Implémentation d'un algorithme de généralisation dans GeOxygene

Benjamin Bout

OpenStreetMap

Master Technologies Biomédicales

Remerciements

Je tiens tout d'abord à remercier Monsieur Jean-Marie Favreau, enseignant-chercheur au LIMOS, pour son accompagnement durant cette période de stage au LIMOS, son aide précieuse et ses conseils qui ont su m'éclairer. Je tiens également à remercier Monsieur Yan Gérard pour les conseils et l'expertise dont il a su me faire part dans le domaine mathématique. Par ailleurs, je souhaite également remercier Guillaume Touya, membre du laboratoire CoGit (Cartographie et Géomatique) de l'IGN, qui a su m'éclairer et m'aider à chaque fois que j'avais une question sur le fonctionnement de GeOxygène. Pour finir, je tiens à remercier Pierre Charles, Nicolas Desforges-Desamin et Jérôme Rolland, également en stage sur le même projet, avec qui j'ai pu échanger et obtenir de l'aide sur certains points de développement et de compréhension des outils de développement.

Sommaire

Introduction

Dans le cadre de ma première année de Master Technologies Biomédicales, à la Faculté de Médecine de Clermont-Ferrand, j'ai effectué un stage de huit semaines au Laboratoire de'Informatique, de Modélisation et d'Optimisation des Systèmes (**LIMOS**), en tant que développeur.

Le **LIMOS** est une Unité Mixte de Recherche en informatique, et plus généralement en Sciences et Technologies de l'Information et de la Communication. Il est principalement rattachée à l'Institut des Sciences de l'Information et de leurs Interactions du CNRS, et de façon secondaire à l'Institut des Sciences de l'Ingénierie et des Systèmes. Le positionnement scientifique du **LIMOS** est centré autour de l'Informatique, la Modélisation et l'Optimisation des Systèmes Organisationnels et Vivants. Il a pour tutelles académiques l'Université Clermont Auvergne, l'Ecole des Mines de Saint-Etienne et l'Institut Français de Mécanique Avancée. Les principaux thèmes de recherche développés au sein du laboratoire sont :

- L'optimisation combinatoire et continue
- La recherche opérationnelle, les systèmes de production, la logistique
- L'algorithmique des graphes et des treillis
- Images et apprentissage
- La modélisation et simulation
- Les grandes masses de données, la fouille de données, l'interopérabilité des systèmes d'information
- Les réseaux de capteurs, la confiance numérique

L'un des projet du **LIMOS** concerne l'amélioration d'une application du laboratoire **CoGit** (Cartographie et Géomatique) de l'Institut National Géographique (IGN), **GeOxygène**. **GeOxygène** est une application Open Source développée en Java qui vise à fournir un cadre de développement pour la conception et le déploiement d'applications s'appuyant sur des données géographiques.

Durant ce stage, nous étions quatre stagiaires à travailler sur ce logiciel et notre contribution concerne l'ajout de différents éléments permettant de générer des cartes géographiques à destination des personnes ayant des déficiences visuelles. Nous avions pour tâches d'implémenter des fonctionnalités qui permettent de générer des textures sur certains bâtiments, de générer les noms des rues en braille, d'améliorer le logiciel permettant la génération d'un modèle 3D d'une carte en niveau de gris et d'implémenter diverses fonctionnalités de généralisation de bâtiments.

Durant ce stage, mon objectif était l'implémentation de fonctionnalités présentées et expliquées dans un article scientifique concernant la généralisation de bâtiments, et plus particulièrement la schématisation des bâtiments tout en conservant leur aire initiale.

Mon rôle était donc de lire l'article scientifique, de le comprendre, de l'implémenter en Java dans l'application tout en l'adaptant aux besoins spécifiques de l'application et en proposant diverses solutions complémentaires.

Présentation de l'environnement

Mon projet s'inscrivant dans la plateforme **GeOxygène**, développée en **Java** par le **CoGit**, il semble interessant de présenter tout d'abord ce qu'est cette plateforme et ce à quoi elle tend.

Figure 1 : Logo du Cogit et de GeOxygene

GeOxygene est un environnement permettant different traitements et utilisations de données géographiques. Il permet d'ouvrir différents types de cartes, 2D ou 3D, de les modifier, d'ajouter des composants, d'effectuer des traitements, d'appliquer des filtres, de faire de la généralisation, etc. Pour notre projet, nous avons uniquement utilisé des cartes de type OSM, **Open Street Map**. OSM est un projet francophone qui a pour but de créer une carte du monde, renseignée librement, accessible à n'importe qui et libre de modifications, d'ajout. Ce projet étant sous licence libre, n'importe qui peut utiliser des cartes OSM pour faire ce qu'il en souhaite, c'est la raison pour laquelle nous avons utiliser ce type de données cartographiques.

