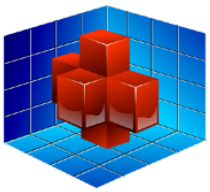
IRAMAT-CRP2A, Universite Bordeaux Montaigne
Maison de l'Archeologie, Esplanade des Antilles
33607 Pessac Cedex, France



Rapport de stage
Développement en Java d'une interface 3D d'édition de fichiers destinés à piloter DosiVox

Stage réalisé par Guitard Alan,
sur la pèriode du 17 Mai 2016 au 31 Juillet 2016

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué à la réalisation de ce stage et à la rédaction du présent rapport.

Tout d'abord, je remercie mon professeur Marc Zeitoun qui m'a redirigé cette offre de stage, offre qui m'a tout de suite intéressé, ainsi que Matthias Robine qui m'as personnellement redirigé plusieurs autres offres, bien qu'aucune d'elles n'ait été fructueuse. J'ai beaucoup de gratitude pour leur implication envers leurs étudiants.

Je remercie par-dessus tout mon maitre de stage, Norbert Mercier, pour son accueil et sa prise de temps à m'expliquer les méthodes de la dosimétrie et le "jargon" scientifique auquel je n'étais pas familier. Il m'a accordé une grande confiance et une grande indépendance durant le stage, en prenant le temps malgré le peu qu'il avait pour me guider dans le projet.

Je remercie aussi Loïc Martin, développeur de DosiVox et doctorant de Mr Mercier, qui a aussi agréablement pris de son temps malgré la rédaction de sa thèse pour m'aiguiller et m'expliquer ses attentes quand Mr Mercier était occupé.

Enfin, toute l'équipe du laboratoire et les étudiants stagiaires en Histoire de l'Art ont été des plus agréables avec moi, et je les remercie pour leur bonne humeur et pour les parties de cartes pendant les pauses-déjeuners.

Résumé

L'IRAMAT-CRP2A (Institut de recherche sur les Archéomatériaux – Centre de recherche en physique appliquée à l'archéologie) est une unité mixte de recherche du CNRS ayant pour thématique l'étude des sociétés anciennes, l'usage de leurs ressources et de leurs histoires. Dans le cadre du projet Dosi-Art, il y a été développé entre autres le logiciel DosiVox, utilisant la librairie Geant4, permettant la simulation dosimétrique dans des conditions complexes, afin de permettre de dater les échantillons par les méthodes dites de la luminescence. Par le biais de fichiers pilotes contenant des matrices et des informations scientifiques, DosiVox crée un monde voxelisé et exécute la simulation d'émissions de particules ; il écrit ensuite les résultats dans des fichiers texte qui seront eux traités par les scientifiques. Norbert Mercier, directeur de recherche au CNRS, développa il y a deux ans une interface en Visual Basic pouvant éditer ces fichiers texte. Cette interface ayant des limitations gênantes en termes d'inter-portabilité et de visualisation, il proposa alors un stage visant à programmer une interface en Java, langage qui, de par sa propriété de conception, est multi-platformes et propose diverses librairies 3D qui devaient répondre au besoin des scientifiques.

Table des matières

Chapitre 1

Introduction - Quelques explications sur le fonctionnement de DosiVox et Thématiques du stage

DosiVox, développé par Loïc Martin dans le cadre du projet Dosi-Art soutenu par le Conseil Régional d'Aquitaine, est un logiciel écrit en C++ lisant des fichiers pilotes décrivant un monde voxelisé avec un nombre défini de voxels sur les axes x, y et z accompagné de paramètres scientifiques globaux. Ces fichiers pilotent la simulation de DosiVox, qui utilise la librairie Geant4 pour simuler le passage des particules émises à travers la matière en utilisant l'algorithme de Monte-Carlo. Geant-4 réalise alors des calculs sur les interactions successives et sur la production de particules secondaires que ces interactions produisent. Ces particules secondaires vont elles-aussi créer des interactions, et ce jusqu'à une limite fixée dans le fichier pilote. A l'aide de ces informations, DosiVox simule le taux de radiations émis par les voxels, les interactions entre les particules (alpha, bêta ou gamma) et enregistre les doses déposées grace

à une sonde placée dans le monde ; il sauvegarde enfin les valeurs utiles à la datation par luminescence de l'objet en question.

Méthode de datation par luminescence La thermoluminescence, un phénomène qui apparait avec la température, décrit le fait que certaines substances émettent une lueur transitoire après avoir été exposées à un rayonnement ionisant. L'explication simplifiée est de dire que lors de son irradiation, le composé thermoluminescent a "piégé" des électrons dans des défauts de sa structure, et les rejettent quand il est chauffé produisant ainsi de la lumière. Dans son milieu naturel, un composé est constamment irradié par les radio-éléments naturels (principalement l'Uranium, le Thorium et le Potassium) et va accumuler un nombre croissant d'électrons dans ses pièges. La luminescence, la quantité de lumière émise lors de la chauffe, sera donc fonction de la durée d'irradiation du composé dans son milieu naturel. Il est important de noter que si le composé est chauffé une deuxième fois, il n'émettra plus de lumière. Les composés datés par cette technique sont prélevés dans les couches inférieures du sol, et la technique est mise en oeuvre dans un laboratoire à lumière "noire" (en fait, une très faible intensité lumineuse est admise). DosiVox a pour fonction de fournir des informations plus précises sur ces irradiations, grace à des simulations numériques, en particulier pour traiter des cas complexes pour lesquels on ne peut déterminer de solution analytique.

Enregistrements des doses radioactives par sondes Dans DosiVox, une sonde est un objet non-physique représentant le volume où les doses sont enregistrées. Elle est plus fréquemment au milieu du monde car, comme la simulation est créée dans un environnement donné, les voxels au centre du monde reçoivent plus de radiations et sont donc plus pertinents à traiter.

— Sonde verticale seulement : cylindre sur l'axe Z qui définit l'endroit où les doses radioactives sont enregistrées. Elle est divisée en segments de même longueur, définissables par l'utilisateur à l'instar de son diamètre. Cette sonde est toujours présente, mais

