PHELMA
Grenoble INP
Minatec – 3 Parvis Louis Néel
CS 50257 – 38016 Grenoble Cedex 1

University of Nevada, Las Vegas
Transportation ,Research Center
4505 Maryland Parkway, PO Box 454007
Las Vegas, NV 89154-4007
United States of America

Internship Report

Developement of a trafic and driving simulator based on Open Street Map

Major: Software development, C++, Object-oriented Programing
Code and diagrams are available on Git-Hub : https://github.com/Dimitri1/glosm

Dimitri Gerin
SICOM 2nd year

Supervisor
Dr. Alexander Paz, Ph.D., P.E.
Transportation Research Center,
University of Nevada-Las Vegas

Technical leader
Romesh Khaddar, Graduate Assistant
Transportation Research Center,
University of Nevada-Las Vegas

07/10/2013

Résumé (long)

Introduction

Pour ma 2ème année de cycle ingénieur SICOM à PHELMA, j'ai eu la chance d'effectuer un stage de 10 semaines dans le *Transportation Research Center* de l'Université du Nevada, à Las Vegas. Durant ce stage, j'ai pris part à un projet de programmation orientée objet en langage C++, visant à développer un simulateur de trafic de conduite. Ce simulateur doit produire un rendu 3 dimensions du trafic routier et le simulateur de conduite doit également permettre à l'utilisateur de contrôler un véhicule à la première personne. La finalité de ce projet est d'être adapté pour la plateforme physique de simulation du TRC (*Annexe 4*), et ainsi de fournir un système permettant d'analyser le comportement du conducteur ainsi que l'influence de sa conduite sur le trafique, à travers des recueils de statistiques en temps réel.

Plan du résumé

Dans ce résumé de rapport, vous trouverez une première partie concernant les logiciels utilisés, suivie de la présentation de l'UNLV et du TRC. Ensuite nous présenterons Open Street Map, puis nous le projet glosm, qui à été utilisé comme point de départ pour notre simulateur. Nous verrons aussi quelques définitions concernant les simulateur de conduite et de trafique puis nous présenterons un modèle mathématique décrivant l'accélération d'une automobile dans différents régimes de conduite. Dans la partie suivante sera présenté le travail de développement qui a été fait durant ces 10 semaines, et nous finirons par la conclusion.

Logiciels utilisés

J'ai eu à utiliser plusieurs logiciels nouveaux durant ce stage, notamment Cmake, Git et Doxygen. Cmake est un logiciel de compilation multi plateforme qui offre une interface de compilation commune à tous les principaux systèmes d'exploitation. Cela sert donc à pouvoir compiler notre projet sous un système basé Linux ou sur Windows avec le même script de compilation (CmakeList.txt). Git est un logiciel un logiciel de gestion de versions décentralisé très utilisé de nos jours par les programmeurs. Il est conçu pour la gestion de projet taille importante, et offre donc des fonctionnalités avancées pour faciliter et sécuriser le partage des fichiers source. Git fonctionne de paire avec Git-Hub, qui permet de stocker et de partager les projets de chacun à travers un site interner interactif. Git est utilisé par la communauté des développeurs libres mais aussi par les entreprises, car Git-Hub permet en effet de créer des dépôts privés (code non accessible au public) moyennant des frais fonctions de la taille du dépôt. Enfin j'ai eu à utiliser Doxygen, un gestionnaire automatique de documentation. Doxygen scrute le code et extrait de manière automatique toutes les instances d'intérêt en fonction du langage utilisé (variables, fonctions, classes…), et produit une documentation agrémentée des commentaires placés dans le code qui sont eux aussi extrait de manière automatique.

Présentation de l'UNLV et du TRC

L'Université du Nevada (UNLV) est une des université principales des Etats-Unis et est classée comme université à haute activité de recherche. Elle comporte plus de 220 programmes d'enseignement répartis dans de nombreux domaines académiques et allant de l'équivalent français de la licence (undergraduate's programs), passant par des master et allant jusqu'aux thèses. Elle comporte 28,000 étudiants. Le Centre de Recherche en Transport (TRC) est situé dans le bâtiment des sciences et de l'ingénierie de l'UNLV, le SEB. C'est le centre de recherche principal du centre de recherche en transport du Nevada (NTUC). La principale mission du NTUC est de mener des études et proposer des solutions technologiques pour faire évoluer le domaine de la gestion des transports dans des zones urbaines à croissance rapide.

