Towards a realistic traffic and driving simulation using 3D rendering

Conference Paper · September 2015

DOI: 10.1109/ICCP.2015.7312683

Alexander Paz
University of Nevada, Las Vegas
76 PUBLICATIONS **528** CITATIONS
SEE PROFILE

Naveen Veeramisti
University of Nevada, Las Vegas
10 PUBLICATIONS **22** CITATIONS
SEE PROFILE

Hanns de la Fuente-Mella
Pontificia Universidad Católica de Valparaíso
64 PUBLICATIONS **173** CITATIONS
SEE PROFILE

Luiza Modorcea
University of Nevada, Las Vegas
2 PUBLICATIONS **4** CITATIONS
SEE PROFILE

Towards a realistic traffic and driving simulation using 3D rendering

Alexander Paz, Ph.D., P.E.
Civil and Environmental Engineering and Construction
University of Nevada
Las Vegas-Nevada, USA
apaz@unlv.edu

Naveen Veeramisti, M.S., E.T.
Civil and Environmental Engineering and Construction
University of Nevada
Las Vegas-Nevada, USA
naveen.veeramisti@unlv.edu

Hanns de la Fuente-Mella, Dr.
Facultad de Ciencias Económicas y Administrativas
Pontificia Universidad Católica de Valparaíso
Valparaíso, Chile
hanns.delafuente@ucv.cl

Luiza Vasilica Modorcea
Faculty of Law
Transilvania University
Brasov, Romania
luizavmodorcea@gmail.com

Heather Monteiro
Research Analyst
University of Nevada, Las Vegas
College of Hotel Administration
heather.monteiro@unlv.edu
Las Vegas-Nevada, USA

***Abstract*— The objective of this research is to provide a deployable driving simulation framework with the focus on increasing modeling realism. A traffic and driving simulator concept was developed using Open Street Map (OSM) for the three-dimensional (3D) generation of the corresponding visualization module. The proposed framework includes Glosm, a hardware-accelerated OpenGL based on OSM. Glosm provides stable 3D generation of a virtual vehicular urban system using OSM data and a real-time first-person viewer. First, a 3D car model was implemented in Glosm. Then, a driving and traffic simulator (without graphics) was developed including all the required functions for representation and motion handling within Glosm. Initially, a way of testing was defined and implemented for all future development of the navigation handler. The goal was to determine the configuration of the OSM-type road network from a given position in order to compute the navigation trajectories. The data system of OSM was highly unsuitable for this application because the roads are included in the same layer as other objects. Unity is an alternative to Glosm. Some information about Unity is provided with recommendations for future research.**

Keywords*— *Driving Simulator; 3D Reconstruction; GLOSM; Driving Assistance Applications

I. INTRODUCTION

Driving simulators provide a mechanism to study actual driver behavior in a controlled and safe environment [1] – [5]. Driving simulators have been developed since the early 1920s [6] by many companies including Daimler-Benz [7], which developed a high-fidelity driving simulator. Research institutions and commercial companies have developed both fixed-based and advanced motion-based driving simulators. Some of the recent developments include: (i) the National Advanced Driving Simulator (NADS) [8]; (ii) the Driving Environment Simulator (DES) [9]; (iii) the VTI Simulator IV [10]; (iv) the driving simulator at the University of Leeds [11]; (v) the DriveSafety driving simulator [12]; (vi) the STISIM Drive™ driving simulator [13]; and (vii) the UC-win/Road driving simulator [14].

The NADS Laboratory [8] includes an advanced motion-based ground vehicle simulator (NADS-1), a fixed-based vehicle simulator (NADS-2), and a PC-based high-performance driving simulator that uses the same technology as NADS-1 (MiniSim™).

The Driving Environment Simulator (DES) [9] is an immersive driving simulator that provides high-fidelity simulation to generate a realistic presence within the simulated environment. DES measures psycho-physiological responses such as eye responses and brain activity such as Evoked Response Potential (ERP).

UCwin/Road [14] provides a three-dimensional Virtual Reality (VR) environment and includes traffic simulation and visualization tools. It uses ground texture maps and includes 3D building images. The environment provides traffic-generation models to generate traffic on various lanes and roadways.