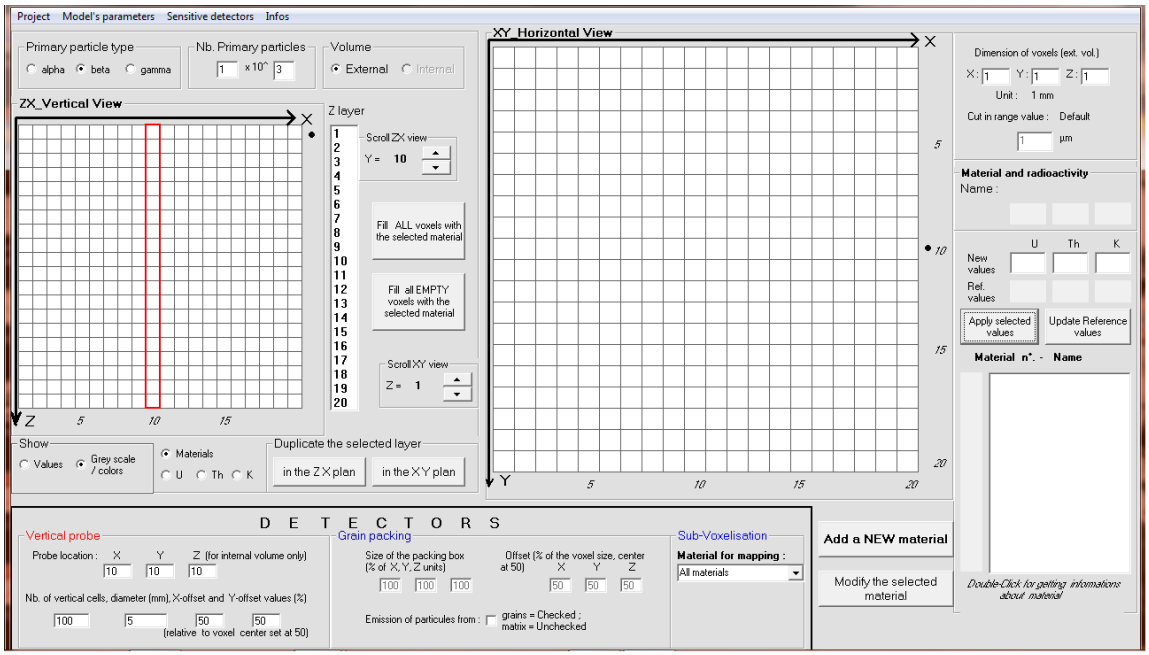
l'utilisateur peut aussi ajouter l'un des trois détecteurs suivants :

— Voxel sous-voxelisé : Permet de sous-voxeliser un voxel éditable comme le monde principal. Chaque sous-voxel enregistrera les doses radioactives.

— Sonde par paquet de grains : Permet de créer des grains mono-mineral dans le voxel souhaité. Chaque grain est considéré comme un detecteur indépendant enregistrant les doses radioactives.

— Sonde par paquet de grains successifs : Permet de créer des grains mono-mineral dans tous les voxels en X et Y et sur tout l'axe Z. Elle longe donc la sonde verticale.

FIGURE 1.1 – Ancienne interface DosiVox, écrite en VisualBasic



Interface déjà existante Chaque voxel doit être rempli par un matériel défini dans le fichier pilote avec des valeurs de radio-éléments donnés. Ces fichiers pilotes sont éditables "à la main" par un éditeur de texte mais cela est une tâche fastidieuse, surtout pour l'édition du monde voxelisé en

lui-même, car il faut que chaque voxel soit représenté. Norbert Mercier a alors développé une interface en Visual Basic (figure 1.1) afin d'éditer les fichiers pilotes. Cette interface, décrite dans le manuel de DosiVox, possède quelques limitations majeures :

- Elle décrit deux vues 2D du monde voxelisé, une en XY (vue du haut) et une en ZX (vue sur côté), l'utilisateur doit se forcer à une gymnastique de l'esprit pour se représenter l'objet en 3D.
- Les grilles de cette interface ont une taille fixe (20*20 voxels).
- L'interface étant codé en Visual Basic, elle ne fonctionne donc que sous Windows alors que DosiVox ne fonctionne lui que sur Linux.

Pour toutes ces limitations, un logiciel 3D plate-forme indépendant était donc souhaité. Le langage Java répond au besoin en ce qui concerne l'indépendance de la plate-forme car le langage imaginé par Sun a été créé dans ce but.

L'interface Java doit fournir les mêmes fonctionnalités que l'interface en Visual Basic (figure 1.1) avec les fonctionnalités supplémentaires suivantes :

- Visualisation 3D
- La possibilité d'éditer des mondes de toutes les tailles possibles.
- L'interface doit pouvoir être utilisable sous Linux, Mac et Windows.

A partir de ces objectifs, quelques choix doivent être faits. Pour commencer, les composants de l'interface seront programmés à l'aide de la librairie Swing. Elle a l'avantage d'être simple d'utilisation et de produire des interfaces qui auront le même aspect quelque soit le système hôte. En ce qui concerne le choix de la librairie 3D et n'en ayant jamais utilisé, une recherche approfondie des librairies disponibles en Java a été faite. Les solutions qui ont été considérées ont été Java3D, wrapper de OpenGL et DirectX, LWJGL (Light Weight Java Game Librairy) et JOGL, ces deux dernières sont des bindings Java d'OpenGL qui elle est écrite en C++. Dans le cadre du stage proposé et de mon éducation informatique, utilisé un wrapper alors que je n'avais jamais utilisé de libraire 3D, ne m'a pas semblé être une bonne approche. De plus, mes recherches sur différents forums (www.developpez.net, ou www.stackoverflow.com) m'ont

convaincu que Java 3D n'est pas un bon choix en termes de performances car beaucoup de voxels doivent être rendus ; Java3D a donc été abandonné.
Entre LWJGL et JOGL, le choix a été fait en fonction de leur but. LWJGL est plus utilisé pour la conception de moteur de jeu car elle gère aussi le son et les périphériques, alors que JOGL est un binding pur d'OpenGL, c'est-à-dire que chaque fonctions d'OpenGL est accessible. Les tutoriaux d'OpenGL seront donc très facilement adaptables, et c'est pourquoi JOGL a été choisi pour gérer la 3D de l'interface.

Chapitre 2

Développement de l'interface : DosiEdit

2.1 Conception de l'interface

2.1.1 Recherche de structure de données efficaces

La première étape avant même de s'occuper de la programmation OpenGL ou des composants Swing est de programmer les classes utiles à la collecte des informations du projet. Ces informations rassemblent les matériaux, les composants des matériaux, et le monde en lui-même. En ce qui concerne le type de données contenant tous ces voxels, quelques recherches ont été faites car pour de grands mondes, il fallait avoir une structure où le temps d'accès aux voxels était constant et le temps de recherche et de lecture proportionnel au nombre de voxels (O(n)). Trois choix ont étaient considérés :

— `LinkedList` :

> public class `LinkedList<E>`
> extends `AbstractSequentialList<E>`
> implements `List<E>`, `Deque<E>`, `Cloneable`, `Serializable`
>
> La lecture et le retrait sont en temps linéaire, mais la suppression

et l'ajout ne se font qu'en queue de liste. La LinkedList a donc vite était écartée.

— `ArrayList` :

public class `ArrayList<E>`
extends `AbstractList<E>`
implements `List<E>`, `RandomAccess`, `Cloneable`,
`Serializable`

la suppression est en temps linéaire, ainsi que la lecture mais l'ajout se fait en fin de liste. L'`ArrayList` a donc été également écartée.

— `HashMap` :

public class `HashMap<K,V>`
extends `AbstractMap<K,V>`
implements `Map<K,V>`, `Cloneable`, `Serializable`

La suppression est en temps constant, ainsi que l'ajout et la recherche. La `HashMap` est l'outil idéal, à condition que les clés pour stocker les voxels à l'intérieur correspondent aux indices d'où se situe le voxel recherché. Cela a paru satisfaisant, le monde voxelisé est donc finalement contenu dans un `HashMap` à 3 dimensions.

