ainsi que de nouveaux outils de développement. J'ai par exemple appris à me servir d'un debugger, qui m'a servi à corriger certains problèmes que j'avais du mal à détecter sans.

Ensuite, puisque le projet sur lequel j'ai travaillé est une base de travail qui sera reprise et poursuivie par d'autres personnes à l'avenir, cela m'a forcé à produire du code lisible et commenté. Il a fallu que je respecte un certain nombre de normes de codage (la façon de nommer mes variables, la façon d'indenter mon code,…) bien spécifiques.

En outre, le fait de m'être retrouvé face à des problèmes dont je ne connaissais pas la solution m'a appris à mieux analyser les problèmes et à y trouver, autant que faire se peut, des solutions en étudiant des documentations.

Enfin, ce stage fut l'opportunité de lier des relations amicales et professionnelles avec les membres du laboratoire, car le dialogue au sein d'une équipe de développement est primordial pour que chacun puisse suivre l'évolution de ses collègues, et les aider ou demander de l'aide au besoin.

Ce stage m'a donc apporté bien plus que ce que j'imaginais, autant sur le plan technique que sur le plan social, et j'ai été très satisfait de voir qu'un projet, qui me semblait au départ colossal et bien au-dessus de ma portée, pouvait être mené à bien à force d'analyse pertinente et de remise en question.