The driving simulator proposed here involves 3D rendering of the complete real-time traffic simulation, for which one player takes control of one car in a first-person viewing mode. This simulator uses Open Street Map for map generation. The corresponding software is being implemented using a framework previously presented by the authors [15].

During the first phase of this research, a basic rendering framework was implemented with BLENDER™, a free, open-source 3D graphics software. This rendering framework was merged with a microscopic traffic flow simulation core model. The second phase was to build traffic and driving simulations. This paper presents several concepts regarding global issues in traffic simulation and describes a proposed model.

II. Methodology

Open Street Map (OSM) was a collaborative project to create a free editable map of the world. Created by Steve Coast in the United Kingdom in 2004 [16], it was inspired by the success of Wikipedia® as well as the preponderance of proprietary map data in the UK and elsewhere.

A realistic driving simulation environment requires a 3D map to be generated from data such as OSM in contrast to an artificial rendering as is the case in video games. There are a number of alternatives to generate this 3D virtual environment using OSM. The two most promising alternatives are Glosm and Unity. Other alternatives such as ShiVa, Unreal Engine and Torque 3D are not as mature as Glosm or Unity. For example, Shiva requires programming languages that are less common than Unity, Unreal Engine was primarily developed for very high quality games, and Torque 3D is just for web-based applications. Another alternative is to use an integrated development environment (IDE) such as Eclipse, Visual Studio, or MonoDevelop. However, Unity is an extension of an IDE [17]. That is, software components can be developed using an IDE and compiled using Unity. This paper focuses on the use of Glosm. Future work will explore the use of Unity relative to an implementation with Glosm.

Glosm derives this information from OSM. OSM data (*.osm*) are available in XML format and consist of tags, elements, and attributes. The data can describe all elements in a hierarchical way, including many attribute types such as strings and float. The tag was used to instantiate an object: each tag represents an element and each element contains attributes.

In addition, it is possible to export the OSM map of any location directly from the website; however, the size is limited in terms of the number of nodes. Many Internet sites exist that make large maps available and it is therefore possible to find complete *.osm* files for each major city, referencing the whole planet in planet.osm.org. However, the file size is limited to 320 GB uncompressed. Glosm provides a stable 3D generation of OSM data and a real-time first-person viewer.

A. Glosm Principle and Architecture

The objective of the Glosm project is to generate 3D objects from Open Street Map data, which is a 2D data type. Currently, Glosm only manages 3D shapes for buildings; this means that the ground elevation is not represented so the Glosm 'world' is flat. For buildings, the OSM database contains some 3D information that may be unusable for a 2D OSM visualizer, such as Google Map.

Glosm extracts the 3D data from the .osm file to build 3D shapes (see Figure 1). However, that information provided by the .osm file is very basic, such as the height of a building and the shape of a roof. This means that the shape of the generated image may be arbitrary or inaccurate. In Glosm, buildings are represented by square shapes and only roof shapes can be changed. Buildings are generated by assembling many squares, similar to a LEGO® game. The 2D size (length and width) of each square corresponds to the footprint of the building. Figure 2 provides an example of Glosm building generation for the Bellagio Hotel and Casino in Las Vegas, Nevada.

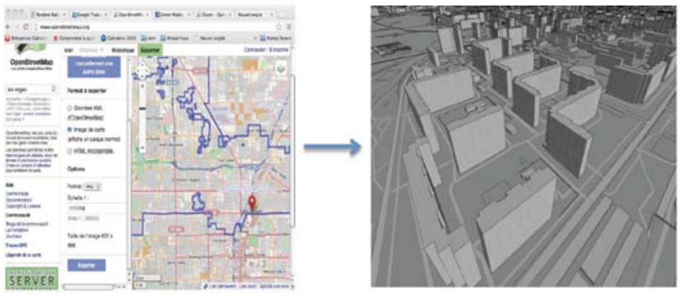

Fig. 1. OSM screen capture converted to a Glosm screen capture.

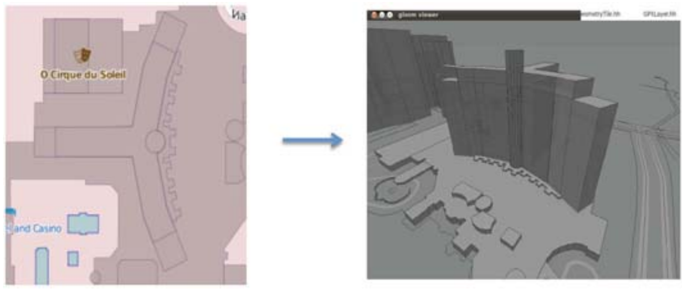

Fig. 2. Example of Glosm building generation of the Bellagio Hotel and Casino in Las Vegas, Nevada.