2.1.2 Description des classes de bases

On veut avoir la possibilité d'intégrer dans un voxel un sous-monde, autrement dit un voxel lui-même voxelisé. Il y a donc une classe `ExternalVoxelizedObject`, qui décrit le monde principal, et la classe `InternalVoxelizedObject`, qui décrit le monde sous-voxelisé, et ces deux classes partagent la classe abstraite `VoxelizedObject`, qui fournit les méthodes d'édition d'un monde voxelisé : différentes méthodes de remplissage (voxel par voxel, couche par couche ou tous les voxels d'un coup), les méthodes utiles au constructeur (celle qui initialise un monde vide ou à partir d'un scan 3D (voir section Scan 3D)), les méthodes de re-

dimensionnement et les accesseurs. `ExternalVoxelizedObject` contient dans un champ privé un `InternalVoxelizedObject` (null si aucun voxel est sous-voxelisé), il contient donc toutes les informations utiles au projet, mis à part les listes de composants et de matériaux.
La classe `Component` décrit un composant, elle est composée d'un champ formula, de classe `Formula`, qui permet de construire la formule chimique du composant, d'une densité et d'un nom. Ces composants composent les matériaux dans une classe `Material`, classe qui rassemble les valeurs de radioactivité, sa densité à sec, son nom et une liste de ses composants.
Avec tous ces objets, la classe `VoxelObject` peut alors être écrite, qui contiendra le `Material` avec lequel il sera rempli, des valeurs de radiations et des accesseurs. Plus tard, cette classe sera étendue à la classe `Rectangle2D.Float` pour l'affichage 2D (voir section Visualisation 2D et 3D : Graphics 2d et JOGL).

2.1.3 Développement des composants Swing interagissant avec les objets de base

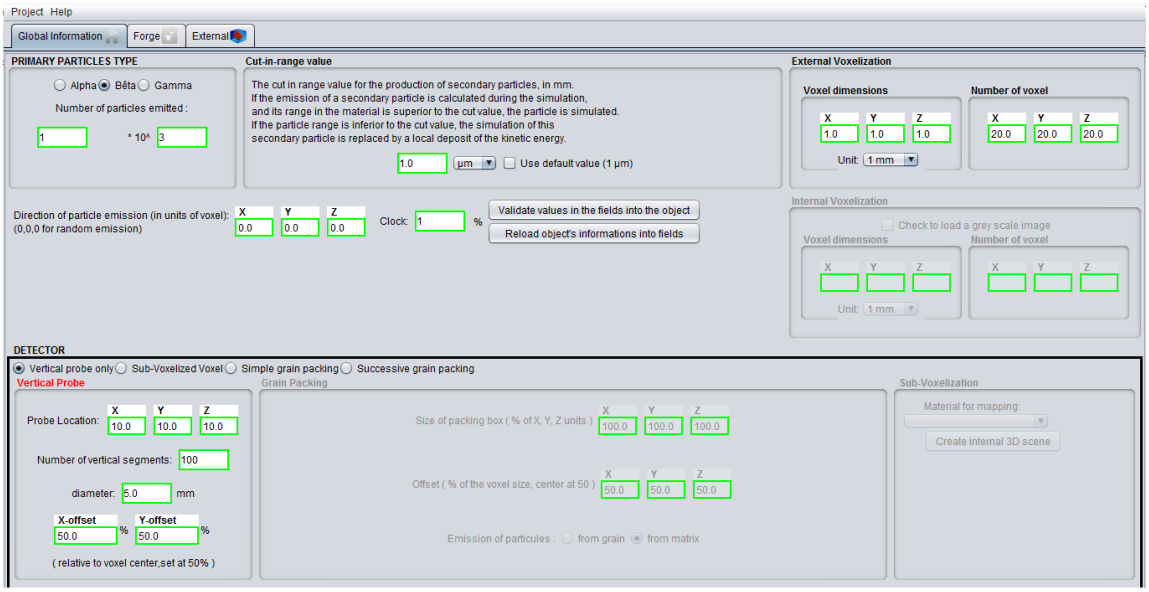
La première approche a été de construire l'interface sur le même patron que celui de l'interface en VisualBasic, en ordonnant les champs plus ergonomiquement. Au cours du développement et sous la direction de Mr Mercier, l'application as été repensée pour épurer la vue de l'utilisateur. L'idée était que certains paramètres sont habituellement changés une fois pendant l'édition, puis ignorer, car ces paramètres sont juste des valeurs à écrire pendant la génération des fichiers pilotes et qui ne sont pas utiles à l'édition du monde. Il as donc semblé judicieux de séparer ces paramètres dans une partie que l'utilisateur pourrait cacher, de même pour la fenêtre de création de matériaux et de composants.
Pour ceci, Java offre un object très utile : `JTabbedPane`. Ce composant Swing offre la possibilité de stocker d'autres composant Swing dans des onglets.

```
public class OngletManagement
extends javax.swing.JTabbedPane
```

implements `javax.swing.event.ChangeListener`,
`InternalVoxListener`, `ManuallyChangeListener`

La classe `OngletManagement`, dérivée de la classe `JTabbedPane`, réécrit la méthode `public void setSelectedIndex(int index)` pour tester avant chaque changement d'onglets si il y en a pas un qui est invalide (un matériel vide, une sonde hors du monde,...) et si c'est le cas, `OngletManagement` affiche l'onglet invalide. Pour les besoins de l'interface, trois onglets ou quatre (si il y a un monde sous-voxelisé) seront suffisants.

Classe MainInfoVoxel, l'onglet principal "Global Information"



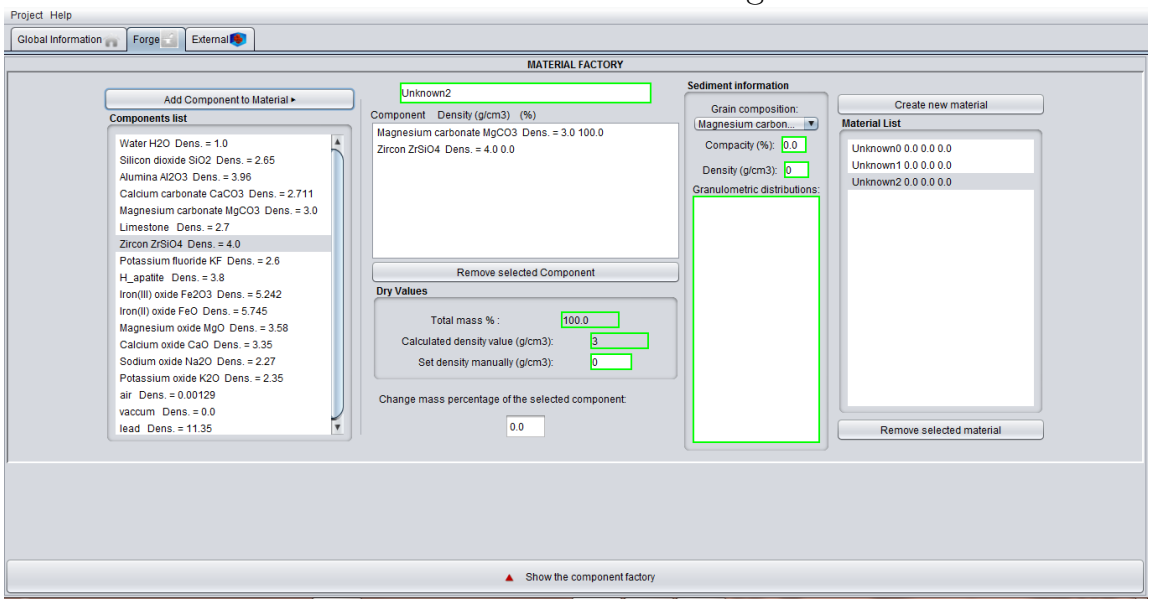
public class `MainInfoVoxel`
extends `javax.swing.JPanel`
implements `java.awt.event.KeyListener`, `java.awt.event.`
`ItemListener`, `javax.swing.event.ChangeListener`,
`MaterialAddedListener`, `java.awt.event.ActionListener`

