

PHELMA

University of Nevada,

Las Vegas Transportation ,Research Center

*4505 Maryland Parkway, PO Box 454007*

*Las Vegas, NV 89154-4007*

*United States of America*

Grenoble INP

Minatec – 3 Parvis Louis Néel

CS 50257 – 38016 Grenoble Cedex 1

Internship Report

Developement of a trafic and driving simulator

based on Open Street Map

Major : Software development, C++

Code and diagrams available on Git-Hub : <github.com/Dimitri1/glosm.git>

**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

Acknowledgement

First, I would like to thank Dr. Alexander Paz for accepting me in the UNLV’s internship program, and giving to me the opportunity to accomplish a part of my study abroad an in addition in an English speaking country. I also would like to thank him for assigning me in an interesting project, where I learn a new language and new tool.

I want to thank Tamar Vered and Penny Tan, from the UNLV’s Office of International Students and Scholars for giving me much of their time for all the administrative formalities, essentially the visa procedure.

I particularly want to thank Romesh Khaddar, my technical responsible for assisting me during the development and giving me interesting advices and methods. Thanks to him, I discovered new tools and systems, which will be useful for me in the future.

Finally, I want to thank the Rhone Alpes district for giving me a financial grant and Margaux Ravix from Grenoble INP, for the administrative preparation.

Summary

Introduction

1. Used Software
2. Brief presentation of the UNLV and Transportation Research Center (TRC).
3. Presentation of Open Street Map
4. Presentation of Glosm Project
   1. Glosm Principle
   2. Glosm architecture
   3. Implementation
5. Traffic and driving simulation: Models and definition
6. Traffic and driving simulation: Implementation on glosm
   1. Increment 1 : Basics classes
   2. Increment 2 : Position and motion handler for cars
   3. Further development: Advanced architecture: Adding Player Hander
7. Conclusion
8. Bibliography and useful sources
9. Glossary

Appendix

Introduction

For my 2nd year in PHELMA, I chose to attend a 10 week internship in the Transportation Research Center of the University of Nevada, Las Vegas.

During this internship, I took part on a project of development traffic and driving simulator.

The driving simulator part is a matter of 3D rendering of the complete real time traffic simulation, for which the player control one car, in a first person viewing mode.

The final goal of the traffic simulator project is to provide a way to analyze the influence

of human driver behavior on the whole traffic simulation. This could be used to gather information in order to improve security or to analyze the indirect repercussions of human factors on the traffic.

My first task was to implement a framework for the 3D visualization based on Open Street Map data to generate the map.

Then, in a second step, I hade to use this framework a starting point to build a traffic and driving simulation system upon it. Regarding the traffic simulation (without visualization), many models of Cars Behavior already exist. All these models govern the interactions between the cars, and are based on statistical model. Even if we haven’t reached the step of Car behavior model implementation during this internship, we will present one of them which may be used in a further development.

A first extremely basic rendering framework has already been implemented with Blender,a [free and open-source](http://en.wikipedia.org/wiki/Free_and_open_source_software) [3D computer graphics software.](http://en.wikipedia.org/wiki/3D_computer_graphics_software) This rendering framework has been merged with a micro-simulation core model, but it was not satisfying for the needs of TRC.

Actually, this first project was not based on open street map data, and in addition, they got compatibility problems with Blender when they tried to implement it the code on the simulator computer.

So, my supervisor requested me to use the OpenGL Library for the graphical part, because he absolutely didn’t want that our traffic-simulator based on any [3D engine, or any advanced graphic software.](http://en.wikipedia.org/wiki/3D_computer_graphics_software) Open GL is a low level library to generate 3D geometries, and many other more advanced graphics like lighting and shadowing effects.

As we didn’t wanted to start from zero for the Graphical part, the first main part of my internship was to prospect about existing open source projects of 3D OSM map Visualization based OpenGL library.

Then, the second main task of my internship was to build our traffic and driving simulation on the existing chosen project, as an extension feature.

In this report, I will first do a quick presentation of the University and the Transportation Research Center, then I will describe the used tools.

I will continue by presenting the Glosm project, which have been chosen for building our simulation system. And, in a short part to introduce my real contribution to this project, we will present some traffic-simulation global idea, and give some concepts and definitions about traffic-simulation existing chosen model.

Finally, in the last part, we will detail as more concrete as possible, the traffic-simulation implementation choices and issues.

1. Used Software



Cmake is a Cross-platform making software, and I had to use it for compiling my projects in C++.

It makes the compilation possible for all the platform (Linux, Window, and Mac OS), with the same Compilation description file. This Compilation description file is named CMakeList.txt, and lists the sources, the headers, and all the useful libraries, which takes parts in the projects. When launched, Cmake scans the CMakeLists.txt and produces all the compilation files depending on the OS. So, for Linux based OS, Cmake produces a Makefile and with Visual studio it produces the

Visual Studio projects files.

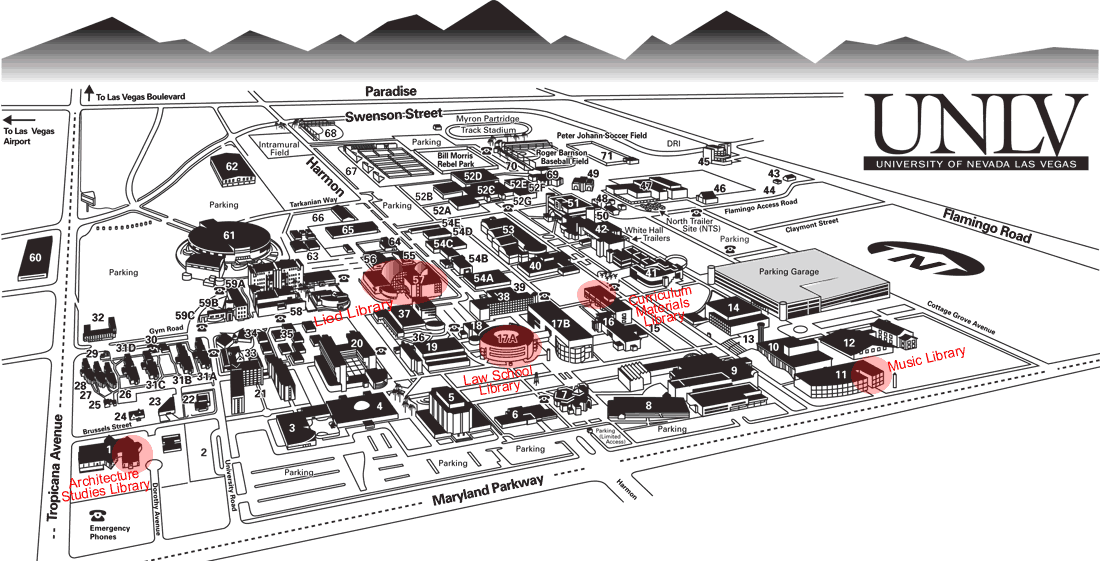
****

Git is a distributed version control and source code management system. It is widely used by many software development communities around the world. It provides many useful functionality for project management like complete history and full version tracking capabilities. For each project, the principle is the same. All the project sources are referenced in a local git repository, present on the local host. Every modification of the tracked sources are detected and referenced by git and saved in the Git repository file system. The user must commit every modification at each step of development and in addition, pushing (uploading) the local repository on the git hub server. One of the fundamental feature of Git is to create branches, whishes are separate way of implementation, exactly like clones. For instance, when the user needs to implement a new functionality, he just has to create a new branch in his repository, on work on this branch. All the new commitments (changing validation) will exist on this new branch and the previous code (before branching) won’t never be affected by the changes. By this way, if the user wants to regress and coming back to the previous state of development, he just has to going back on the previous branch (often the main branch named *master*). If the source code tracked by this new branch is satisfying, the user can merge the new branch to the original branch. At every time of development, the user can save his files on the git-hub, as a copy of the local git repository (present on his computer).

dim_dim:private:var:folders:Id:Id42F6jhHtqZr8yBEdzZCE+++TI:-Tmp-:TemporaryItems:doxygen.png

Doxygen is software, which scans the tracked source files of a project and produces documentation in much format type like html, PDF and Latex. It is very practical to use because Doxygen simply scans the code and extract automatically all the instances of a source code like class, attributes and methods if case of an oriented object language like C++. In addition, it extracts the comments for each class, attributed and method and finally builds a rich documentation. I used it to generate a website, which contains all the detailed information about my code.