Roads are an important part of this research. In an OSM file system, roads are generated using nodes, junctions between nodes, and their connections. Figure 3 shows a road of instantiation in the .osm file.

As shown in Figure 3, the most important feature is the tag number, for instance, highway tag k="highway"; this links two nodes: nd ref="1314230193" and ndref="1314230247". All of the roads are instantiated in that manner for all OSM entities.

```
<way id="116614243" user="robgeb" uid="336460" visible="true" version="1" cha
ngeset="8361612"timestamp="2011-06-06T16:05:45Z">
<nd ref="1314230193"/>
<nd ref="1314230247"/>
<tag k="highway" v="footway"/>
</way>
```

Fig. 3. An example of road generation in an OSM file system.

The terms ‘way’ and ‘road’ have the same meaning in English; however, it is necessary to be specific to avoid confusion. In this use, the ‘way’ tag is used to declare all entities in OSM; thus, the ‘way’ tag is used to declare buildings, roads, and all other entities. This tag does not provide any information about the instance type. In Figure 3, the type declared in the line <tag k="highway" v="footway"/>, is the character chain to represent the road type name, ‘highway’. This type of instantiation implies that the roads cannot be curved. They are drawn as portions of a segment.

Glosm works with different levels of graphics generated from OSM data. First, an XML parser scans the .osm file and all the primitives (node and way) are extracted. Second, the GeometryData source processor transforms the basic primitives into 3D triangles. Third, the 3D triangles are associated in the GeometryLayer processor. The second layer is the GPX Layer, which represents the GPX tracks. These two layers are gathered by the Render. Finally, the Viewer allows the user to observe the Render Data by using a FirstPersonViewer mode. The Glosm architecture is shown in Figure 4.

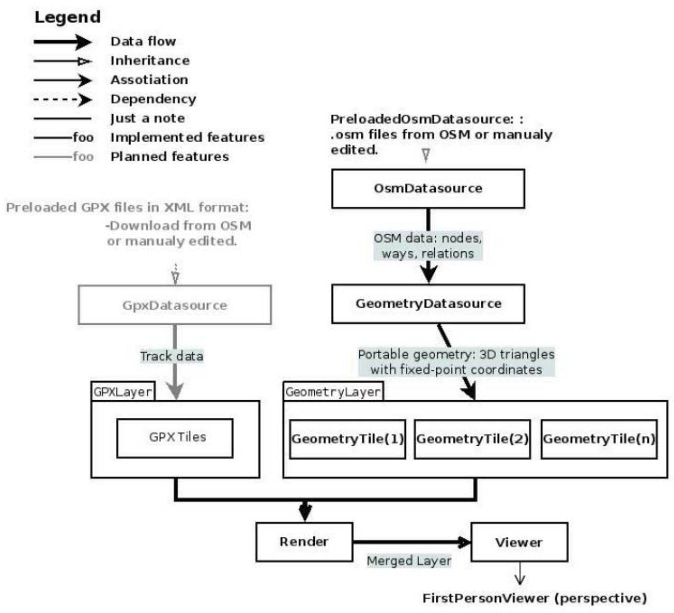
Legend
Data flow
Inheritance
Assotiation
Dependency
Just a note
foo Implemented features
foo Planned features
PreloadedOsmDatasource: : .osm files from OSM or manualy edited.
OsmDatasource
OSM data: nodes, ways, relations
GeometryDatasource
Portable geometry: 3D triangles with fixed-point coordinates
GeometryLayer
GeometryTile(1)
GeometryTile(2)
GeometryTile(n)
Preloaded GPX files in XML format: -Download from OSM or manualy edited.
GpxDatasource
Track data
GPXLayer
GPX Tiles
Render
Merged Layer
Viewer
FirstPersonViewer (perspective)

Fig 4. Glosm architecture.