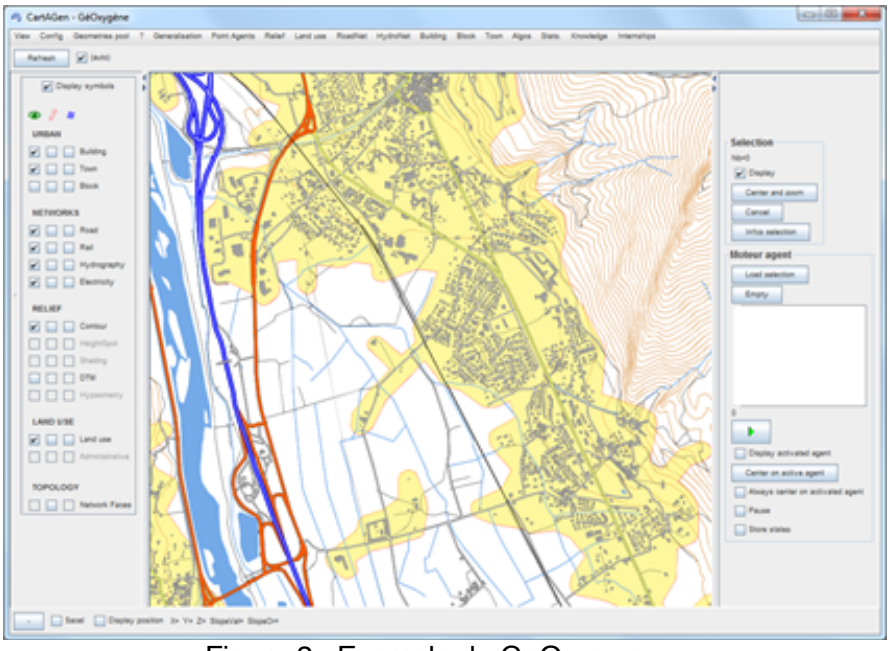


Figure 2 : Exemple de GeOxygene

Figure 3 : Logo d'OpenStreetMap

Présentation du Projet *Cartes tactiles*

Le projet **Cartes tactiles** est un projet ayant pour but de permettre d'**imprimer des cartes géographiques en 3D à partir de données cartographiques 2D** à destination des personnes ayant des déficiences visuelles (non-voyants et mal-voyants). Pour réaliser ces cartes tactiles, il est nécessaire d'effectuer tout un processus de traitement des données cartographiques :

- Enregistrement un fichier OSM (sélectionner une zone géographique à enregistrer)
- Import du fichier OSM dans GeOxygene grâce au plug-in OSM
- Application de divers traitements de généralisation
- Application de feuilles de styles (SLD) de mise en niveaux de gris
- Application des SLD pour appliquer des textures à certains éléments de la carte
- Simplification des noms des rues
- Application de la police braille via un SLD
- Génération de la légende de la carte dans un fichier HTML
- Export en image de la carte générée
- Création d'un maillage 3D à partir de l'image dans le logiciel 3DMapGen
- Impression de la carte en 3D à partir du maillage

La tâche que j'ai eu à réaliser concerne l'**application de divers traitements de généralisation**. En me basant sur un article scientifique traitant de la schématisation des bâtiments, j'ai eu à implémenter diverses fonctionnalités de généralisation, afin de rendre les cartes générées plus simples à lire. Cette tâche s'inscrit dans ce projet de **Cartes tactiles** mais les fonctionnalités que j'ai eu à implémenter peuvent être utilisées en dehors du processus de création de cartes tactiles.

Développement

1. Document support

L'article scientifique sur lequel je me suis basé pour réaliser mon projet s'intitule Area-Preserving Subdivision Schematization et a été rédigé par Wouter Meulemans, André van Renssen et Bettina Speckmann, du département de mathématiques et d'informatique de l'Université de Technologie d'Eindhoven aux Pays-Bas.

Cet article décrit des algorithmes utilisés pour effectuer une **schématisation de polygones en conservant l'aire originale de ceux-ci**. Il est dans un premier temps décrit comment transformer un polygone quelconque en un polygone rectilinéaire de même aire. Il est ensuite présenté différentes méthodes permettant la simplification de polygones tout en conservant leur aire. Afin de simplifier les polygones, il est dans un premier temps nécessaire d'effectuer leur orthogonalisation, afin d'obtenir un polygone rectilinéaire.

L'orthogonalisation est réalisée par rapport au repère général, ainsi, les polygones en résultant seront composés uniquement de segments verticaux et horizontaux.

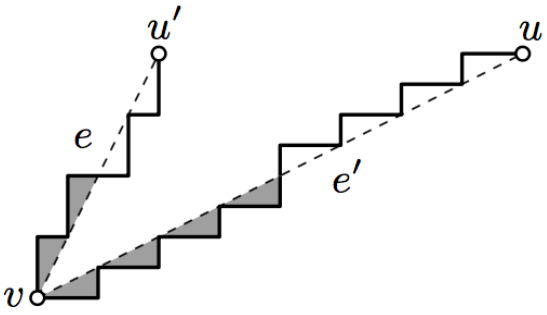


Figure 4 : Méthode d'orthogonalisation

Ici, les segments [vu'] et [vu] sont découpés en plusieurs segments orthogonaux.