2. Brief presentation of the UNLV.



*Figure 1 : UNLV Campus map*

UNLV (University of Nevada, Las Vegas) is a premier metropolitan research university. Its 332-acre (134-ha) main campus, located on the Southern tip of Nevada in a desert valley surrounded by mountains, is home to more than 220 undergraduate, master's, and doctoral degree programs, all accredited by the Northwest Commission on Colleges and Universities.

UNLV is a doctoral-degree-granting institution with more than 28,000 students, more than 7,000 of who are graduate/professional students. Nearly 120 graduate degree programs are offered, including 36 doctoral and professional degrees. The university is ranked in the category of "high research activity" by the Carnegie Foundation for the Advancement of Teaching. UNLV offers a broad range of respected academic programs.



*Figure 2 : Science and Engineering Building (SEB), UNLV*

The Transportation research center, located on the SEB (refer to figure 2), which is the most recent and modern building of the whole UNLV campus. It gathers many scientific an technic field of research and teaching like computing science, electronics, micro electronics, materials, physics, biology, and civil engineering.

3. Presentation of Open Street Map http://www.openstreetmap.org/



Open Street Map Wikipedia definition: <http://en.wikipedia.org/wiki/OpenStreetMap>

**OpenStreetMap** (**OSM**) is a [collaborative project](http://en.wikipedia.org/wiki/Virtual_community) to create a free editable [map](http://en.wikipedia.org/wiki/Map) of the world.

Created by [Steve Coast](http://en.wikipedia.org/wiki/Steve_Coast) in the UK in 2004, it was inspired by the success of [Wikipedia](http://en.wikipedia.org/wiki/Wikipedia) and preponderance of [proprietary](http://en.wikipedia.org/wiki/Vendor_lock-in) map data in the UK and elsewhere.

The specificity of our driving simulator is that the 3D map has to be generated from Open Street Map data and not from an arbitrarily builded map, like a city in a video game for example. The Glosm project (described in the next part of the report) takes his information from OSM. So for my internship, I had to understand how the OSM data are organized.

**Data management in OSM**

The OSM data’s (.osm) are written in XML format.

XML Wikipedia definition: <http://en.wikipedia.org/wiki/XML>

**Extensible Markup Language** (**XML**) is a [markup language](http://en.wikipedia.org/wiki/Markup_language) that defines a set of rules for encoding documents in a [format](http://en.wikipedia.org/wiki/File_format) that is both [human-readable](http://en.wikipedia.org/wiki/Human-readable_medium) and [machine-readable](http://en.wikipedia.org/wiki/Machine-readable_data).

This language is basically composed of tags, elements and attributes, and could describe all elements in hierarchical way, and accepts many attributes types, like strings and float. The tag are used to instantiate an object. Each tag represents an element, and each element contains attributes.

**Example:** **<img** src="madonna.jpg" alt='Foligno Madonna, by Raphael'**/>**

This is a declaration of an *img* element, which contains two elements *src* and *alt*, which takes the values of "madonna.jpg” and 'Foligno Madonna, by Raphael' respectively. In OSM there are two different types of element, the way and the nodes. The nodes are the remarkable geometric point (or vertex). The way is an abstract element and can represent many sub types like buildings, highway, and tree for example. The *way* declaration is fully dependent of the nodes, because *ways* are builded on nodes. For example, for the declaration of a square shape building, we must first declare 4 nodes, with them respective position attributes (latitude and longitude in OSM), and then declare a way of sub type building, and mention that this building is associated with the 4 nodes described above.

<node id='-23563' action='modify' visible='true' lat='53.893335223876946' lon='-29.529670738241325' />

<node id='-23562' action='modify' visible='true' lat='53.89336616257267' lon='-29.529906995604907' />

…

…

<way id='-23464' action='modify' timestamp='2011-02-20T17:58:02Z' visible='true'>

<nd ref='-23144' />

<nd ref='-23138' />

<nd ref='-23140' />

<tag k='highway' v='service' />

</way>

<way id='-23462' action='modify' timestamp='2011-02-20T17:58:02Z' visible='true'>

In this example, we can see a declaration of two nodes and a declaration of a ‘highway’ way type.

**Exportation of OSM data**

It is possible to export any place of OSM map , directly from the website, but the size is limited

in terms of nodes number. However, there are many Internet sites, which make large map available. It is possible to find .osm files for each major city. It is even possible to find the complete file

which reference the whole earth planet in planet.osm.org, but the size is 320 GB uncompressed.

4. Presentation of Glosm Project

Glosm Wiki definition : <http://wiki.openstreetmap.org/wiki/Glosm>

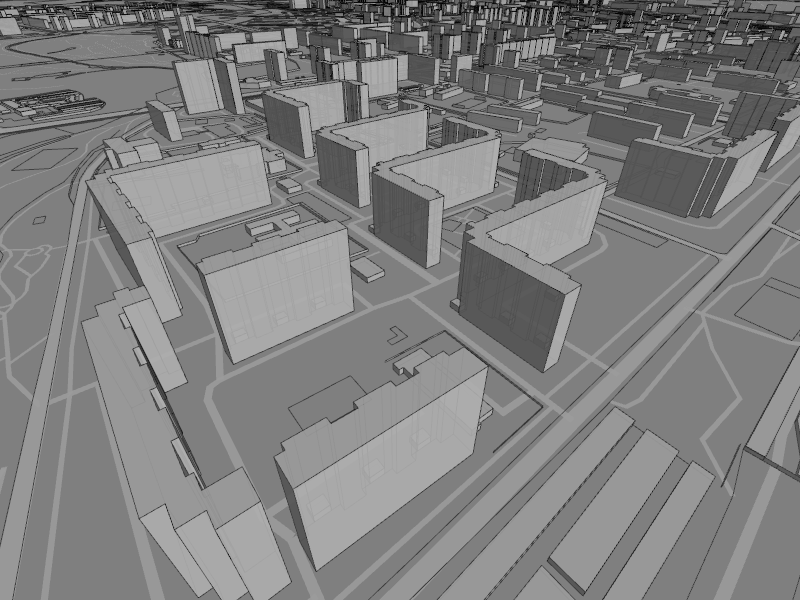
*Glosm is hardware-accelerated OpenGL-based OpenStreetMap renderer by*[*User:AMDmi3*](http://wiki.openstreetmap.org/wiki/User:AMDmi3)*. Both tile rendering and realtime first-person viewer are supported.*