B. *Traffic and Driving Simulation: Implementation on Glosm*

The objective of this research was to obtain a realistic environment in a traffic and driving simulator. The driving simulation was considered an extended capability of the traffic simulation. Instead of following the traffic simulation model, the driver’s car was controlled by the driver and provided first-person viewing (see Figure 5). Thus, the first task was to define a plan to build both an efficient and an accurate traffic-simulation framework. Therefore, the simulation was separated into two layers: a meso-simulation zone and micro-simulation zone, to reduce the computing time when implementing the system on a machine. The meso-simulation layer provides global traffic-flow information from the entire simulation area, and the micro-simulation layer deals with information about cars present in the user’s proximity.

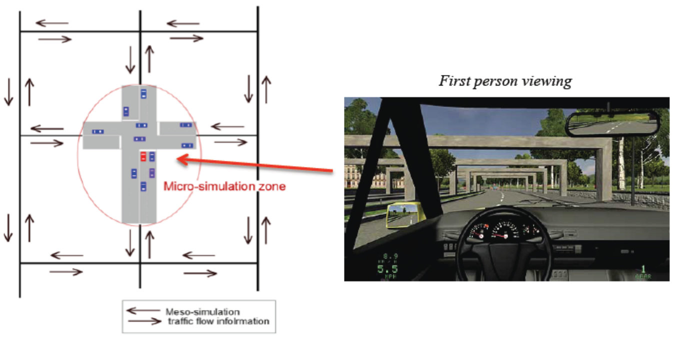

Fig. 5. Micro-simulation model.

Incremental development was adopted when building this model. The first increment involved building an architecture of the basic classes required for traffic-simulation, especially car and car-container classes. The second increment provided the car class, with all the functionality required for position and motion management in the OSM map type; in this study this was called the ‘navigation handler’.

Since the code was completely dependent on the Glosm framework, the Glosm used the OpenGL library for rendering 3D graphics. OpenGL functions were used through the application programming interface (API), OpenGL Utility Toolkit (GLUT). The programming and the use of GLUT was problematic for this research because it ran under a timer frequency to update the graphics. Therefore, the code must be built in the most independent manner possible for Glosm to be independent of OpenGL. The best solution to run independently of Glosm was to use a detached thread [14] to run the simulation in a separate process.

The thread function must be executed when calling a new thread. This function must implement the entire Car Behavior handling by using resources of the Car Navigation Handler Class (Increment 2). The output of one thread function was the trajectory of a car (e.g., one thread for one car); more precisely, this car thread function produced a vector container with all the point coordinates of the entire trajectory during the traffic simulation.

At the same time, the goal of Increment 1 was to implement a testing method to validate advancement at each step. In order to validate the trajectory of each car (produced by each thread), a functionality was chosen that already was implemented in Glosm: GPX track rendering. Glosm allows rendering GPX tracks in the viewer mode through first person viewing. In this model, it was used to directly verify the trajectory of each car (Increment 2) in the viewer mode. In total, a Car Container class was implemented that contained car objects (CarGlosm class) in which an attached thread handled the car's behavior. In the future, all the new Car Behavior models will be handled using this thread method (see Figure 6).

Figure 7 represents the system developed for Increment 1. This figure gives an idea of the development approach by describing the algorithm of the two modes depicted in Figure 6. The simulation was launched in the GPX_Write_mod. The GPX_Read_mod was the original Glosm viewing mode, which allowed direct observation of the map. GPX track rendering resulted in a simulation in which each car was represented by the CarGlosm class with two main attributes: one to represent the 2D position (pos_) and one for identification (ID). The CarGlosmContainter class was used to store the car and inherited CarListsLoader, which is a class for parsing an input CSV format file that lists each car in the simulation. Each listed car must be written as follows:

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

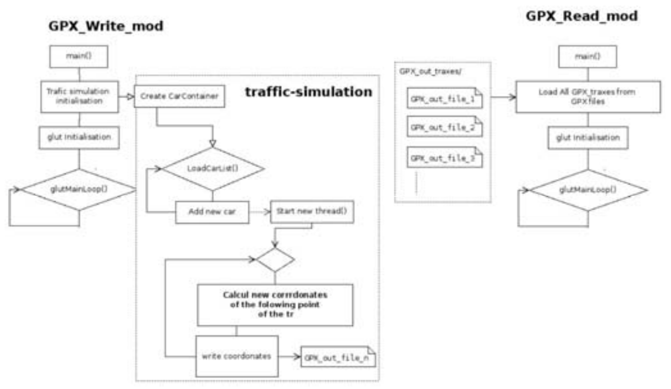

Fig. 6. Two modes for developing and testing the traffic simulation under Glosm.