Cet onglet rassemble toutes les informations que l'utilisateur a besoin de paramétrer une fois ou presque pendant toute la construction d'un projet. Elles ont été rassemblées ici pour ne pas surcharger sa vue.
Comportement : Dès qu'un champ est modifié, sa bordure devient rouge pour indiquer à l'utilisateur que le champ n'est pas à jour avec la valeur présent dans l'objet qu'elle décrit. Pour rentrer les valeurs dans l'objet `VoxelizedObject`, l'utilisateur peut presser la touche Entrée ou appuyer sur le bouton "Validate fields into object". Les bordures redeviennent alors vertes. Pour la validation des champs, les valeurs sont vérifiées et sont chargées si elles sont valides, sinon le champ est mis à jour avec les valeurs déjà présentes. Il est important de préciser qu'à la validation, tous les espaces du champs sont au préalable effacés pour pouvoir convertir l'objet `String` en `Integer` ou en `Float` sans `NumberFormatException`.
Pour créer un monde interne, l'utilisateur doit au préalable sélectionner la sonde sous-voxelisée (voir paragraphe sur l'explication des sondes dans l'introduction). Il doit ensuite entrer les valeurs de dimension de voxel et le nombre de voxels puis cliquer sur "Create internal world". Si la `JCheckBox` "check to load grey scale image" est cochée, il n'as alors pas besoin d'entrer de valeurs et l'action de cliquer sur le bouton de création du monde ouvrira alors une fenêtre lui permettant alors de chercher le dossier où les fichiers du scan sont stockés (voir section Scan 3D pour plus de précisions).
Programmation : Beaucoup de champs fonctionnent par trois, comme les champs de coordonnées de voxels, de dimensions ou de nombres. Il a donc était utile de programmer une classe de base pour ces champs, `TripleFormattedField`, qui partage les mêmes comportements, c'est-à-dire qu'ils acceptent dans leurs champs les mêmes types de valeurs (informations contenues dans un objet `MaskFormatter`) et ont les même tailles. La logique est la même pour les champs uniques, `FormattedLabelField`, qui eux, à l'instar du `TripleFormattedField` et de ces bordures à titre, offre la possibilité de décrire dans un `JLabel` à gauche le champ correspondant.

Classe ForgeWindow, l'onglet "Forge" de création des matériaux et des composants

Cet onglet est divisé en deux parties. La partie Sud est cachée et est affichable en appuyant sur le bouton "Show the component factory".

FIGURE 2.1 – NorthMaterialPanel : forger des materiaux

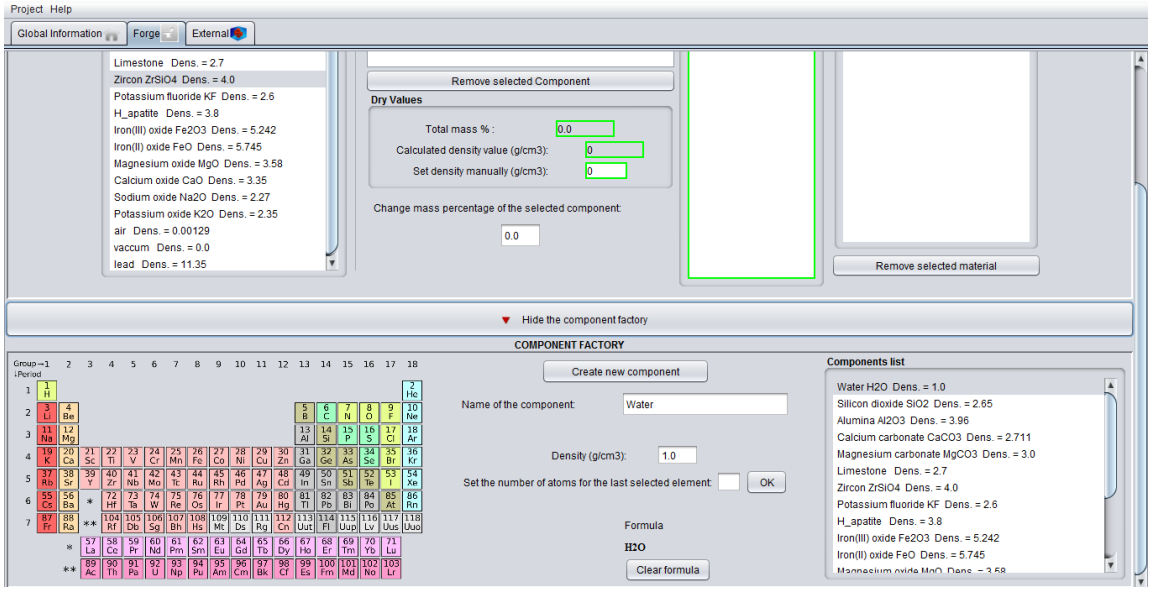


public class `NorthMaterialPanel`
extends `javax.swing.JPanel`
implements `javax.swing.event.ListSelectionListener`, `java.awt.event.ActionListener`, `ComponentAddedListener`, `java.awt.event.KeyListener`, `java.awt.event.ItemListene`

Cet partie de l'onglet, contenue dans la classe `NorthMaterialPanel`, est un utilitaire permettant la création et l'édition de matériaux.
Comportement : L'utilisateur crée un matériau par le bouton "Create new material" et s'affiche dans la liste avec un nom par défaut (UnknownX, où X est son indice dans la liste des matériaux). Il peut alors

y ajouter des composants parmi la liste disponible à sa gauche par le bouton "Add component to material". Par les champs de saisie, il peut changer la densité du matériau sec, et le pourcentage du composant sélectionné à l'intérieur. Le total des pourcentages des différents composants doit être égale à 100 pour avoir un matériau valide. Il peut aussi entrer des valeurs granulométriques dans le cas des sondes par paquet de grains, en entrant deux valeurs par ligne dans le `JTextArea` : la première est le diamètre des grains, la deuxième son taux de pourcentages dans le matériau, une ligne représentant une des distributions des grains.
Programmation : Les listes des composants et des matériaux sont affichés par des `JList<T>`, où T est soit la classe `Material`, soit la classe `Component`, mais aussi la classe `Couple<Component,Float>` pour la liste des composants dans le matériau avec son taux de pourcentage correspondant (la classe `Couple<T,P>` est une classe générique associant deux objets que l'on veut lié).