Présentation d'Open Street Map (OSM)

Open Street Map est un projet collaboratif de création d'une carte du monde éditable librement. Crée par Steve Coast au Royaume-Uni en 2004, il a été inspiré par le succès de Wikipédia et par la volonté de contrer la prédominance du caractère propriétaire des données cartographiques à travers le monde. Une des spécificités de notre simulateur est qu'il doit comporter générer sa carte à partir des données Open Street Map, et non à partir d'une carte fictive, comme cela est souvent le cas dans les jeux vidéo par exemple. Les données XML sont écrites au format XML (Extensible Markup Language). Au même titre que HTML, XML est un format qui a été crée dans le but d'être facilement compréhensible par les machines et les humains. Il est composé de tags, d'éléments et d'attributs. Tous les principaux types d'éléments peuvent être représentés en XML comme les *string* et les *float*. XML est hiérarchique. Les tags sont utilisés pour instancier chaque objet et chaque objet comporte un ou plusieurs attributs. Dans OSM, il y'a deux types d'éléments ; les *nodes* (nœuds) et les *way* qui peuvent représentent tout type d'objet du monde réel comme les routes, les bâtiments et les arbres. Les *way* et les *node* sont complètement interdépendants. Les node servent simplement à déclarer des points. Dans notre cas, en 2 dimensions, le tag *node* à donc 2 attributs latitude et longitude pour représenter la position de chaque point. Les *way* se déclare en listant un certain nombre de *node* dans un premier temps, puis en citant son type dans un second temps. Par exemple, si on veut déclarer un bâtiment carré simple, on aura besoin d'au moins 4 *node*, qui représenteront l'emprunte au sol du bâtiment. Les cartes OSM sont directement téléchargeables sur le site *openstreetmap.org* mais la taille maximale d'exportation est assez limitée. Il est néanmoins possible de trouver des cartes de taille plus importantes et même la carte du monde entier sur planet.osm.org.

Principe de glosm

L'idée principale de glosm est de produire un rendu 3 dimensions des données OSM. Dans ce projet, on cherche donc à avoir une carte 3 dimensions à partir de données 2 dimensions ce qui peut paraitre étrange. En effet, de par le fait d'être limité par les données d'entrée dans lesquels il manque une dimension, le rendu de sortie 3 dimensions ne pourra forcément pas être très détaillé. Heureusement, les donnés OSM comporte des informations sur la hauteur de chaque bâtiment, et c'est bien évidemment cette donnée qui est exploité dans glosm. Glosm permet également de générer plusieurs types de toitures à conditions bien sure que cela soit précisé sous forme d'option dans le ficher source, ce qui n'est la plupart du temps pas le cas dans OSM (si rien n'est précisé, les toitures sont par défaut des dalles plates). Concernant les routes, il faut savoir qu'il n'existe que des lignes droites dans OSM et dans glosm, et pas de courbes. Les routes courbées sont simplement une juxtaposition de segment de routes droites inclinées l'une par rapport à l'autre.

Architecture et implémentation de glosm

Glosm fonctionne avec différentes couches graphiques. Le fichier source contenant les donnés OSM (.osm) est *parsé* par un parseur XML afin d'extraire de manière structuré, toutes les primitives. Ensuite viens le *GeometryDatasource processor* qui génère des triangles 3 dimensions pour représenter chaque primitive OSM crée précédemment. Ces triangles sont alors associés de manière géographique (placement des triangles par leurs coordonnées) par le GeometryLayer *processor*. Tout cela constitue la couche appelée couche géométrique (GeometryLayer) qui est la couche principale de glosm, dans lequel est représenté le monde OSM 3 dimension. Cette couche géométrique est composée de données de type *GeometryTile* qui sont les faces de chaque objet 3D de glosm. Ces faces sont le produit du placement géographique des triangles du *GeometryDatasource processor*. Il existe une seconde couche dans glosm, qui sert à représenter les traces GPS, appelées *GPX traxes* à partir d'un fichier source *.gpx* au format XML. Ces deux couches sont fusionnées par le *Render*.