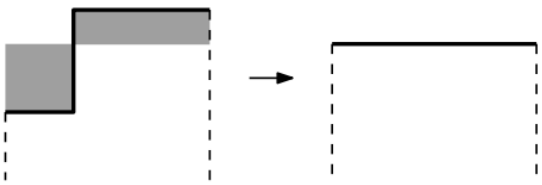


Figure 5 : Méthode de simplification

Une fois l'orthogonalisation effectuée, on peut passer sur la simplification. Celle-ci est effectuée simplement en effectuant une moyenne des différents segments du polygone. Ici par exemple, on va équilibrer les deux rectangles gris, afin de n'obtenir plus qu'un seul segment tout en conservant l'aire initiale. Ce processus peut être répéter un certain nombre de fois jusqu'à ce que le polygone soit un rectangle et ne puisse donc plus être simplifié.

2. Réalisation du projet

a. Orthogonalisation des bâtiments

Mon objectif principal était d'implémenter dans GeOxygene les différents algorithmes présentés dans l'article scientifique afin de pouvoir simplifier certains bâtiments. J'ai donc choisi dans un premier temps d'implémenter la méthode d'orthogonalisation présentée dans l'article.

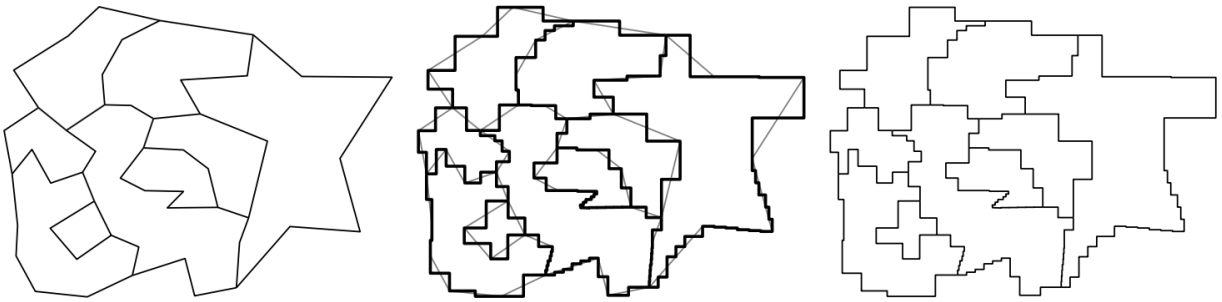


Figure 6 : Exemple d'orthogonalisation de polygone complexe

La méthode utilisée ici consiste à découper les arêtes non-orthogonales en multiples segments orthogonaux. Il a donc fallu dans un premier temps récupérer les données géométriques des bâtiments. Il existe une méthode permettant de récupérer la liste de tous les points décrivant la géométrie d'un bâtiment. Grâce à celle-ci, j'ai pu travailler sur les bâtiments simplement grâce à leurs points décrivant leur géométrie et mon travail a donc essentiellement porté sur ces points. J'ai tout d'abord implémenté une méthode qui permet, pour chaque bâtiment, de récupérer la liste de ses points et de rajouter des points intermédiaires entre chaque paire de points formant des segments non orthogonaux.

J'ai tout d'abord implémenté une méthode similaire à celle présentée dans l'article mis à part le fait qu'elle découpe chaque segment non orthogonal en trois segments orthogonaux (Voir figure 7)

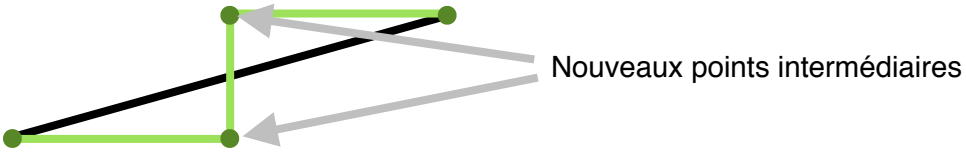


Figure 7 : Technique d'orthogonalisation utilisée

Dans cet exemple, on peut voir que les nouveaux points intermédiaires ont pour coordonnées ((x2-x1)/2 , y1)) et (x2-x1)/2, y2).

Une fois cette méthode appliquée sur tous les côtés d'un bâtiment quelconque, le bâtiment ne possèdes plus que des arrêtes parfaitement orthogonales entre elles et l'aire initiale du bâtiment est conservée.

Cependant, cette méthode comporte un problème dans le cadre de l'utilisation dans un logiciel de cartographie. En effet, les bâtiments sont ici orthogonalisés par rapport à un axe y vertical et un axe x horizontal ce qui n'est pas judicieux étant donné que chaque bâtiment possède sa propre orientation. Ici, on perd les formes initiales des bâtiments et on obtient des formes parfois bien plus compliquées que la forme initiale du bâtiment.