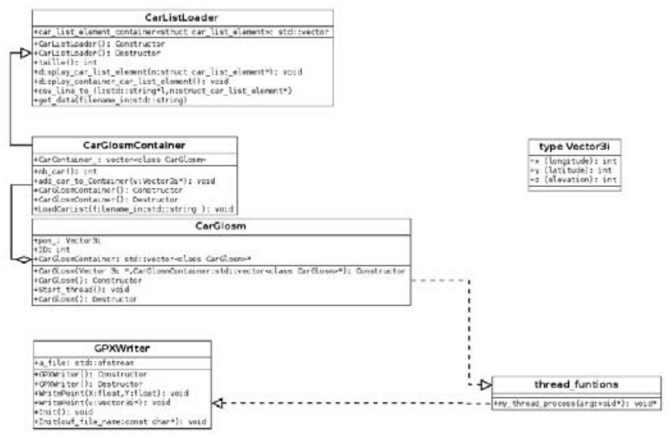

Fig. 7. UML diagram of Increment 1.

All the thread functions were called by each CarGlosm object on the CarGlosmContainer, and had access to all the attributes of the CarGlosm object. In addition, the CarGlosm class had a CarGlosmContainer pointer as an attribute. This pointer provided full CarContainer access to the CarGlosm class. This access was necessary because each car needed to know the position of all other cars during the simulation. The GPXWriter class was used to create an XML file, which referenced the coordinates of the car's trajectory during the simulation. As seen earlier, this class was used only in the simulation mode (GPX_Write_mod).

For Increment 2 (Car Navigation Handler), a system was needed that allowed the extraction of information about the configuration of the nearest road network for a given position. This system, called 'point-on-a-way check', would be useful to compute trajectories and check if a point lies on a way. To implement this point-on-a-way check, each node was iterated twice. Then, for each pair of nodes, a way that goes between them was determined, and finally each point was confirmed to lie on this way.

This method implies a complexity, o($N.N.W.n$), where N is the number of nodes, W is the number of ways, and n is the average number of nodes in a way. A more optimal solution is to iterate directly 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 its ID is constant since hash maps are used for these in Glosm.

However, Glosm already implemented a very interesting tool for this model, the Boundary Box (BBox), which allowed filtering of a way for a determined square-shaped area. As the BBox returned all the ways intersected by the square-shaped area – and it is important to know that a way represents each object in OSM and not just roads – a second filtering was needed to extract the type 'highway', which represents roads. The complexity of one filtering operation with a BBox is constant; therefore, if the second filtering is taken into consideration, the result is a complexity of o(W), where W now is the number of ways extracted from the BBox.

Once the 'point-on-a-way check' principle was validated, a method must be implemented to guide the car while navigating. The chosen method provided a complete trajectory of many points by passing coordinates for the origin and the destination points. In fact, the OSM way (roads) was represented by portions of many segments because it was impossible to build a curve in OSM. By combining the point-on-a-way check and the trajectory builder, it was possible to create a complete highway trajectory by re-building a new trajectory at each new segment of the highway. The point-on-a-way check was used to obtain the two vertex points of each highway segment. Then, the trajectory builder provided the trajectory points between them with a determined distance.

In the Figure 8, it is possible to see an illustrated example of a complete trajectory in red, composed of several small trajectories of each segment of the curved highway detected by the BBox. Note that the BBox and trajectory builder calling occurs at each segment ending.

The point-on-a-way-check and the trajectory builder were implemented in the CarNavigationHandler class. To access these functions, CarGlosm initiates CarNavigationHandler. The trajectory builder uses the Vector3i type, which is a structure containing three 32-bit fixed-point variable types to represent each component (*x, y,* and *z*). At this point, there are two main functions: GetPointsTrajectoryFromTo and WriteTrajectory. GetPointsTrajectoryFromTo returns a vector of Vector3i type, which contains all the points in the trajectory between the two points passed in a parameter with a fixed step passing through the parameter. WriteTrajectory is used to write all points of the vector returned by GetPointsTrajectoryFromTo in the GPX file, using GPXWriter class features.