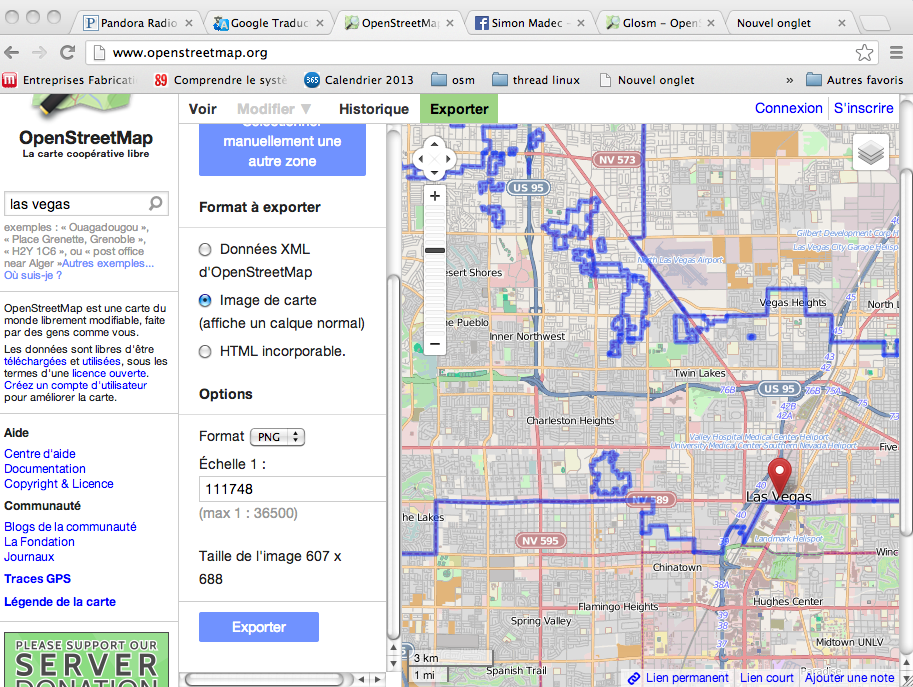
The Glosm project is exactly what I needed as a starting point, because it already provide a stable 3D generation of OSM data, and a real-time first-person viewer, as cited above. The author of this project is Dmitry Marakasof, an active Open Source Software Programmer.

**a. Glosm Principle**

The principle of the glosm project is to generate 3D object from open street map data, which

are 2D data type.



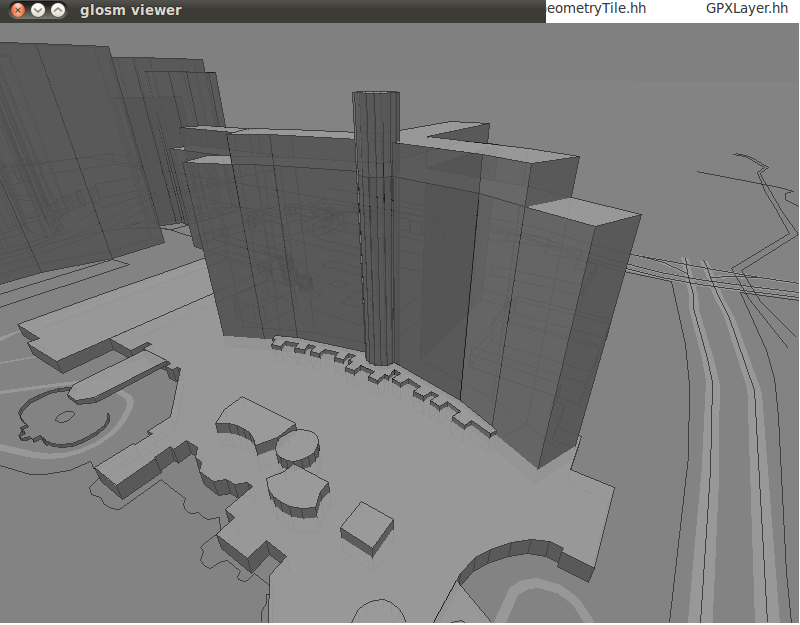


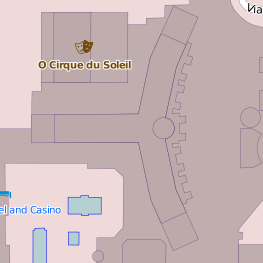
*Figure 3 : OSM Screen capture Figure 4 : Glosm Screen capture*

**Generation of the Buildings**

At now, Glosm manage the 3D shapes only for the buildings. It means that the ground elevation is not represented, so the Glosm world is flat. For the buildings, the OSM database contain some 3D information, even if they are inutile for the OSM visualiser, which is a 2D visualiser like Google Map. So, Glosm extract 3Ds datas from the .osm file, and build 3D shapes from those. But , those informations are very basic, like the heigh of the building, and the roof shape. It means that the shape of the generated buildings is totaly arbitrary for the programmer. In the Glosm case are square, and only the roof shape could change. So, the buildings are generated by putting together many square exactly like a LEGO game, were the 2D size (length and width ) of each square correspond of the foot print of the building.

*Example of a Glosm building generation (Bellagio Hotel, Las Vegas) :*





*Figure 5 : Bellagio hotel in OSM Figure 6 : Bellagio hotel in Glosm*

**Generation of the ways**

The ways is an important part for us, because it is the place where the car sould progress,

during the simulation. In the osm file system, the ways are declared as a jonction between

many nodes.

<way id="116614243" user="robgeb" uid="336460" visible="true" version="1" changeset="8361612"timestamp="2011-06-06T16:05:45Z">

<nd ref="1314230193"/>

<nd ref="1314230247"/>

<tag k="highway" v="footway"/>

</way>

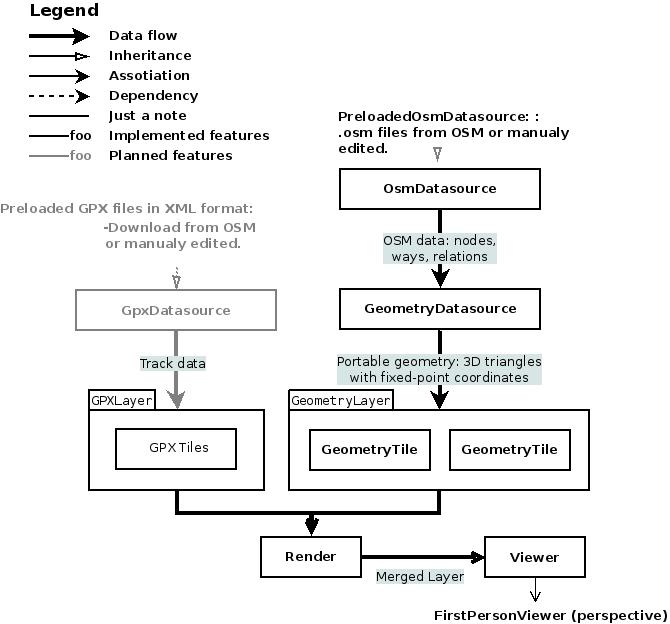
This example shown a way instantiation in the .osm file.

What is remarkable is that we could see that the tag type is highway tag k="highway"

, and it links two nodes : nd ref="1314230193" , nd ref="1314230247". All of the ways are instantiated like that, and it is even the same for all OSM entities. We do must not confuse the tag way with the declaration of ways. Actually, the way tag is use to declare all entities in OSM, so the way tag <way is used to declare the buildings, the roads, and all other types. This tag doesn’t provide any information about the instance type. In our example, the type declared in the line <tag k="highway" v="footway"/>, so the character chain to represent the way type name is "highway”. This type of instantiation implies that the ways cannot be curved. They are drawn as portions of segment.

**b. Glosm architecture**

Glosm work with different levels of graphics generation of OSM data. First, the .osm file is scanned by an XML parser and all the primitives (Nodes, Ways and relations) are extracted. Then, the GeometryData source processor transform the basics primitives into 3D triangles. Finally, the 3D triangles are associated in the GeometryLayer. The second layer is the GPX Layer, which represent the GPX traxes. This two layer are finaly gathered bu the Render.Finaly , the Viewer allow the user to observe the Render Data’s with a FirstPersonViewer mode.