Parser : En informatique, *o*pération qui consiste à parcourir un fichier et extraire des informations d'intérêt.

Enfin le *Viewer* permet de visualiser le résultat à travers un mode de vue à la première personne (FirstPersonViewer). Glosm est implémenté à travers 3 librairies : *libglosm-geomgen, libglosm-server, and libglosm-client*. *libglosm-geomgen* implémente la génération des primitives géométriques de base. *libglosm-server* implémente toutes les fonctions de traitement des sources OSM et permet de générer les facettes des objets 3D (*tiles*). La librairie client permet de fusionner les données de type *tile.* Enfin, le cœur d'exécution glosm est implémenté dans le dossier *viewer* qui initialise OpenGL les librairies glosm. Il démarre ensuite le mode de vue à la première personne lorsque le programme est lancé. Le système de compilation de glosm est basé sur Cmake. Des CmakeList sont présents dans chaque dossier de glosm pour lister les sources. Un CmakeList principal est présent à la racine de glosm et sert à compiler le projet final en listant tout les dossiers utiles (librairies), et les dépendances.

Simulateur de conduite et de trafique : Model et définitions

On peut considérer le simulateur de conduite comme une extension du simulateur de trafique dans lequel l'utilisateur contrôle un des véhicule. La plupart des simulateur de trafique sont découpé en 2 niveaux différents de description, le niveau mésoscopique et microscopique. Le niveau microscopique décrit les informations concernant chaque véhicule alors que le niveau mésoscopique ne traite qu'avec les flux routier (voir figure 10 du rapport). Le principe est que la zone de visibilité du conducteur est considérée comme étant la zone microscopique et la zone invisible est gérée de manière mésoscopique. Puisque le niveau microscopique traite les informations au niveau de chaque véhicule, il doit être muni de modèle qui régit l'interaction entre ces véhicules. Bien qu'ils n'aient pas été implémentés durant ce stage, j'ai eu à faire des recherches concernant ces modèles. Un des modèles les plus difficile à appréhender est celui qui régit le régime de suivi des véhicules, c'est à dire le modèle qui définit l'accélération du véhicule en fonction du temps. Un des modèle les plus connu est qui est utilisé dans plusieurs outils de simulation du TRC à été développé au MIT, basé dur l'approche MITSIM. Il définit mathématiquement l'accélération du véhicule en fonction du temps pour 3 cas différents : Suivi proche d'un véhicule, conduite libre et régime d'urgence.

Simulateur de conduite et de trafique : Implémentation dans glosm

Nous avons du dans un premier temps définir une méthode de travail. Nous avons donc choisi une méthode incrémentale, qui est adaptée à la programmation orientée objet. L'objectif de l'incrément un est double. Il doit fournir dans un premier temps un mode de test du simulateur sans faire appel au aucun modèle 3 dimension d'automobile. Autrement dis, on doit pouvoir tester les futures fonctions qui vont gérer la dynamique des véhicules de manière graphique, directement sous le *FirstPersonViewerMode,* et ceci sans avoir de modèle 3 dimension pour représenter nos véhicules. Cet incrément doit aussi dans le même temps, fournir les classes de base nécessaire à la représentation et au stockage des véhicules. Concernant l'incrément 2, le but principal est d'implémenter le Gestionnaire de navigation, qui doit comporter toutes les fonctions nécessaires au placement, au déplacement et à l'interaction de chaque véhicule dans le réseau routier de *glosm.*