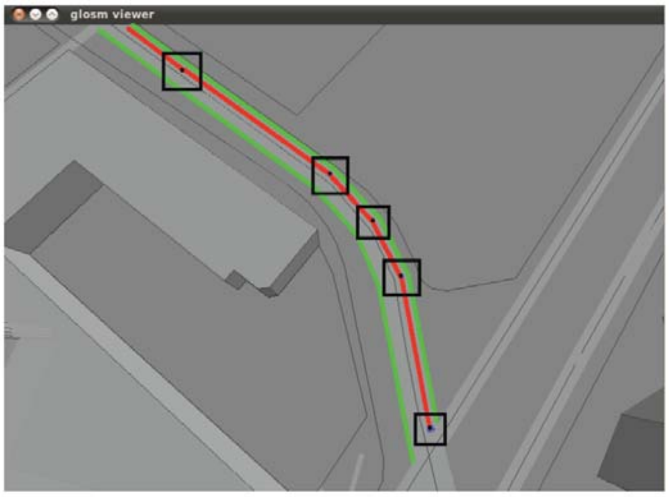

Fig. 8. Illustration of the trajectory builder principle.

C. *An implementation using Unity*

Unity is an open source development environment, which includes a game engine. There is an impressive and very active community of Unity users; hence, there is significant development support on the web. In 2014, there were approximately 2.5 million Unity users. Unity enables the export of the development to various platforms including IOS, Android, Windows, OS X, and the web. Unity includes a 3D physics engine (PhysX by NVIDIA), a sound engine, and supports scripts writing in many languages such as c#, Javascript, and Python. Figure 9 illustrates the Unity development environment. Figure 10 provides a view from the back of the vehicle that is driven by an actual driver.

I. Conclusions

This study proposes an architecture for a traffic and driving simulator involving 3D rendering, using Glosm based on Open Street Maps (OSM). The proposed architecture increases the realism of existing0 alternative modeling approaches because it uses images that better-represent the real world. In addition, the architecture explicitly and simultaneously includes actual drivers, pedestrians, and bikers. Further, the architecture enables modeling of the entire network with reasonable resources.

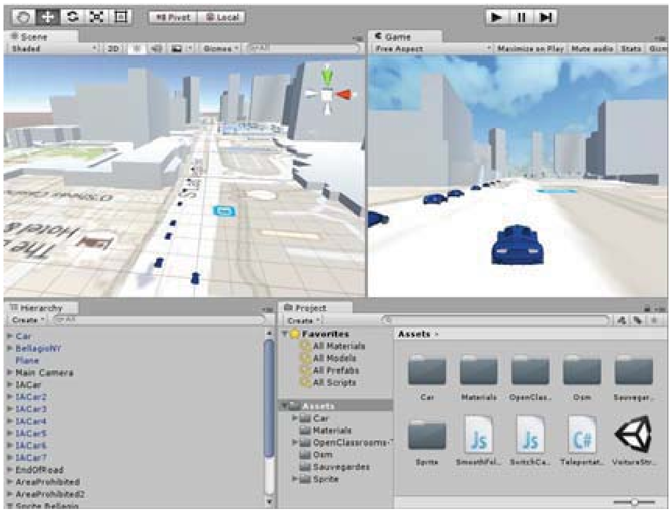

Fig. 9. Unity development environment.