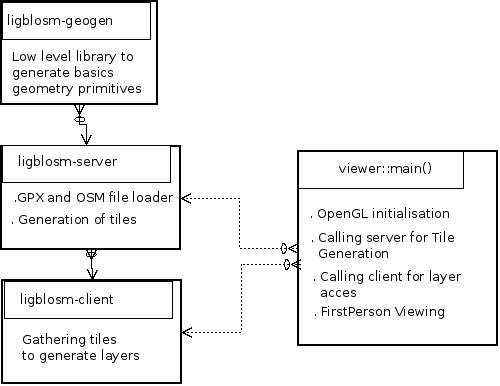


*Figure 7: Simplified glosm architecture*

(The complete glosm architecture is available in *annex 1.*)

**c. Implementation**

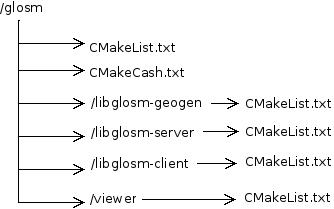
Glosm is implemented in 3 main libraries , libglosm-geomgen, libglosm-server, and libglosm-client.

**

*Figure 8 : Glosm Code implementation*

**Compilation system**

As, Glosm has been programed to be buildable in all the platform, and due to the quite important size of the project, Cmake is used to compile glosm.

****

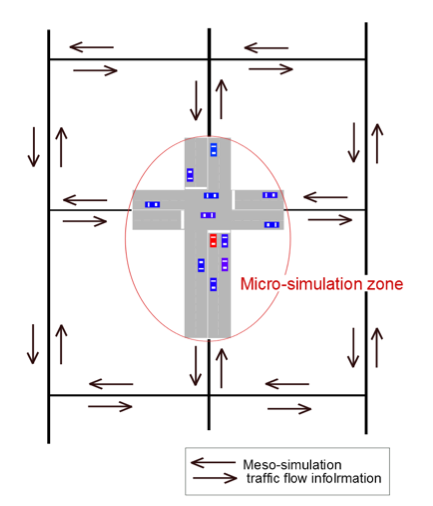
*Figure 9 : Glosm Compilation system*

For each directory which contains sources files, a CMakeLists must exist to list them. A CMakeList is also present under glom/ dir. It is the main CMakeLists, which links the low level CMakeLists, by listing all the concerned sub directory. Then, in our case, on Linux, it produces Makefiles in glosm directory and also for each sub directory.

5. Traffic and driving simulation: Models and definition

The final goal of this project is to obtain a both traffic and driving simulator. So, our choice was to consider the driving simulation feature as an extended possibility of the traffic simulation. Instead of following the traffic simulation car model, the player’s car is simply controlled by the player and provide a first person viewing. The first work was to define a way to build a both efficient an accurate traffic simulation framework. Many of them are already exists, and for all, the idea is quite the same : Separate the simulation in two layers, meso-simulation zone and micro simulation zone, to simply reduce the computing time when implementing the system on a machine.

**Micro simulation model**

****

**Micro simulation model**

First person viewing

****

*Figure 10 : Micro simulation model*

The meso-simulation layer provide global traffic flow information from the whole simulation area, whereas the micro-simulation deals which information about car presents in the user proximity area. The meso-simulation has not yet been treated during my internship and we chose to start to develop micro simulation. The micro simulation model have either not yet been implemented properly, but only the basic traffic simulation architecture necessary to implement the model in the future.

**The Car Following model**

The project was not as advanced to reach the Car Following model implementation step,

but I had to make research about it in the same time.

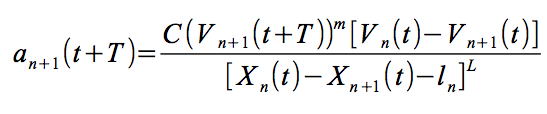
A common car following model is the MIT’s one, composed of 3 different regimes.

1. Car Following Model
2. Emergency Regime
3. Free flow regime

This model proposes 1 equation for each regime, which is the image of the acceleration in time.

**1 . Car following regime** [8]

When the speed of a car is limited by an other slower car ahead, the car following regime is active. The main equation of the car following is :





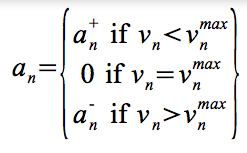
: Acceleration and speed of the following vehicle.

 : Gap between the two vehicles

C ,mandn**:** Constants to calibrate, and must be different for acceleration and deceleration.

**2 . Free flow regime**

When there is no car ahead or if the gap is important enough according to a time interval or/and a distance criteria. This is the only regime taking the driver’s wanted speed into account : So if a car desired speed is lower than the leading car’s actual speed, or if the car is above its desired speed, we might also switch to this regime. Here is its equation:



: Current Acceleration

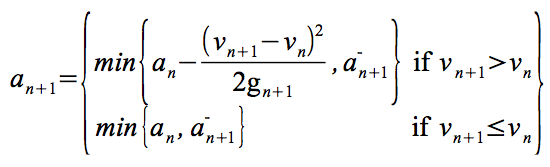


 : Current Speed

: Maximal acceleration and desired speed

**3 . Emergency regime**

When two cars are too close to each other according to a distance and/or a time interval criteria.

****

**Other models to implement in a further development** [8]

To build a complete basic driving simulator, we also need to determine two other main models under the micro simulation: The *changing lane model* and the *gap acceptance* model (the car following model don‘t treat the gap between car, but only the acceleration).

6. Traffic and driving simulation: Implementation on glosm

At this point, I had all the sources present on my computer and all the compilation required library.

I spent a important time to understand many important details of the glosm code before being able to build on it. The final goal of this project is to obtain a both traffic and driving simulator, so, our choice was to consider the driving simulation feature as an extended possibility of the traffic simulation. Instead of following the traffic simulation car behavior model, the player’s car is simply controlled by the player and provides a first person viewing.

**Note about Development organization and objectives**

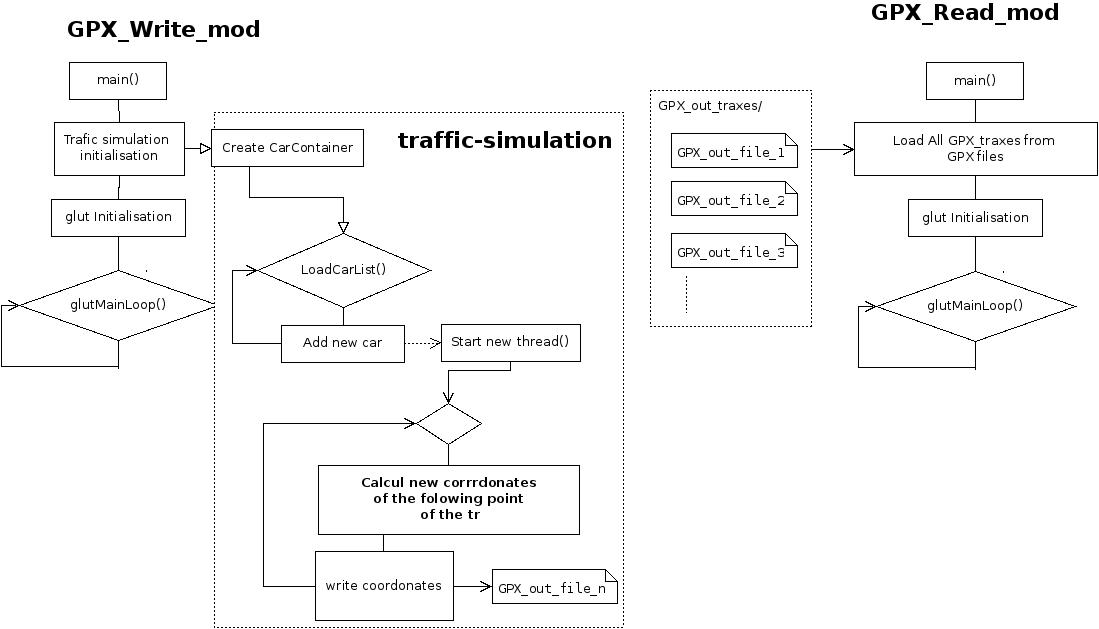
We chose to adopt an incremental development. The first increment objective was to build the architecture of basics classes required for the traffic-simulation, especially Car, and Car-container classes. The second increment was to provide to the Car class, all the functionality required for position and motion management in the OSM map type. We call it *navigation handler* in this report. The driving simulation feature has not been treated during the internship. I ended my internship during the second increment, but the UML diagram of increment 3 has already been defined, and we present it at the end of this report, in the *further development* part.