FIGURE 2.2 – SouthComponentPanel : forger des composants

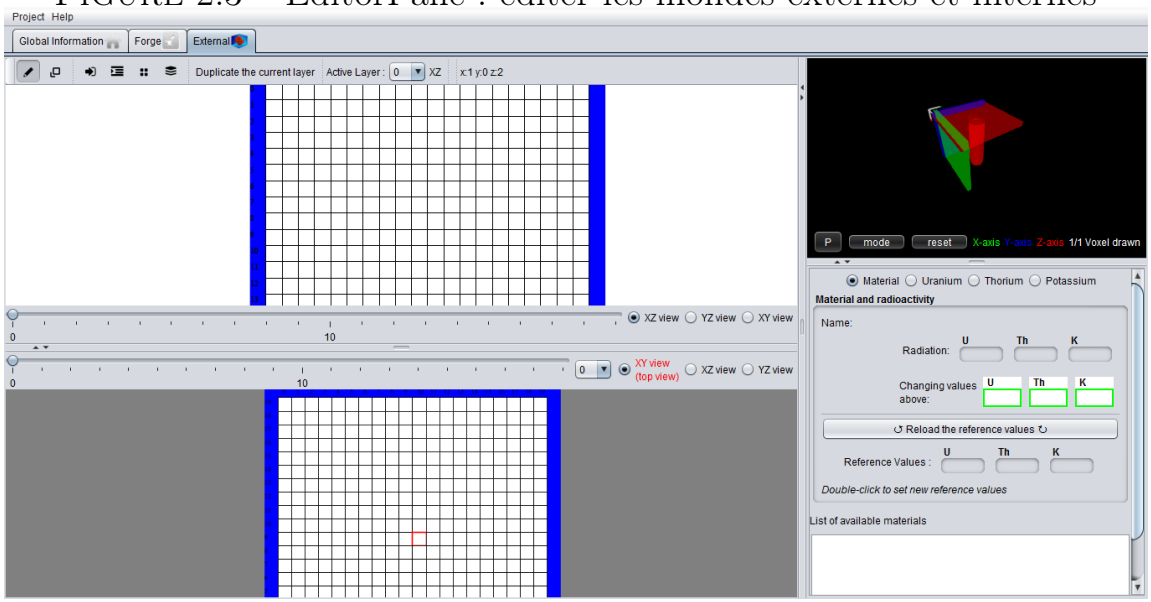


public class `SouthComponentPanel`
extends `javax.swing.JPanel`
implements `java.awt.event.ActionListener`, `javax.swing.event.ListSelectionListener`

Comportement : Après action sur le bouton "Show component factory", la forge des composants se dévoile. 18 composants sont déjà créés, ils sont notés comme basiques. Ils ne peuvent pas être modifiés. Pour créer un nouveau composant qui lui sera modifiable, l'utilisateur clique sur "Create new component" et un composant se créé de nom "UnknownX", où X s'incrémente. Son nom est toujours précédé d'un underscore "_" pour signifier que c'est un composant créé par l'utilisateur. Pour créer la formule chimique du composant, l'utilisateur clique simplement sur une case du tableau périodique pour ajouter l'élément en queue de formule. Si l'élément est le même que le dernier courant, il s'incrémente. L'utilisateur peut aussi indiquer manuellement le nombre de dernier élément de la formule.
Programmation : La liste des composants est créée "en dur" au début du programme et une `JList` l'affiche sur la droite. Pour l'image cliquable, la solution la plus simple a été de créer une classe `PeriodicTablePanel` qui étend la classe `JPanel` avec l'image du tableau périodique en fond. Une classe interne `LittleButton` permet de créer des boutons transparent de taille uniforme et quand la souris passe sur le bouton, après une seconde s'affiche le nom scientifique et le numéro atomique de l'élément associé au bouton. Tous ces boutons créés sont ensuite disposés par un `GridLayout`, les zones vides remplies par des composants vides créés par la fonction statique `Box.createRigidArea(Dimension m)`.

Classe EditorPane, l'onglet "Editor" d'édition des mondes externes et internes

FIGURE 2.3 – EditorPane : éditer les mondes externes et internes



public class `EditorPane`
extends `javax.swing.JSplitPane`
implements `javax.swing.event.ListSelectionListener`, `java.awt.event.FocusListener`, `java.awt.event.MouseListener`, `java.awt.event.ActionListener`, `java.awt.event.KeyListener`, `javax.swing.event.ChangeListener`, `java.awt.event.ItemListener`, `java.awt.event.MouseMotionListener`

La classe `EditorPane` est utilisée pour afficher en 2D et en 3D un `VoxelizedObject`, ce qui lui permet d'être capable d'afficher un objet externe commme interne.

Comportement : Cet onglet se divise en quatre parties. La partie Ouest rassemble deux grilles d'édition, et la partie Est rassemble la vue 3D et la liste des matériaux avec leurs informations de radiations. L'utilisateur

peut se déplacer dans l'objet grâce aux ascenseurs horizontaux, représentant l'indice de la couche courante suivant le mode sélectionné (en mode XY, l'ascenseur se déplace sur l'axe Z, en XZ sur l'axe Y et en YZ sur l'axe X). Il peut alors cliquer du bouton gauche dans un carré pour remplir le voxel correspondant avec la couleur du matériau, maintenir le bouton droit enfoncé pour déplacer la grille ou zoomer grâce à la molette. Il est fourni une barre d'outils pour l'édition plus aisée du monde :

- Mode crayon : le clic gauche enfoncé remplis les voxels sur lesquels la souris passe.
- Mode rectangle : le clic gauche enfoncé et le mouvement de la souris crée un rectangle qui part de l'endroit où le clic a été fait, jusqu'à l'endroit courant de la souris. A la relâche du clic gauche, tous les voxels intersectant avec l'aire du rectangle sont remplis.
- Remplir voxel vide de la couche courante : Rempli tous les voxels remplis par aucun matériau sur la couche courante avec le matériau sélectionné.
- Remplir voxel de la couche courante : Rempli tous les voxels sans exception sur la couche courante avec le matériau sélectionné.
- Remplir voxel vide de l'objet : Rempli tous les voxels de l'objet remplis par aucun matériau avec le matériau sélectionné.
- Remplir voxel de l'objet : Rempli tous les voxels de l'objet sans exception avec le matériau sélectionné.