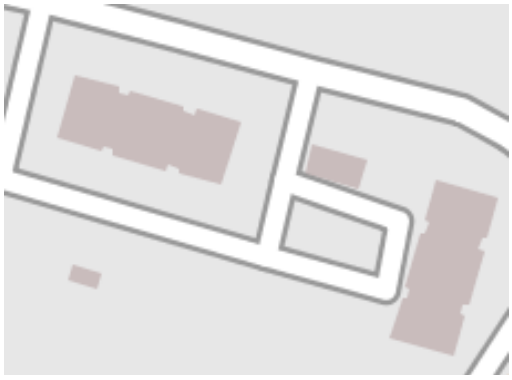


Figure 8 : Bâtiments dans leur géométrie initiale

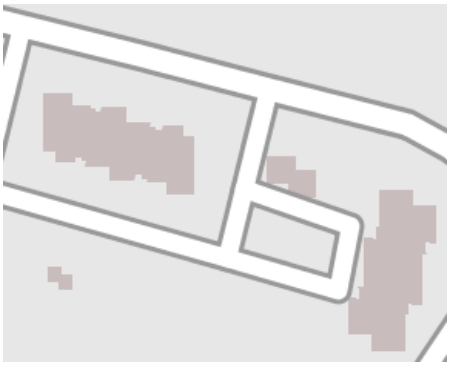


Figure 9 : Bâtiments orthogonalisés

Le problème était donc ici de tenir compte de l'orientation initiale du bâtiment afin de pouvoir effectuer une orthogonalisation en fonction de celle-ci. Pour résoudre ce problème et orthogonaliser les bâtiments en tenant compte de leur orientation, j'ai réfléchi à plusieurs solutions. Mon algorithme d'orthogonalisation étant déjà réalisé et fonctionnel pour des bâtiments orientés horizontalement ou verticalement, il fallait selon moi effectuer une rotation du bâtiment à orthogonaliser afin que son orientation principale soit verticale ou horizontale et par la suite appliquer l'algorithme d'orthogonalisation. Une fois l'orthogonalisation effectuée, on effectue à nouveau une rotation sur ce bâtiment afin de le remettre dans son orientation initiale. Le problème ici était donc d'arriver à déterminer l'orientation principale d'un bâtiment.

La première solution qui est venue à moi est une méthode déjà implémentée dans GeOxygene et qui permet de récupérer l'orientation d'un bâtiment. Cette méthode se trouve dans la classe MesureOrientation et s'appelle getOrientationGenerale(). Elle renvoie l'angle du Polygone correspondant par rapport à l'horizontal sous forme de Double. En analysant le code de cette méthode, je me suis rendu compte que celle-ci se base sur le plus petit rectangle englobant le polygone pour déterminer son orientation principale. Après plusieurs recherches, le plus petit rectangle englobant (aussi appelé rectangle convexe minimum) semble être la méthode la plus fiable afin de déterminer l'orientation d'un bâtiment. La technique est de calculer le plus petit rectangle englobant d'un polygone, et l'angle d'orientation de ce rectangle correspond à l'orientation du bâtiment.

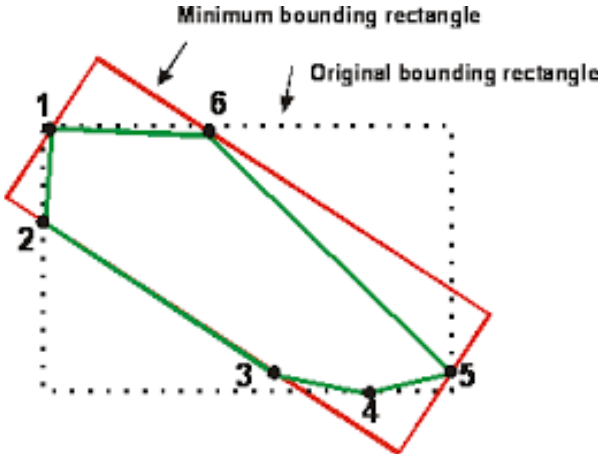


Figure 10 : Exemple du plus petit rectangle englobant

Cette méthode pourrait sembler être une méthode fiable, mais, après plusieurs tests sur différents bâtiments complexes, l'algorithme ne s'est pas avéré parfaitement efficace. Sur les figures ce-dessous, on peut voir que l'orientation générale du bâtiment n'a pas été respecté avec cet algorithme, ce qui rend l'orthogonalisation non-conforme à nos attentes.

Figure 11 : Cité judiciaire dans sa géométrie initiale

Figure 12 : Cité judiciaire orthogonalisée avec la méthode du plus petit rectangle englobant

Cette solution ne faisant pas l'affaire, et l'algorithme étant utilisé par d'autres fonctionnalités de l'application que je ne connaissais pas, j'ai préféré décider d'implémenter moi-même une nouvelle méthode de determination de l'orientation d'un bâtiment. Une première idée fut d'analyser le bâtiment à orthogonaliser, détecter son plus grand côté, calculer l'orientation de ce côté (angle par rapport à l'horizontale) et définir cette dernière comme étant l'orientation principale du bâtiment. Cependant, cette solution n'était pas fiable à 100%. En effet, certains bâtiments, comme la cité judiciaire présente sur la Figure 13 n'ont pas la même orientation générale que celle de leur plus grand côté.

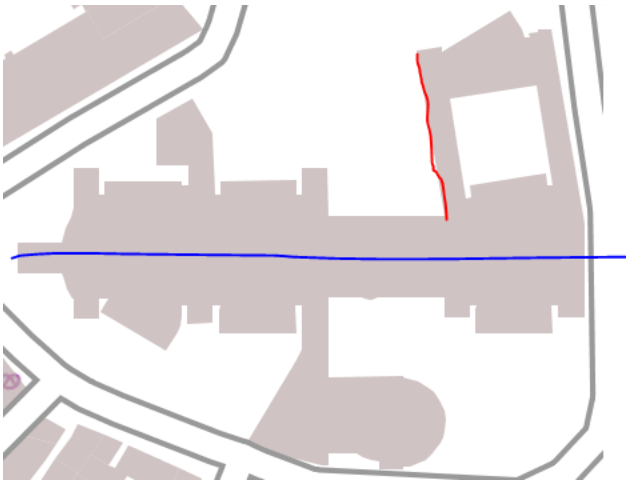


Figure 13 : Exemple du plus long côté sur la Cité judiciaire

Ici, le côté le plus long du bâtiment est le côté rouge, or on désirerait que l'orientation principale du bâtiment soit le segment bleu.