**Global principle of traffic-simulation implementation**

As we implement our code under glosm, we are fully dependent of the glosm framework.

An important consideration we have to keep in mind is that glosm used the OpenGL library

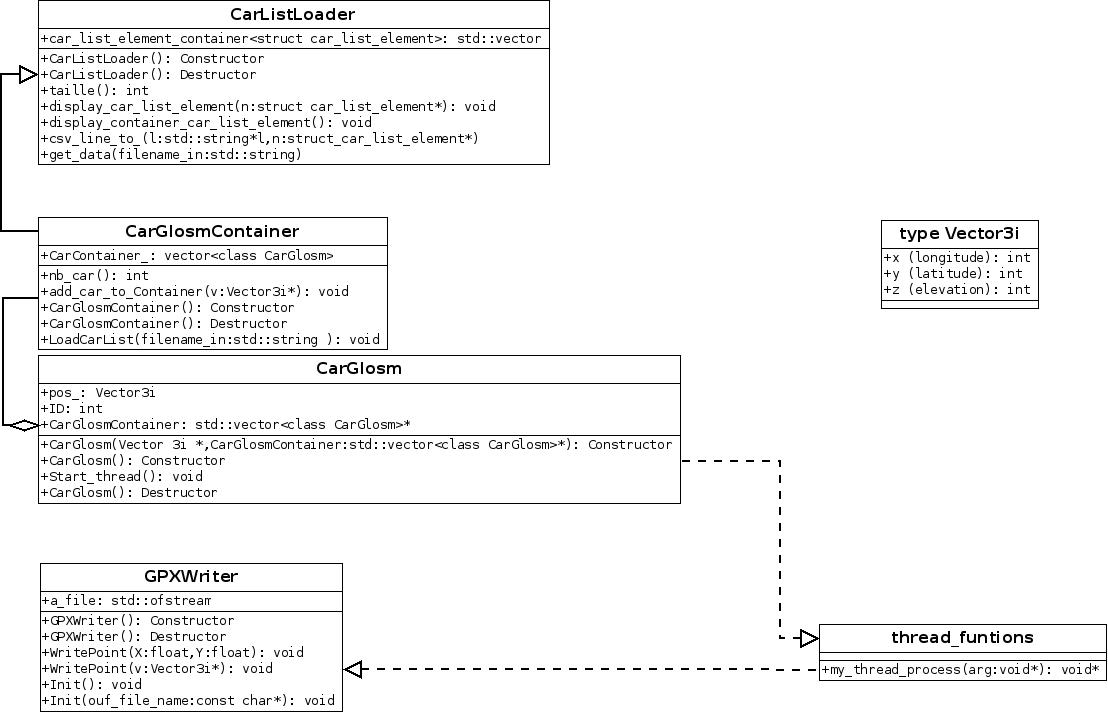
for 3D graphics rendering. OpenGL functions are used through the *glut* API. The way of programming and using glut is a little bit problematic for us especialy because it runs under a timer frequency to update the graphics, and for many others technical reasons whishes are not detailed in this report. So, we needed to build a code in the most possible glosm’project independent way, to be independent of OpenGL.The best solution to be independent of the library is to use detached thread to run our simulation in separate process. Our thread function is the function, which is executed when calling a new thread process (refer to *annex 2* for more information about Linux thread details). This function must implement the whole Car Behavior handling, by using the *Car Navigation Handler* Class resources (increment 2). The output of one thread functions is the trajectory of the Car (one thread for one Car). More precisely, this Car thread function produces a vector container with contains all the points coordinate of the whole trajectory during the traffic-simulation. So, in the same time, the goal of the increment 1 was to implement in itself a testing method to validate the advancement at each step. So, to validate the trajectory of each Car (produced by each thread), we chose to use a functionality already implemented in glosm : The *GPX taxes rendering* . Glosm allow a GPX traxes rendering in the viewer mode (first person viewing). We simply chose to use it to verify the trajectory of each car (increment 2), by writing points , whishes are images of the trajectory. To sum up , we implemented a CarContainer class which contains Car class (named CarGlosm), in which there is a dethatched thread method, which handle the Car behavior. In the future , all the new Car Behavior models have to be handled in this thread method.



*Figure 11 : The two modes for developing and testing the traffic simulation under glosm*

This diagram represents the system we obtained on the increment 1.It gives an idea of the development philosophy, by describing the algorithm of the two modes. The simulation is launched in the write mode, while the read mode is just the original glosm viewing mode, which allows us to observe directly on the map, the GPX traxes resulting of the simulation.

**a. Increment 1 : Basic classes**



*Figure 12 : UML diagram : increment 1*

Each car is represented by the CarGlosm class, and an now there are two main attributes, one to represent the 2D potition (pos\_) and one for identification (ID). The CarGlosmContainter class is used to store the car and inherits CarListsLoader. CarListsLoader is a class for parsing an input CSV\* format file, which lists each car of the simulation. Each listed cars must be written like this :

*ID,"car","latitude","longitude"* ***example :*** *1,"car","-29.53850173950195","53.89432144165039"*

All the thread function are called by each CarGlosm object on the CarGlosmContainer and has access at all the atrributes of the CarGlosm object. CarGlosm Class also has a CarGlosmContainer pointer as attribute. This pointer provides a full CarContainer access to the CarGlosm class. This access is absolutly required because each Car must know the position of all other cars during the simulation.

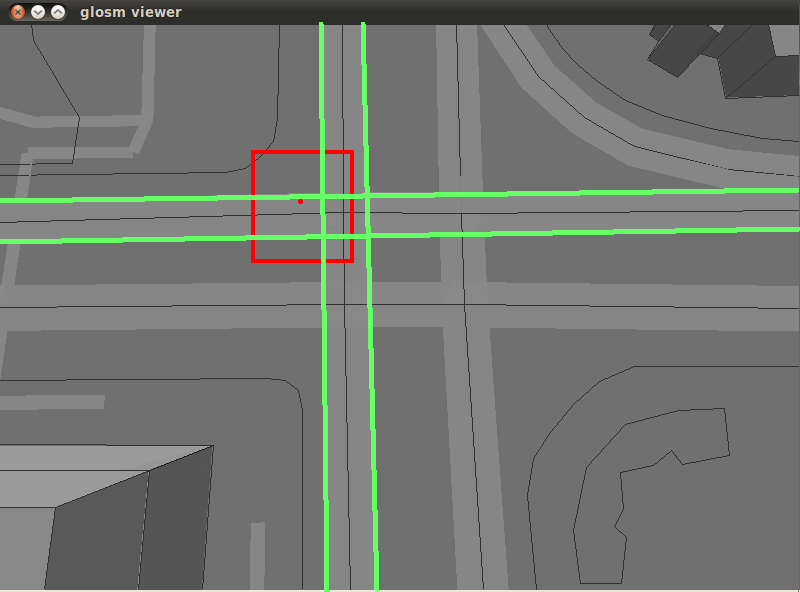
The GPX Writer class is used to create an XML file which reference the coordinates of the car’s trajectory during the simulation. As we have seen previously, this class is used only in the simulation mode, called the GPX\_Write\_mode.