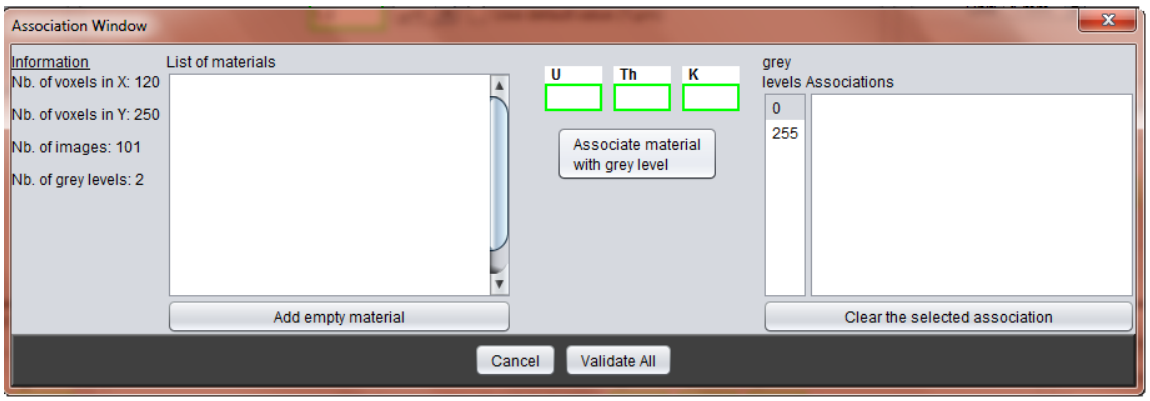
Il est important de noter que seule la grille du haut est éditable et est connectée avec la vue 3D, c'est-à-dire que la l'ascenseur de cette couche contrôle aussi les couches représentées dans la vue 3D. Au contraire, la vue du dessous n'est pas éditable, elle est juste un moyen pour l'utilisateur de se déplacer dans l'objet sans tout bouger, de sorte qu'il ait à faire le moins d'aller-retour possibles entre les couches.
En ce qui concerne la vue 3D, l'utilisateur peut zoomer sur l'objet avec la molette, le translater en maintenant le clic droit enfoncé ou le tourner (rotation) en maintenant le clic gauche. Il peut aussi changer la couleur de la sonde grâce au bouton "P", changer le mode d'affichage (cube rempli ou arêtes dessinées seulement) ou réinitialiser la caméra à son état d'origine.

Programmation : Pour le détail des classes utiles à l'affichage 2D et 3D, voir chapitre 3 "Visualisation 2D et 3D : Graphics2d et JOGL".

2.2 Scan 3D

Un scan 3D est représenté par un dossier contenant des fichiers textes, chaque fichier représentant une tranche de l'objet vue du haut (en XY) par une matrice contenant des niveaux de gris. Chaque niveaux de gris représente chaque matériau de l'objet. Cette série d'images-texte peut être créée à partir du logiciel ImageJ (voir Annexes), ou à partir d'un objet réel passé dans un scanner 3D.

Comportement : Pour charger un scan 3D, l'utilisateur se place dans l'onglet "Global informations" et sélectionne la sonde sous-voxelisée (Sub-Voxelized Probe). Dans la zone Internal Voxelisation, il doit cocher "check to load a grey scale images" et cliquer sur le bouton "Create internal world". Une fenêtre apparait alors lui permettant de sélectionner le dossier contenant les fichiers textes. L'interface lit alors tous les fichiers pour enregistrer les différents niveaux de gris et une nouvelle fenêtre apparait :



L'utilisateur doit alors associer les niveaux de gris avec des matériaux, pour que l'interface puisse créer le monde interne en remplacant les valeurs de niveaux de gris par les matériaux associés respectivement. Il peut spécifier des valeurs de radiations, créer de nouveaux matériaux

au cas où il lui en manque et cliquer sur "validate all" une fois que tous les niveaux de gris ont été associés avec un matériau, sans exception. Cette fonctionnalité de charger un scan 3D était déjà présente dans l'interface en VisuelBasic (voir Figure 1.1) mais quelques améliorations ont été réalisées :

- Si il manque un matériau durant la phase d'associations, l'utilisateur n'a pas besoin d'annuler la procédure pour créer les matériaux dont il a besoin, ce qui était le cas dans l'ancienne interface.
- Des utilitaires Java permettent une meilleure approche pour l'utilisateur, comme la barre de progression lui permettant de savoir où en est le chargement de son objet, qui peut être très long à construire.
- Après la phase d'associations, une fenêtre de dialogue lui offre la possibilité de charger le monde provenant du scan non pas dans le monde interne, mais dans le monde externe en écrasant le monde déjà existant. Cela lui donne la possibilité de charger une image 3D mais de pouvoir utiliser une autre sonde après chargement.

Programmation : Un scan 3D se déroule en trois phases :

1. *Phase de collecte des niveaux de gris :* Un objet `Loader3DScan` est créé.

   public class `Loader3DScan`
   extends `java.lang.Object`

   Grâce à un `BufferedReader`, cette classe lit chaque fichiers du dossier et ajoute un niveau de gris à une liste triée (`SortedList`) si le niveau de gris n'est pas déjà dans la liste.

2. *Phase d'associations :* Le loader est maintenant envoyé à un objet `AssociationFrame`.

   public class `AssociationFrame`
   extends `javax.swing.JDialog`
   implements `java.awt.event.ActionListener`

La `JList` affiche une liste de `Couple<Material,Float>` qui stocke les associations matériau/niveau de gris. Une fois que l'utilisateur a réalisé ses associations, la liste est envoyée au loader et l'objet `AssociationFrame` est détruit. Le loader est alors mis dans un état dit valide et la phase suivante peut alors être réalisée.

3. *Phase de création du monde :* Grâce à une simple `JDialog` modale (qui impose le focus sur elle), l'utilisateur choisi si il veut créer le monde en interne ou en externe. Dans les deux cas, la même méthode est appelée car elle est contenue dans la classe abstraite `VoxelizedObject`.

```
public void ScanToVoxel(Loader3DScan loader,
                        javax.swing.JProgressBar bar)
                  throws java.lang.NumberFormatException,
                         java.io.IOException
```

Si le monde est demandé à être créé en interne, cette méthode est appelée par le constructeur `InternalVoxelizedObject`. La méthode prend comme paramètre le loader, elle vérifie si il est dans un état valide (c'est-à-dire qu'il est passé par une `AssociationFrame`) et relie les fichiers du chemin stocké dans le loader, en remplacant cette fois chaque niveau de gris rencontré par les matériau avec lequel il est associé. Si le monde créé est interne, un nouvel onglet apparait nommé "Internal" et est identique à l'onglet "External" à l'exception près qu'il n'affiche pas de sonde verticale.

Chapitre 3

Visualisation 2D et 3D : Graphics2d et JOGL

3.1 Grid2D et ViewGrid2D :

Classe Grid2D : C'est la classe qui affiche les voxels de l'objet, de façon éditable :

public class `Grid2D` extends `JPanel` implements `ChangeListener`, `MouseListener`, `MouseMotionListener`, `ActionListener`

Cette classe affiche les VoxelObject étendu de `Rectangle2D.Float` :

public class `VoxelObject` extends `Rectangle2D.Float` implements `Serializable`

La classe `Rectangle2D.Float` représente un rectangle qui sera dessiné dans le contexte graphique courant, représenté par une instance de `Graphics2D`. Ce rectangle est dessiné en fonction des coordonnées en X et Y de son coin haut-gauche, de sa largeur et sa hauteur.
Des fonctions ont été ajoutées à la classe abstraite `VoxelizedObject` pour la gestion de ces rectangles :

— public void `updateAccordingView`(`Layer mode`, int `activeLayer`):

Cette méthode est appelée quand l'utilisateur change de mode de vue (XY,XZ ou YZ). Elle modifie les coordonnées des `VoxelObject` pour les réafficher correctement dans le contexte graphique. Exemple : Dans une vue en XY (vue du haut), la hauteur du rectangle est donc égale à la taille du voxel en Y. Mais si l'utilisateur change la vue pour regarder la tranche de l'objet en YZ (vue de côté), la hauteur du rectangle doit être égale à la taille du voxel en Z, et ce pour chaque voxel de l'objet.