Afin de trouver la véritable orientation des bâtiments, il ne suffit donc pas de regarder l'orientation du plus long côté du bâtiment mais il faut prendre en compte l'orientation de tous les côtés du bâtiment. J'ai par conséquent mis en place un algorithme, qui, pour chaque côté du polygone, regarde son orientation et ajoute la longueur de ce segment à la longueur des autres côtés ayant cette orientation :

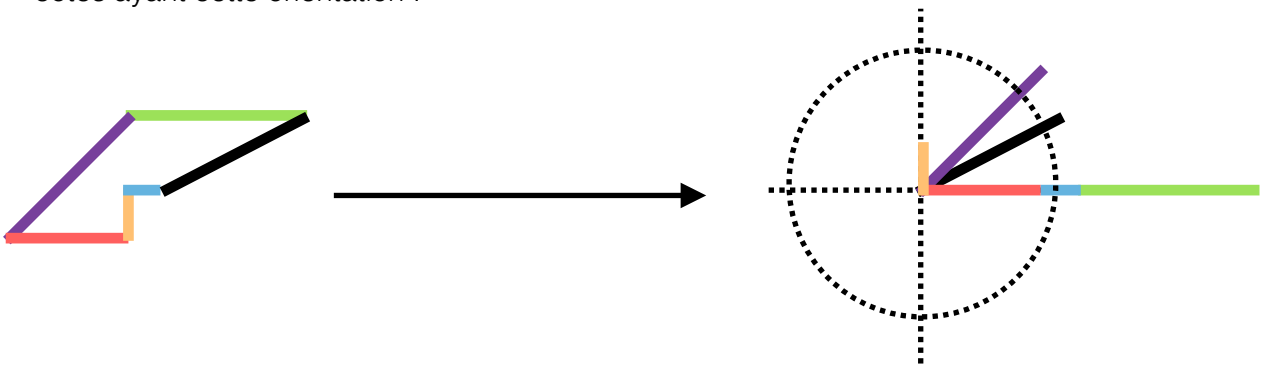


Figure 14 : Illustration de la méthode utilisée pour déterminer l'orientation d'un bâtiment

On ajoute les longueurs des segments ayant le même angle. On peut voir ici que le segment composé des côtés vert, bleu et rouge est le plus long, c'est donc leur orientation qui est la plus représentée donc ce sont eux qui orientent le bâtiment. Cette solution semble être interessante en théorie, mais en pratique, il s'est avéré qu'il y avait quelques manques à cette solution. En effet, les données des bâtiments étant extrêmement précises, il n'y avait quasiment jamais deux côtés d'un bâtiment ayant exactement la même orientation. Il a donc fallu adapter mon algorithme afin de rassembler les orientations à peu près égales.

Cet algorithme permet d'obtenir toutes les orientations existantes des arrêtes d'un bâtiment et permet donc de voir laquelle et la plus représentée, ce qui correspond donc à l'orientation principale du bâtiment. Pour chaque bâtiment sélectionné, l'algorithme va donc determiner son orientation principale, effectuer une rotation pour que le bâtiment soit orthogonal au repère général, effectuer l'orthogonalisation, et effectuer une rotation inverse de la précédente afin de remettre le bâtiment dans son orientation originale.

```
obj.setGeom(CommonAlgorithms.rotation(obj.getGeom(), Math.toRadians(-angle)))
```

Figure 15 : Code utilisé pour effectuer une rotation sur un bâtiment

L'utilisateur est libre de choisir le niveau d'orthogonalisation qu'il souhaite. Par défaut, chaque arrête non-orthogonale est découpé en 3 arrêtes orthogonales. Cependant, cela peut poser des problèmes pour des arrêtes de grande taille et déformer le bâtiment.

Figure 16 : Exemple de différentes précisions d'orthogonalisation

Ici, on peut voir une orthogonalisation plus précise sur l'image de droite, ce qui permet de garder au mieux la forme initiale du bâtiment. Pour permettre à l'utilisateur de choisir son niveau d'orthogonalisation, je lui demande un entier entre 1 et 10. Celui-ci correspond au nombre de fois que sera découpé chaque arrête pour effectuer l'orthogonalisation.

Dans un premier temps, l'algorithme va découper chaque arrête en un certain nombre de segments en plaçant des points intermédiaires sur ceux-ci. Ensuite, tous les points vont être parcourus et des points intermédiaires vont être rajoutés entre chaque paire de points, comme dans la technique présentée précédemment.