**b. Increment 2 : Car Navigation handler**

**Point-on-a-way check**

This part is more difficult one, mainly because it deals with the OSM data, which is not an adapted for what we need. The principal difficulty was to manage the position of the car each car in order to place it and maintain it on a way. We will call it ***point-on-a-way check***. As it is explained in the *OSM presentation* part, OSM data’s are organized by couples of nodes and way. This level of description is very low, and it implies an important amount of data and an important computing time to deal with. There are few different way to implement a ***point-on-a-way check***. The most obvious one is to iterate all nodes, then for each node, iterate all nodes again, then for pair of nodes find a way which goes between them and finally check if our point lane on this way. This is also obvious that this solution implies an horrific complexity: o(N.N.W.n), where N is number of nodes, W is number of ways and n, the number of nodes in a way. A better solution is to directly iterate through all ways and for each way segment, check whether the point in question lies on it. The complexity will be o(W.n) because getting a node by it’s ID is constant, as *hash maps* are used for these in glosm. In addition, Glosm already implements a very interesting tool for us, the **Boundary Box (BBox),** which allows a filtering of way for a determined square shaped area.

As the BBox returns all the way intersected by the square shaped area (important to know a way represent each object in OSM and not just roads), we must make a second filtering to extract the type *highway* , which represent the real way (eq roads). The complexity of one filtering operation with a BBox is constant, and so, if we takes our second filtering in consideration, we get a complexity of o(W), with W is now the number of way extracted from the BBox. The example below illustrates the BBox detection of two ways around a point.

****

*Figure 13 :*

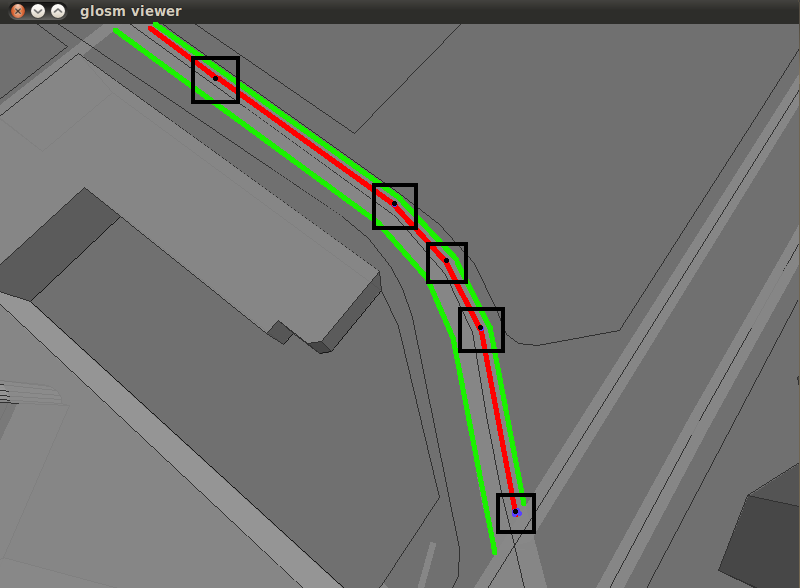
*Highways intersected by a BBox*

**Trajectory builder**

Once our Point-on-a-way check principle is validated. We have to implement a way to guide the car

while navigating. The idea was to obtain a function, which provides us a complete trajectory of many points, by passing the origin and the destination points coordinates. In fact, the OSM highway (roads) are represented by portions of many segment, because the curved highway doesn’t exists in OSM.

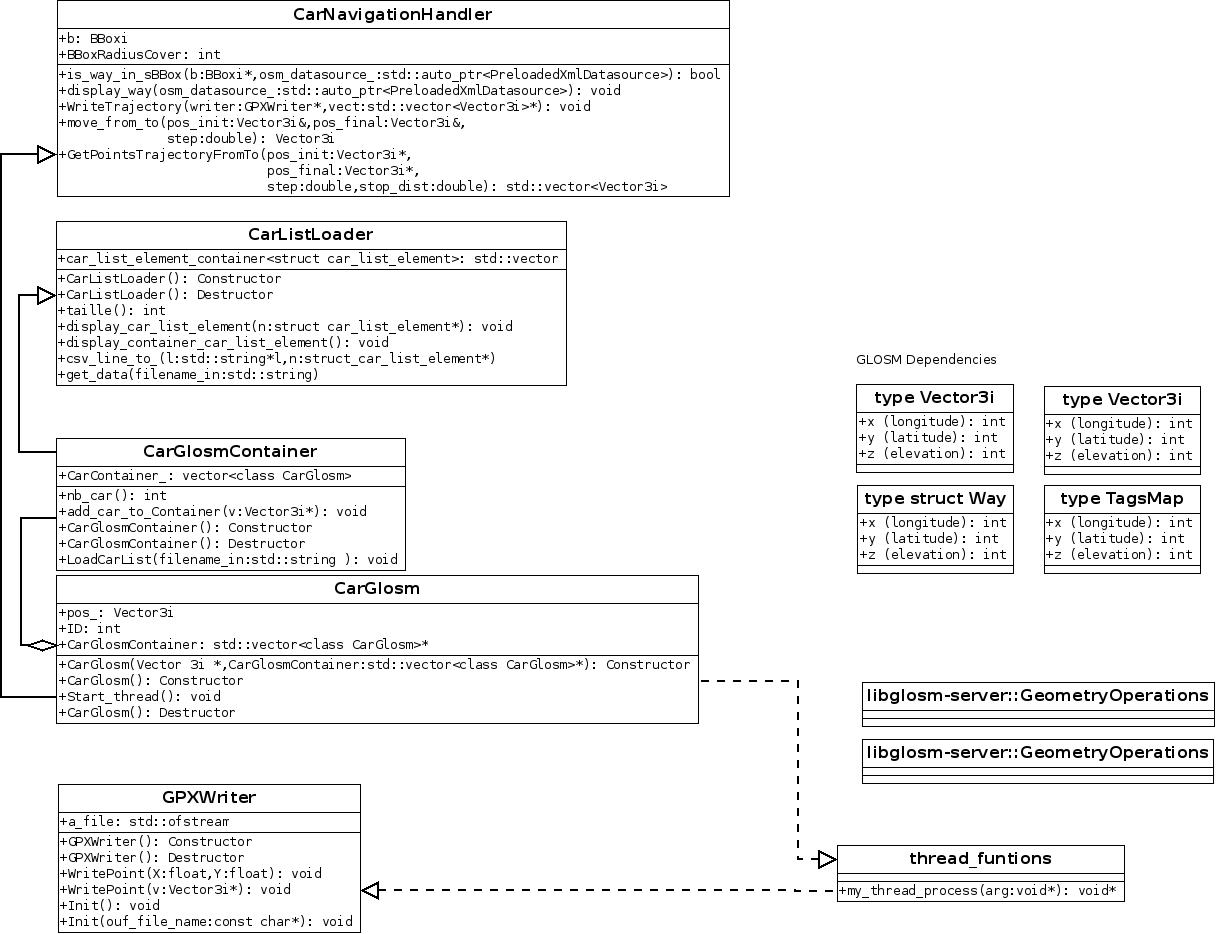
So, by combining the Point-on-a-way check and the Trajectory builder, it is possible to create a complete highway trajectory, by re-build a new trajectory at each new segment of the highway. The point Point-on-a-way will be used to obtain the two vertex points of each highway segment ant then, the trajectory builder will provide the trajectory points between them , with a determined step.

****

*Figure 14 : Illustration trajectory builder*

In the figure above, you can see an illustrated example of a complete trajectory (in red), composed of several small trajectory of each segment of the curved highway detected by the BBox. Note that the BBox and trajectory builder calling occurs at each segment endings.

**Increment 2 : Class diagram with CarNavigationHandler**



*Figure 15 : UML diagram : Increment 2*

The *point-on-a-way-check* and the *trajectory builder* are implemented in the *CarNavigationHandler* class. To access these functions, *CarGlosm* inheritates *CarNavigationHandler.*

It is important to know that the trajectory builder uses the Vector3i types, which is a structure containing 3 32bit fixed point variable type, to represent each component (x,y,z).