Incrément 1

Comme nous voulons bâtir un code dans glosm, nous sommes totalement dépendants de la philosophie de glosm, qui est elle même basée sur OpenGl, sa librairie graphique. Nous cherchons donc à implémenter notre code de la façon la plus indépendante possible de glosm. Pour cela, nous avons donc choisi d'implémenter l'exécution des fonctions relatives aux déplacements des véhicules dans des thread détachés. C'est donc dans ces fonctions de thread détaché que nous allons exécuter le Gestionnaire de Navigations (incrément 2). Un thread sera exécuté par véhicule. Les résultats produits pas ces thread seront des trajectoires, c'est pourquoi nous avons du trouver un moyen de les observer. Pour cela, nous avons utilisé une fonctionnalité déjà existante dans glosm, qui permet la visualisation des traces GPS (voir partie principe de glosm) dans le mode de vue à la première personne. Nous avons donc crée 2 modes d'exécution. Le premier est le *GPX_Write_mod,* dans lequel s'exécute la simulation et dans le quel chaque thread produit sont propre fichier .gpx,

pour représenter la trajectoire du véhicule simulé. Ensuite, on exécute le *GPX_Read_mod*, qui permet d'observer toutes les traces GPX au mode de vu à la première personne, en chargeant tout les fichiers écris dans le *GPX_Write_mod* (voir figure 11). Concernant la représentation des véhicules, on les représente pour l'instant de la manière la plus simple. Ils ont une positions sous forme de vecteur 2 dimensions et un numéro d'identification. Nous avons implémenté, la classe *CarListsLoader* qui permet de charger automatiquement une liste de véhicule à partir d'un fichier au format CSV. Les fonction de thread sont appelés au sein de la class *CarGlosm* et on accès à tout les autres véhicules grâce à un pointeur sur le conteneur de véhicule (CarGlosmContainer). Pour l'écriture des trajectoires dans les fichier .gpx, on utilise la classe *GPXWriter*.

Incrément 2

L'incrément 2 a été la difficulté principale de ce stage en grande partie à cause du fait que les donnés OSM ne sont pas adaptées à notre problème. Le but est de créer des fonctions permettant de gérer la position des véhicules dans le réseau routier de type OSM. Nous avons vu précédemment que les objets de type route (*highway*) étaient déclarés de la même façon que tout autre type d'objet. Nous avons donc implémenté un système appelé *point-on-a-way check* qui permet d'acquérir des informations sur le réseau routier à proximité du véhicule. Pour cela, nous avons d'abord pensé à une première méthode naturelle qui s'est avérée trop couteuse en nombre d'opération, puisqu'elle impliquait une complexité o(N.N.W.n), ou N est ne nombre de *node* present dans le fichier .osm, W le nombre de *way* et n le nombre de *node* référencé dans chaque *way* (voir présenation de OSM). Une meilleure solution fut adoptée par la suite. Elle utilise une fonctionnalité de glosm, la *Boundary Box (BBox),* qui permet d'extraire les *way* contenu dans une surface carrée dont les coordonnées des côtés sont passées en paramètres. L'idée est donc de définir une zone de détection autour du véhicule et d'utiliser la *BBox* pour détecter les coordonnées des routes et ainsi de pouvoir en déduire des trajectoires afin de faire circuler la voiture sur la route. Néanmoins, l'utilisation de *BBox* nous impose de faire une seconde opération de filtrage puisque BBox nous renvoi l'ensemble des *way (au sens tag)* contenu dans le carré et non pas les routes *(highway)* uniquement. Lors de cet incrément 2, nous avons implémenté un second système nécessaire à la génération des trajectoires, le *trajectory builder*. Il permet d'obtenir un vecteur composé des coordonnées des points de la ligne, qui joint le point de départ et le point d'arriver passé en paramètre.

Conclusion

Je suis satisfait de ce stage puisque j'ai pu apprendre et approfondir la programmation objet en langage C++ et découvrir quelques outils réellement utilisée dans le monde de l'informatique tel que GIT. Un point intéressant de ce stage est que mon travail était assez libre du point de vue des choix technique, et je remercie de la confiance que mon responsable technique Romesh Khaddar pour la confiance qu'il m'a accordée. Cette liberté de développement fut par ailleurs une difficulté du fait qu'on peut aisément de faire de mauvais choix, et tomber dans des impasses technique.