Au final l'algorithme ce comporte de la façon suivante. Il regarde s'il y a des objets sélectionnés, s'il y en a, il regarde si ce sont des bâtiments et il les ajoute à une liste de bâtiments. Si rien n'est sélectionné, il ajoute tous les bâtiments de la carte à la liste des bâtiments sur laquelle il va effectuer des traitements. Pour chaque bâtiment de la liste, on récupère la liste des points du bâtiment, on appelle la fonction getMainOrientationPositions qui permet d'obtenir l'angle d'orientation générale du bâtiment. On effectue une rotation du bâtiment afin qu'il soit orienté par rapport au repère monde (vertical/horizontal). On effectue ensuite une analyse de la longueurs des côtés et de leur angle afin de savoir comment les orthogonaliser.

L'objectif ici est de rajouter des points intermédiaires en fonction du choix de l'utilisateur. Pour les segments dont la longueur est supérieure à une certaine longueur et avec un angle supérieur à 5° et inférieur à 85°, on rajoute des points à ce segment de manière à le découper en plus ou moins de parts égales (en fonction du choix de l'utilisateur). Ce choix de 5° est un choix arbitraire qui permet de ne pas découper en trop de segment un côté d'un bâtiment étant déjà presque orthogonal au repère (afin de ne pas avoir un surdécoupage).

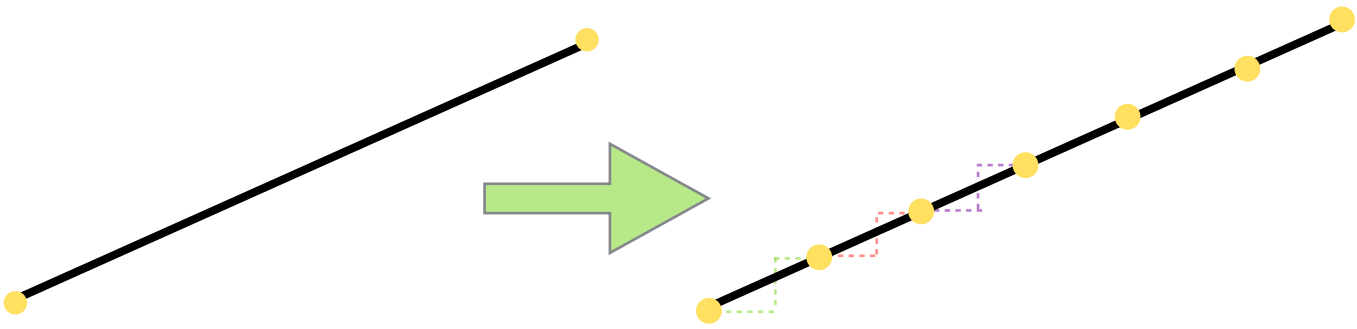


Figure 17 :Illustration du découpage de segment en segments orthogonaux

Ici, le segment est découpé en 6 segments identiques qui seront chacun découpés en segments orthogonaux par la méthode Orthogonalization. Une fois tous les segments « découpés », on passe la liste de points à cette méthode qui renverra une liste de points « orthogonalisés ».

Une fois l'algorithme finalisé, j'ai mis un place un bouton dans un menu déroulant déjà existant qui permet d'effectuer l'algorithme sur tous les bâtiments sélectionnés par l'utilisateur. Voici le résultat obtenu sur une région de plusieurs bâtiments, certains étant complexes et d'autres étant presque de simples rectangles. Chaque segment non-orthogonal a été découpé en plusieurs segments orthogonaux, ce qui va permettre par la suite de grandement faciliter la simplification des bâtiments.

Figure 18 : Illustration du résultat de la méthode d'orthogonalisation

L'algorithme est fonctionnel et permet d'obtenir des bâtiments parfaitement orthogonalisés et qui possèdent la même aire que les bâtiments initiaux. L'orthogonalisation des bâtiment est une étape obligatoire dans la généralisation des bâtiment telle que nous avons choisi de l'effectuer, elle permet de grandement faciliter la simplification des bâtiments, telle qu'elle est présentée dans l'article scientifique à implémenter.

b. Simplification des bâtiments

L'étape d'orthogonalisation était une étape obligatoire afin d'effectuer celle de la simplification. En effet la simplification de polygones, telle qu'elle est présentée dans l'article scientifique, ne peut se faire qu'avec des polygones possédant des arrêtes orthogonales entre elles. L'une des contraintes de cette simplification est qu'elle devait impérativement conserver l'aire initiale du bâtiment. Pour ce faire, il suffit de rééquilibrer les arrêtes deux à deux afin d'obtenir une moyenne pondérée par la longueur des arrêtes.

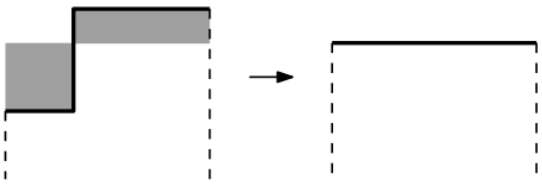


Figure 19 : Méthode de simplification

Dans un premier temps, on vérifie que l'objet sélectionné est bien un bâtiment. Ensuite on copie la liste des points du bâtiments et on s'assure qu'elle comporte plus de 5 points (sinon le bâtiment est un rectangle et ne peut pas être simplifié). On sauvegarde la liste initiale des points dans un historique (liste de liste) afin de pouvoir annuler la simplification si besoin. On supprime également les cours intérieures présentes dans le bâtiment. On détermine ensuite l'orientation principale du bâtiment et on le tourne afin qu'il soit orthogonal au repère général, comme pour l'étape d'orthogonalisation.