At now, there are two main functions: *GetPointsTrajectoryFromTo* returns a vector of Vector3i type, which contains all the point composing the trajectory between the two point passed in parameters, with a determined step passed in parameter to. *WriteTrajectory* is used to write all points of the vector returned by *GetPointsTrajectoryFromTo* in the GPX file. For that his function uses the GPXWriter class features. There is an example of a builded trajectory in *annex 2.*

**Note about the used variable type**

I have first started by using float type for Navigation operations, but I rapidly encountered a problem of precision. In fact in OSM, the conversion between int and float to represent a position are : *float float\_coord = int\_coord / 10000000.0f;*

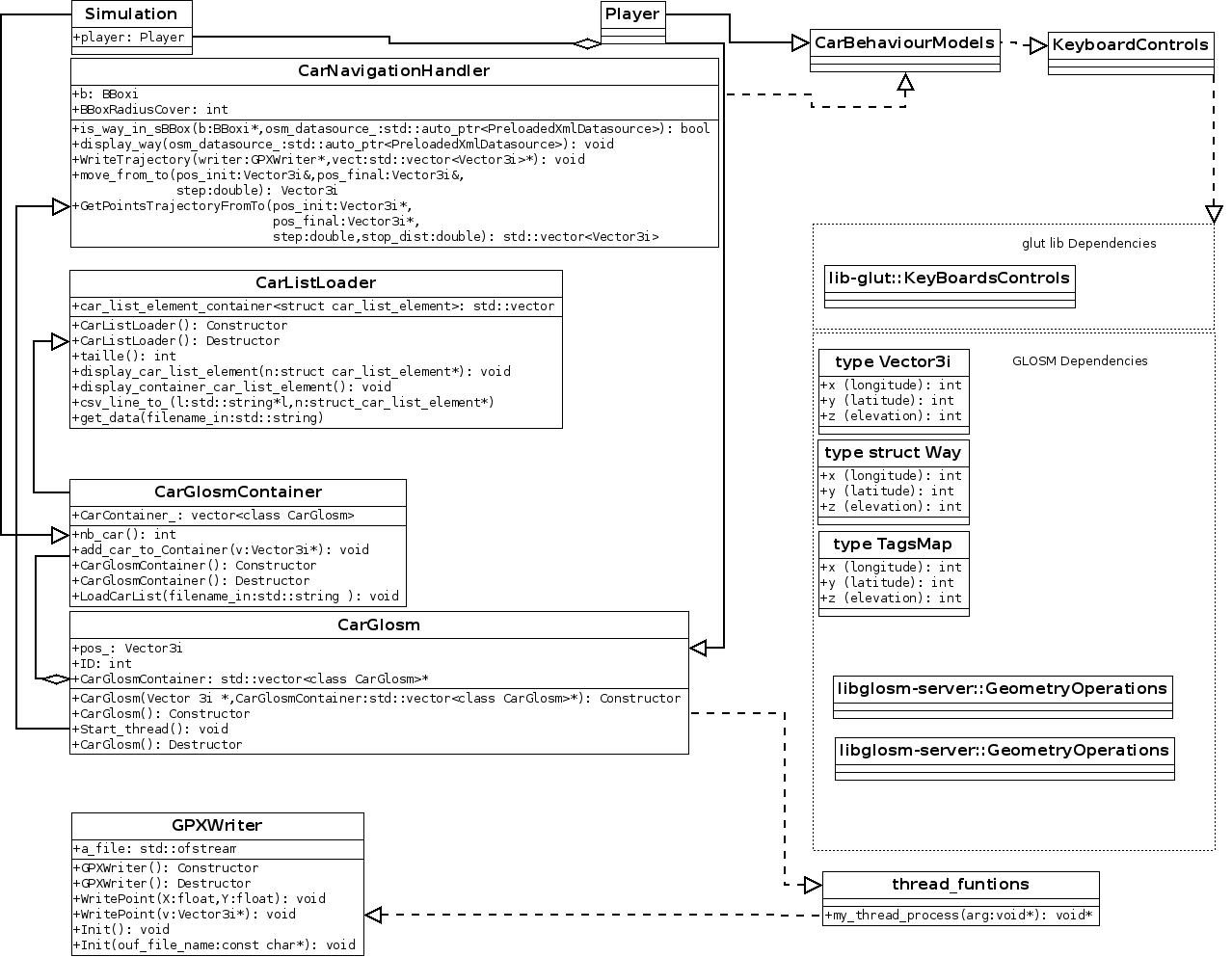
*int int\_coord = (int)(float\_coord \* 10000000.0f);*

so, 12.34 would be 123400000 in osmint\_t (signed int), and one meter is

approximately equal to 0,00001 in float representation.

It implies that it is risked to make geometric operations in a meter range with a float, due to the limited size of 23 bit for mantissa. It is for this reason that I choose to use the Vector3i ty In Vector3i, one meter is equal to 152 , so there is no more problem of precision with operations in meter range. A problem of precision have been found if the step passed to *GetPointsTrajectoryFromTo* is less that one meter, but this problem have not been fixed yet. But, a step of one meter should be widely sufficient to observe a trajectory with precision.

**c. Further development: Advanced architecture: Adding Player Hander**



*Figure 16 : UML diagram, further development*

This UML diagram has not been implemented during my internship, but it will be useful in a further development and it was a part of my work to define it. It just provides a way of implementation for 3 new class, the Simulation class, which should be the top level class of the traffic-simulation, the player class, which inherits of the Car Glosm class and Keyboard Controls. The KeyboardControls class will be the first fully OpenGL dependent class. In fact, the glut OpenGL’s API already provides a full KeyboardControls interface, which is already used in glosm to handle the FirstPersonViewer motion. So, the idea is to re use this glut-KeyBoardControls to implement our Car-Player-KeyBoardControls in our traffic-simulation.

Another class CarBehaviourModel inherits KeyBoardsContols class and is implemented by the CarNavigationHandlerClass. The idea is the the CarNavigationHandlerClass woks like a filer with Keyboards signals in input and motions information in outputs. These motions information will be finally interpreted by the CarNavigationHandler to build a trajectory. Note that this motion information in input should be only an acceleration variable, and not a direction. The direction should be calculated directly by the CarNavigationHandler because the directions determination implies a knowledge’s of the highway’s segment orientation on the map, and this is implemented only CarNaviGationHandler.

7. Conclusion

In conclusion, I m satisfied of this internship. An appreciated point for me is that I was not limited for my work, and I had to make many choices for myself like choosing the starting point project and define the starting point architecture. But, this quite important liberty of development was a difficulty in the same time. In fact, I tried to do my best to define logic and clear way of development, in order to don’t fall into a technical impasse. A second difficulty is that I had to learn languages and tools in the same time than developing. I spend an important time to find and solve simple language syntax errors, which has reduced my coding efficacy. An interesting component of this internship was the fact of working in the conditions of a real C ++ Open Source Project like glosm, without any SDK, with all files manually edited on Linux.

8. Bibliography and useful sources

[1] www.wiki.openstreetmap.org

[2] www.wikipedia.org

[3] www.cplusplus.com

[4] [www.git-scm.com](http://www.git-scm.com)

[5] [www.doxygen.org](http://www.doxygen.org)

[6] [www.cmake.org](http://www.cmake.org)

[7] [www.opengl.org](http://www.opengl.org)

[8] Maxime Vey Internship report : Micro-Simulation Model and Database Management

[9] Pierre Ficheux Website , Linux Multi Threading : [www.pficheux.free.fr](http://www.pficheux.free.fr) [10] Nicolas Roussel, LRI, INRIA : Introduction à OpenGl et Glut [11] Nate Robins : OpenGL turorial : user.xmission.com/~nate/tutors.html

9. Glossary

*GPX traxes :* Name used to describe a simple point on a map, like a GPS position capture.