— public `ArrayList<Integer> checkAndFill`(`Rectangle2D.Float selectRect`,`Layer mode`,int `activeLayer`,`Material matToFillWith`):

Cette méthode est appelée quand l'utilisateur clique dans un Voxel pour le remplir, pour un remplissage avec un rectangle de sélection. Elle vérifie dans tous les voxels de la couche courante si le rectangle passé en argument intersecte avec eux, et si oui, remplit le voxel avec le matériau `matToFillWith`. Cette méthode existe aussi en mode crayon, en passant les coordonnées de la position de la souris à la place de `selectRect`. (Pour plus de précisions à propos des modes, voir Chapitre "Developpement de l'interface : DosiEdit", section "Developpement des composants Swing", à la description de la classe "EditorPane"). Elle renvoie une liste avec les coordonnées des VoxelObject modifiés pour indiquer à la scène 3D quels voxels doivent être mis à jour.

— Des méthodes de redimensionnement : Méthodes appelées quand l'utilisateur change le nombre de voxel en X, Y ou Z, ou modifie leur taille.

Classe ViewGrid2D : C'est la classe qui affiche les voxels, de façon non-éditable :

public class `ViewGrid2D` extends `JPanel` implements `ChangeListener`

Cette classe, a contrario de `Grid2D`, n'affiche pas les `VoxelObjet` eux-mêmes. Elle crée à la place un tableau à deux dimensions de `MinimalVoxel` (classe interne), qui sont des copies des `VoxelObject` de l'objet avec seulement les informations utiles à l'affichage.
Elle contient l'équivalent des méthodes de `Grid2D` pour l'affichage et la mise à jour :

— public void `updateMode`(`Layer mode`,int `newLayer`):

Cette méthode est l'équivalent de `updateAccordingView` de `Grid2D`. Elle modifie le tableau de `MinimalVoxel` quand le mode de vue est changé par l'utilisateur (XY,XZ ou YZ).

— public void `updateMaterial`():

Cette méthode est appelée quand l'utilisateur a modifié des voxels dans la couche courante. Dans ce cas-là, les rectangles affichés par `ViewGrid2D` doivent être à jour avec ceux affichés par `Grid2D`, si le mode d'affichage est le même et si la couche courante est la même affichée.

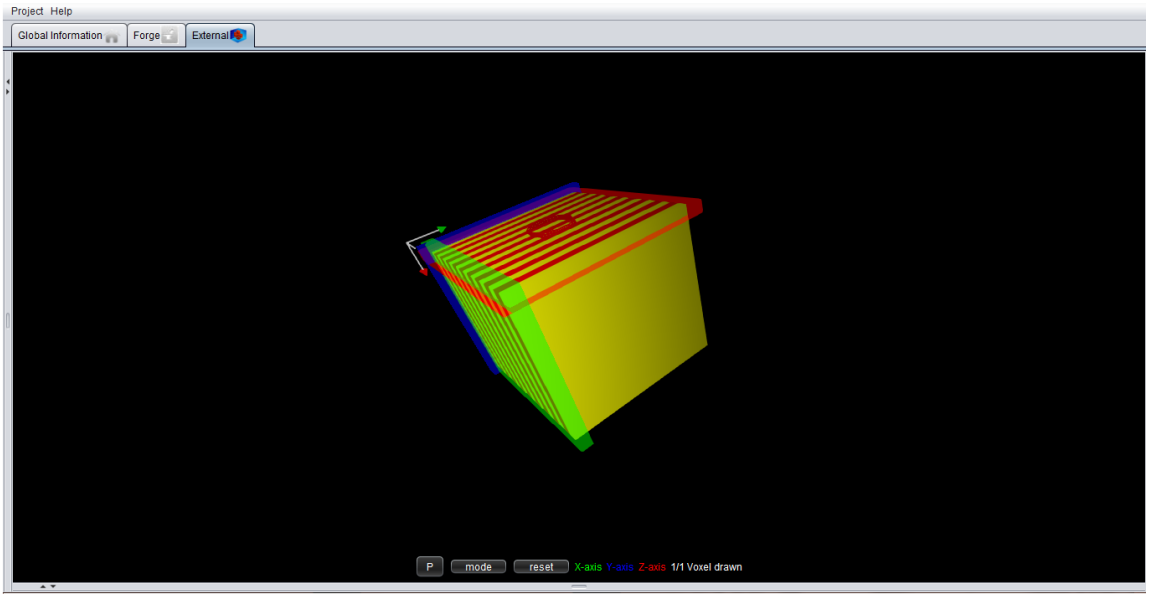
Classe ZoomAndPanListener : Cette classe code la caméra de la vue 2D, c'est-à-dire qu'elle manipule les matrices de translation, de rotation et d'échelle.

public class `ZoomAndPanListener` implements `MouseListener`, `MouseMotionListener`, `MouseWheelListener`

Pour mettre à jour ces matrices dans le contexte graphique, `Graphics2d` propose une fonction `setTransform` qui reçoit un objet `AffineTransform`, objet qui stocke les matrices. A chaque mouvement de la souris, les méthodes venant des interfaces `MouseListener`, `MoustMotionListener` et `MouseWheelListener` sont appelées et modifie les matrices de transformations, et au début de chaque appel de la méthode `draw()` des classes

d'affichage, les matrices modifiées sont envoyées au contexte graphique. *Cette méthode n'as pas été écrite par mes soins, elle a été trouvée sur https ://github.com/2m/toh/blob/master/src/ZoomAndPanListener.java pendant mes recherches d'apprentissage de manipulation de matrices.*

3.2 Scene :



public class `Scene` extends `GLCanvas` implements `GLEventListener`

Implémentations et problèmes rencontrés :

N'ayant jamais utilisé de fonctions 3D précédemment, une recherche approfondie a été faite étant donné que l'application doit potentiellement afficher des milliers de voxels en même temps. La première méthode est l'affichage immédiat, elle se présente sous cette forme :

```
gl.glBegin(GL.GL_QUADS)
```

```
  gl.glVertex3f(xf, yf, zf)
  gl.glVertex3f(xf, yf, zf)
  ...
gl.glEnd();
```

QUADS représente le type de polygones à dessiner que représente les sommets qui suivent. Tous les appels à `glVertex3f` définissent un sommet du polygone. Le 3 défini le nombre de paramètres de la fonction, et f le type de données envoyées (f pour float, d pour double, i pour integer). L'appel à glEnd() défini la fin de la définition des sommets de polygone.