Ensuite, afin d'effectuer la simplification, on s'assure d'un certain nombre de critères. On prend chaque ensemble de 4 points adjacents, et on vérifie que la configuration en présence est simplifiable.

Figure 20 : Exemples de configurations simplifiable et non simplifiable

Sur le premier exemple de la figure 20, pour simplifier cette géométrie, on supprime les quatre points initiaux (en jaune) et on rajoute 2 nouveaux points, qui ont pour position en X la même que le premier et dernier points initiaux. En revanche, la valeur en Y de ces points correspond à la moyenne des Y initiaux des terminaisons pondérée par la longueur des segments. Ainsi on a une conservation de l'aire lors de la simplification.

L'algorithme détecte la forme de la géométrie à simplifier et s'adapte en fonction des différentes configurations :

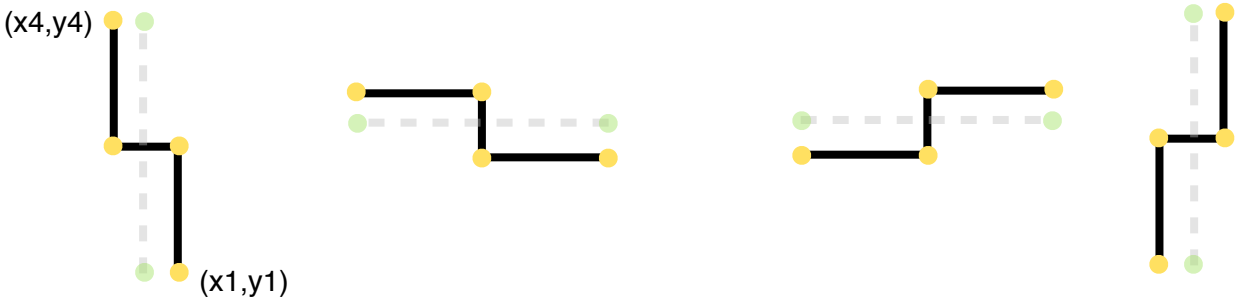
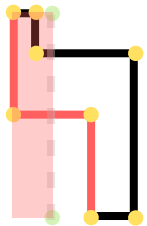


Figure 21 : Exemples de différentes configurations

Sur le premier exemple, les coordonnées finales seront (((x1+x4)/2, y1)((x1+x4)/2, y4)). Le parcours des points se faisant dans un certain sens il a fallu le prendre en compte afin que les coordonnées calculées soient correctes.

Il y a également certaines configurations qui ne doivent pas être simplifiées pour ne pas avoir de croisement :



Ici, si on simplifiait la configuration en rouge, on aurait un problème de croisement par la suite. L'algorithme détecte donc cette configuration (qui peut être orientée de nombreuses manière, c'est pourquoi il y a un certain nombre de tests dans l'algorithme) et décide donc de ne pas simplifier celle-ci.

Figure 22 : Exemple de configuration qui ne doit pas être simplifiée

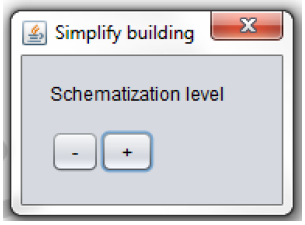


Figure 23 : Fenêtre de gestion de la simplification

L'utilisateur a à sa disposition un bouton dans un menu déroulant lui permettant de gérer la simplification des bâtiments. Celui-ci ouvre une fenêtre lui permettant de simplifier plus ou moins les bâtiments sélectionnés. Lorsque l'utilisateur clique sur le bouton +, on lance l'algorithme de simplification du bâtiment. Celui-ci va parcourir tous les points du bâtiment une fois et simplifier ce qui est simplifiable.

L'utilisateur pourra ensuite cliquer à nouveau sur le bouton + afin de simplifier à nouveau le bâtiment si besoin et ainsi de suite. Une fois que le bâtiment ne peut plus être simplifié (si c'est un rectangle ou si aucune configuration ne permet de simplification), le bouton + se grise et n'est plus cliquable. Si jamais l'utilisateur a besoin de revenir en arrière dans la simplification, le bouton -, l'algorithme va chercher la dernière liste de points dans l'historique et remplace la liste de points du bâtiment par cette dernière, ce qui permet donc de récupérer la dernière version de la géométrie du bâtiment.

À chaque opération de simplification, on actualise la vue afin de permettre à l'utilisateur d'avoir un rendu immédiat.

```
ProjectFrame selectedProjectFrame = GeOxygeneEventManager.getInstance().getApplication().getMainFrame().getSelectedProjectFrame();
selectedProjectFrame.getLayerViewPanel().repaint();
```

Figure 24 : Code permettant la mise à jour de la vue

La méthode de simplification des bâtiment est relativement longue est complexe à cause des nombreuses configurations possibles et de leurs différents traitements. De nombreuses conditions sont testées avant d'effectuer la simplification afin de s'assurer que le bâtiment est bien simplifiable. Cette méthode effectuant des traitements pour chaque groupe de points contenus dans chaque bâtiment sélectionné, et effectuant beaucoup de vérifications, il a fallu s'assurer que l'exécution de l'algorithme ne prenait pas trop de temps de traitement. Malgré sa complexité, l'algorithme effectue son traitement quasiment instantanément et ne pose donc aucun problème à ce sujet. En Annexe 1 et 2 se trouvent des exemples de simplification effectuée sur des bâtiments à géométrie complexe.