Meso-simulation : High level of simulation. Deals with global traffic information.

*Micro-simulation :* Low level of simulation. Deals with discrete objects (cars in our case).

*OSM :* Open street Map

*CSV :* Coma separated value format

*XML :* Extensible Markup Language (used for GPX traxes file in glosm)

*KeyboardControls* : Controls from Keyboard

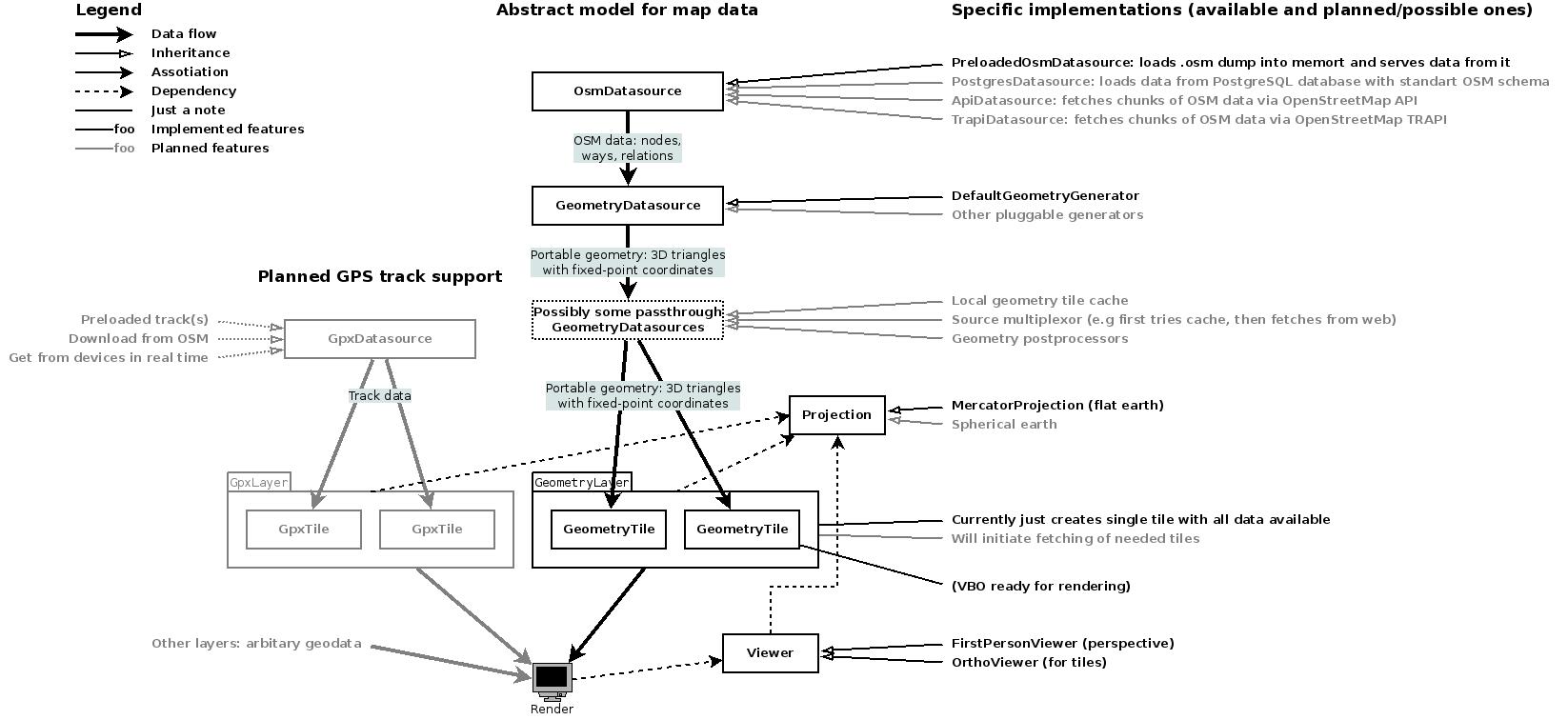
*Way :* In OSM , way is a tag to represent each object like highway, building , tree…

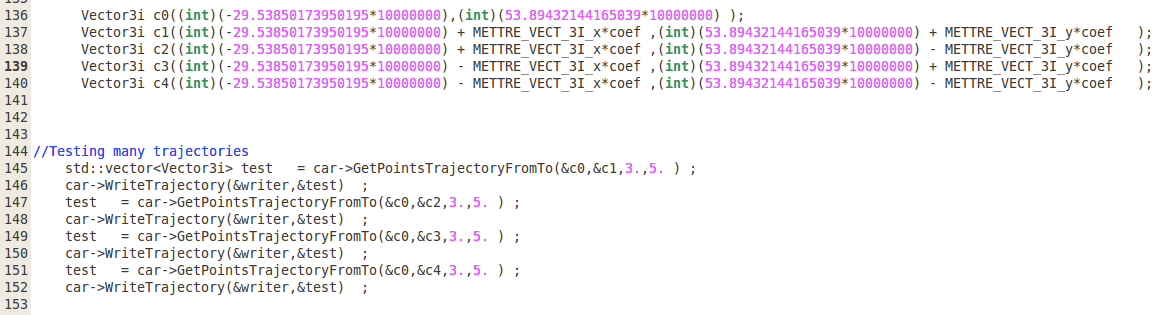
*BBox: (Boundary Box)* A common tool in OSM and glosm, which provides many features to manage map’s data, from a selected square shaped area.

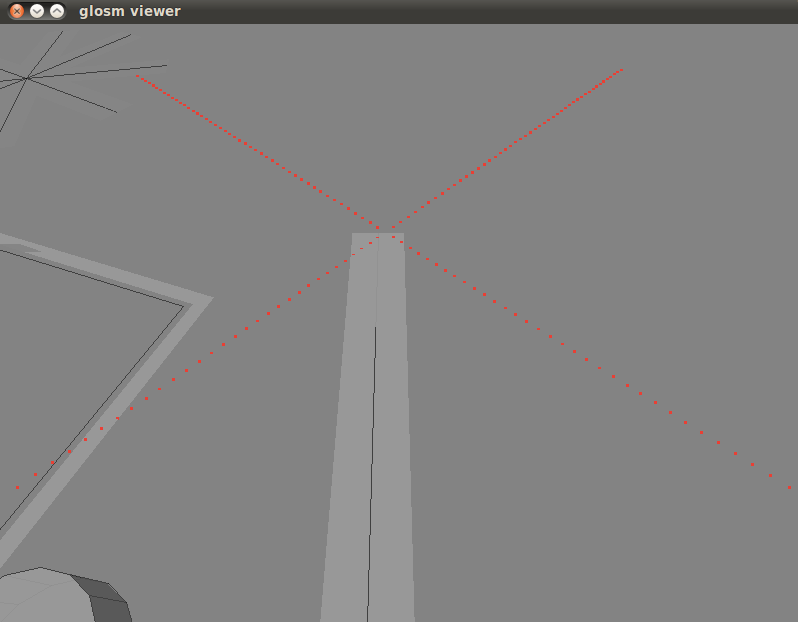
*FirstPersonViewer* : A viewing mode to simulate the user point of view, like in reality.

Appendix

***Annex 1 : Complete glosm architecture by* Dmitry Marakasof :**



***Annex 2 : Example of a trajectory builded by*** *GetPointsTrajectoryFromTo* ***(increment 2) :***



In this example, we first define 5 points c0,c1…,c4, and we succevely computes and write in GPX file the four trajectory : c0 to c1, c0 to c2 ,…, c0 to c4. Note that for each trajectory, we choose a step of 3 meter (distance between each points) and a stop distance of 5 metter.