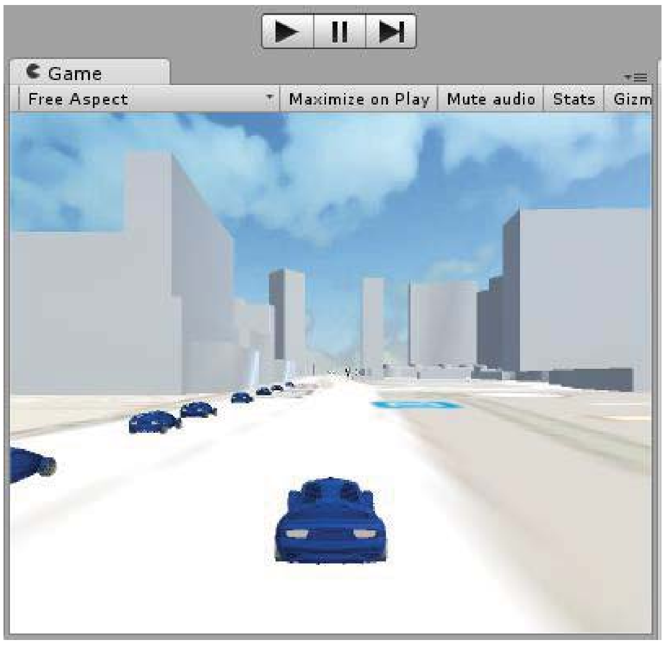

Fig. 10. View from the back of the vehicle driven by the actual driver.

To the best of our knowledge, no other alternative architecture exists that simultaneously considers all the elements of reality included in this proposed framework. Existing frameworks focus on model components of the real world, while the remaining aspects of reality are ignored or modeled using artificial entities. Implementation of this architecture is expected to provide unique opportunities to study vehicular traffic systems using actual drivers.

Future research could involve development and implementation using Unity as an alternative to Glosm and a subsequent comparison of the two implementations. Unity is a mature development environment, which utilizes common programming language and is a promising alternative to Glosm.

References

[1] F. Yang, Z. Kan, and S. Xianghong, “Behavior analysis of traffic accidents with high fidelity driving simulator,” International Conference on Transportation Engineering, Southwest Jiaoton, vol. 345, pp. 3231-3235, July 2009.

[2] F. Bella, and G. D’Agostini, “Combined effect of traffic and geometrics on rear-end collision risk: driving simulator study,” in Transportation Research Record: Journal of the Transportation Research Board, n^{o}. 2165, 2010, pp. 96-103.

[3] E. Blana, “Driving simulators as research and training tools for improving road safety. System and robot analysis and control design,” Lecture Notes in Control and Information Sciences, vol. 243, pp. 289-300, 1999.

[4] D. A. Liao, “High-Fidelity immersive cluster-based driving simulator for transportation safety research,” ACM International Conference on Virtual Reality Continuum and its Applications, New York, NY, USA, pp. 361-364, 2006.

[5] S. Danielsa, J. Vanrie, A. Dreesen, and T. Brijs, “Additional road markings as an indication of speed limits: results of a field experiment and a driving simulator study,” Accident Analysis and Prevention, vol. 42, pp. 953–960, 2010.

[6] J. Watchel, “Brief history of driving simulators,” TR News, Washington, issue 179, 1995, pp.26-27.

[7] J. Drosdol, and F. Panik, “The Daimler-Benz driving simulator,” SAE Technical Paper, Society of Automotive Engineers, Warrendale, Pa, USA, 1985, pp. 86-90.

[8] The National Advanced Driving Simulator. University of Iowa, Iowa. http://www.nads-sc.uiowa.edu. Accessed June 5, 2010.

[9] Human Factors Interdisciplinary Research in Simulation and Transportation, University of Minnesota, Minnesota. http://www.humanfirst.umn.edu/Capabilities/DrivingSimulation.html. Accessed April 5, 2010.

[10] VTI, Swedish National Road and Transport Research Institute, Sweden. http://www.vti.se/templates/Page_11862.aspx. Accessed April 6, 2010.

[11] University of Leeds Driving Simulator (UoLDS). http://www.its.leeds.ac.uk/itsresearch/facilities/uolds/. Accessed October 25, 2011.

[12] University of Michigan Transportation Research Institute. University of Michigan, Michigan. 2006. http://www.umich.edu/~driving/sim.html. Accessed April 6, 2010.

[13] T. J. Rosenthal, STISIM drive user’s manual, Systems Technology Inc., Hawthorne, CA, 2011.

[14] FORUM8 Co., Ltd. UC-Win/Road Ver.4 Operation guidance manual, 2010.

[15] A. Paz, R. Khaddar, N. Veeramisti, P. Kachroo, and N. Renous, “Architecture for an Interactive Motion-based Traffic Simulation Environment,” In: 91th Annual Meeting of the Transportation Research Board, Washington, DC, 2012.

[16] S. A. Mathieson, (11 May 2006), "A sidestep in the right direction," The Guardian. Retrieved 15 September 2009.

[17] Unity [Computer software]. (2015). Retrieved from http://unity3d.com/unity .