3. Bilan et perspectives

Ce projet avait pour but d'implémenter deux techniques de généralisation complémentaires qui sont l'orthogonalisation et la simplification de bâtiment. L'objectif était d'implémenter les techniques présentées dans un article scientifique traitant le sujet. Cependant, cet article ne fournissait pas suffisamment d'information pour pouvoir implémenter les techniques présentées telles quelles. En effet, à aucun moment dans l'article il n'est mentionné le sujet de l'orientation des polygones lors de l'orthogonalisation. Ce sujet a été pourtant un des plus complexes et intéressant du projet et c'est sans doute celui sur lequel j'ai passé le plus de temps. Dans l'article, il est en effet supposé que les polygones sont déjà tous orientés parfaitement. Les techniques de simplification sont également présentées sans prendre en compte les différentes contraintes qu'il peut exister au sein d'un polygone. Comme je l'ai décrit précédemment, de nombreuses configurations ne doivent pas être simplifiées comme il le décrivent au risque de perdre l'intégrité du polygone.

Bien que ces deux techniques aient été implémentées et sont parfaitement fonctionnelles, il subsiste quelques points d'amélioration notables. Par exemple, lorsque l'on simplifie un bâtiment, il semblerait intéressant de choisir quelle façade du bâtiment on souhaite simplifier par exemple, au lieu de simplifier directement l'ensemble du bâtiment. Il semblerait intéressant pour l'utilisateur d'avoir plus de contrôle sur l'orthogonalisation et la simplification des bâtiments. Il existe également plusieurs simplification possibles pour chaque bâtiment, selon le point de départ de la boucle réalisant l'opération, il pourrait donc perte judicieux de proposer à l'utilisateur de choisir la version simplifiée qu'il préfère. Il pourrait sembler intéressant également de permettre à l'utilisateur de choisir les arrêtes de bâtiment qu'il souhaite orthogonaliser. La mise en place de simplification de bâtiment sans passer par l'étape d'orthogonalisation pourrait également être intéressant et permettrait d'obtenir des bâtiments plus proches de la réalité mais cela semble beaucoup plus compliqué à mettre en place.

Conclusion

Durant ces deux mois de stage au LIMOS, j'ai pu découvrir une application du développement que je connaissait peu, le traitement de données géographiques. En ayant comme base de travail un logiciel déjà très complet et conséquent, j'ai eu à implémenter deux fonctionnalités importantes tout en respectant les normes de développement établies et l'intégrité du projet. À ce jour, l'objectif d'implémenter les méthodes décrites dans l'article scientifique a été atteint. L'implémentation fut très différente et plus compliquée que la description faite par l'article scientifique mais celui-ci donnait tout de même des bases de départ importantes. C'était la première fois pour moi que j'avais à effectuer un développement en me basant sur un article scientifique, et cela m'a paru intéressant de confronter les idées présentées aux miennes, de voir l'approche qui a été utilisée pour résoudre ces problèmes d'orthogonalisation et de simplification.

Malgré les explications données par l'article scientifique, la majeure partie du projet fut également de la recherche de mon côté afin de trouver des moyens d'adapter les méthodes décrites au sujet concret. Plusieurs problèmes ce sont présentés mais avec l'aide de spécialistes en informatique, mathématiques et cartographie, ils ont été résolus avec succès.

Ma tâche s'inscrivant dans le projet de Cartes Tactiles à destination des personnes atteint de déficiences visuelles, j'ai pu être sensibilisé à ce problème et acquérir diverses connaissances à ce sujet. À ce jour, il est possible d'imprimer des cartes tactiles avec des bâtiments simplifiés, afin que les personnes mal-voyantes ou non-voyantes puissent avoir une vue d'ensemble d'un quartier d'une ville simplement, sans avoir de superflu. La simplification des bâtiments a également permis de permettre une impression légèrement plus rapide, les bâtiments ayant moins de détails.

D'un point de vue technque, ce projet m'a permis de faire travailler mes connaissances en mathématiques ainsi que d'en apprendre de nouvelles, d'améliorer mon niveau de développement en Java, ainsi que la gestion de projet, étant donné que nous étions quatre à travailler en même temps sur la même application.

Webographie et bibliographie

Area-Preserving Subdivision Schematization
publié par Wouter Meulemans, Andrée van Renssen et Bettina Speckmann

Urban and Regional Data Management: UDMS 2007 Annual
publié par Massimo Rumor,Volker Coors,Elfriede M. Fendel,Sisi Zlatanova

Minimum-Area Rectangle Containing a Set of Points
publié par David Eberly

wikipedia.org : renseignements sur le domaine mathématique

openclassroom.com, stackoverflow.com, developpez.net : aide au développement

mathematiquesfaciles.com : aide à la compréhension de principes mathématique

Annexe 1 : Exemple de simplification de bâtiment

Annexe 2 : Exemple de simplification de bâtiment