Inconvénients : Ces définitions de polygone sont pratiquées dans la méthode `display(GLAutoDrawable drawable)` appelée régulièrement. L'inconvénient est que, quand il y a beaucoup de polygônes a afficher comme dans le cas présent, la carte graphique est surchargée par les appels de fonctions graphiques et le rendu devient soit terriblement lent, soit plante purement et simplement.
Un apprentissage complet a donc été fait en ce qui concerne les Vertex Buffer Object (VBO).

Vertex Buffer Object - VBO :

Les Vertex Buffer Object sont le moyen d'envoyer à la carte graphique les sommets à afficher et le mode d'affichage de ces sommets en gardant un identifiant du buffer côté "client", le carte graphique représentant le "serveur". Dans la boucle d'affichage, il suffit donc de demander à la carte graphique d'afficher le buffer ayant l'identifiant qu'on lui demande d'afficher. Malgré cela, l'application est très lente quant elle doit afficher énormément de voxels, ce qui n'arrive pas rarement. La solution proposée est la suivante : Une constante `MAXIMUM_RENDER_VOXEL` égale à 40*40*40 représente le nombre maximum de voxels que la scène peut afficher. Si le nombre de voxels dépasse ce nombre, la scène ne récupère les informations que d'un voxel sur deux. Si le nombre de voxel rendu est encore plus grand que la constante, elle ne récupère les informations que d'un voxel sur trois. Et ainsi jusqu'à ce que le nombre de voxel a afficher soit plus

petit que la constante.

Implémentation : La classe `Scene` affiche un `VoxelizedObject`, il peut donc afficher un objet externe comme interne. Quand il s'agit d'un objet externe, il affiche la sonde verticale et les couches courantes qu'affiche la vue éditable (`Grid2D`). La classe abstraite `GLcanvas` impose les méthodes suivantes :

- `init(GLAutoDrawable)` : Initialise le contexte graphique 3D. Cette méthode est appelée une première fois, et juste une fois, avant toute autre appel de fonction graphique.
- `display(GLAutoDrawable)` : Méthode appelée régulièrement. Dans le cas présent, elle est appelée au mieux à 60 frames par secondes (FPS).
- `reshape(GLAutoDrawable)` : Méthode appelée quand la fenêtre d'affichage du contexte graphique 3D est redimensionnée.
- `dispose(GLAutoDrawable)` : Dernière méthode appelée avant la destruction de l'objet. C'est la dernière chance d'effectuer des actions avant la fin de l'objet.

La classe `Scene` contient une méthode `update()` qui permet de mettre à jour le Vertex Buffer Object avec les informations contenues dans le `VoxelizedObject`. Les fonctions `updateXYLayer()`,`updateXZLayer()` et `updateYZLayer()`, quant à elle, modifie la position des rectangles bleue, rouge et vert représentant les différentes couches courantes.

Camera et MouseCameraManager :

L'implémentation de la caméra a été codé à l'aide du tutoriel présent à l'adresse http ://plegat.developpez.com/tutoriels/jogl/pge-camera/.

public class `Camera`
extends `java.lang.Object`

Cette classe contient les matrices de rotation, de translation et d'échelle. Ces matrices sont modifiées quand l'utilisateur utilise la molette de la

souris (échelle), ou bouge la souris avec clic gauche (rotation) ou droit enfoncé (translation), grâce à la classe suivante.

public class `MouseCameraManager`
extends `java.lang.Object`
implements `java.awt.event.MouseListener`, `java.awt.event.MouseMotionListener`, `java.awt.event.MouseWheelListener`

Cette classe, contenant une instance de classe `Camera`, modifie au changement d'état de la souris les matrices de cette classe. La scène met alors à jour, dans la méthode `display(GLAutoDrawable)`, ses matrices courantes en récupérant celles contenues dans la classe `Camera` :

```
gl.glLoadMatrixf(this.camera.getCameraMatrix());
```

Chapitre 4

Améliorations possibles

L'interface, bien qu'acceptable, présente néanmoins quelques limites et certaines améliorations peuvent être apportées, en termes de nouvelles fonctionnalités et de techniques de programmation :

Textures et shaders : Dans le menu contextuel offrant à l'utilisateur de changer de matériau, il y a été ajouté la possibilité de poser une texture plutôt que des voxels. Seulement, le temps n'as pas suffi pour permettre à la scène d'afficher ces textures. Un apprentissage approfondi sur l'utilisation des shaders servirait donc à non seulement pouvoir afficher des textures, mais aussi à améliorer le rendu en utilisant des effets de lumières et d'ombres.

Implémentation des classes Swing : Les composants Swing ont été programmés par Sun pour pouvoir être utilisés facilement avec le pattern `Observer`. Un réusinage de code serait donc à envisager, en implémentant l'interface `Observable` dans l'objet `VoxelizedObject`, à la place d'envoyer une référence à tous les composants Swing qui l'éditent. Cela produirait un code beaucoup plus maintenable que celui-ci.

Liaison avec DosiVox : DosiEdit édite des fichiers pilotes, et l'utilisateur le passe à DosiVox par un interpréteur de commandes (expliqué dans le manuel). Il serait aisé d'avoir, dans le menu de l'interface, la

possibilité d'envoyer directement les fichiers pilotes à DosiVox, sans que l'utilisateur n'ait à passer par un interpréteur de commandes. DosiVox ne fonctionnant que sous Linux, l'interface avant devra tester si elle peut le faire, et si c'est le cas, elle lancera DosiVox par appel système, avec les lignes de commandes correspondantes au choix de l'utilisateur.

Chapitre 5

Conclusion

L'interface répond aux besoins et les améliorations demandées le sont. L'affichage OpenGL en 3 dimensions de l'objet était pour moi le défi du stage et le résultat, bien qu'il soit sujet à diverses améliorations en termes de qualité graphique, est très satisfaisant.
A propos des composants Swing, leur utilisation n'a pas été codée d'une manière très maintenable car elle ne profite pas de toutes les fonctionnalités que propose le patron de conception Observeur/Observable. Pour un prochain projet de conception d'interface, les classes conçues par Sun seront utilisées avec tout leur potentiel.
L'application de l'informatique à la datation d'objets est un sujet qui s'est révélé être passionnant et j'espère que ma carrière m'orientera vers d'autres projets de ce genre, de l'anthropologie à toutes autres formes de science.
Je compte être à la disposition des chercheurs pour toute demande touchant à l'interface (idées d'amélioration à ajouter, reports de bugs, etc...) et ainsi maintenir l'application dans le futur pour les besoins des chercheurs utilisant le logiciel DosiVox.

Annexe A

Liens utiles

Adresses du responsable de stage :

norbert.mercier@u-bordeaux-montaigne.fr

Site de téléchargement de DosiVox et Geant-4 sur :

http ://www.iramat-crp2a.cnrs.fr/spip/spip.php ?article144

Site fréquentés tout au long du stage :

- StackOverFlow : http ://www.stackoverflow.com
- OpenClassRoom : http ://www.openclassroom.com
- Developpez.net : http ://www.developpez.net
- Documentation d'Oracle sur le langage Java : http ://docs.oracle.com/javase/7/docs/api/
- Documentation et forum sur JOGL : http ://jogamp.org/jogl/www/
- Site de téléchargements d'ImageJ : http ://imagej.nih.gov/ij/