OpenEnergySim: A 3D Internet Based Experimental Framework for Integrating Traffic Simulation and Multi-User Immersive Driving

Arturo Nakasone
National Institute of Informatics
2-1-2 Hitotsubashi,Chiyoda-ku
Tokyo 101-8430, Japan
arturonakasone@nii.ac.jp

Helmut Prendinger
National Institute of Informatics
2-1-2 Hitotsubashi,Chiyoda-ku
Tokyo 101-8430, Japan
helmut@nii.ac.jp

Marc Miska
Smart Transport Center
Queensland University of Technology
GPO Box 2434
Brisbane, QLD 4001, Australia
marc.miska@qut.edu.au

Martin Lindner
National Institute of Informatics
2-1-2 Hitotsubashi,Chiyoda-ku
Tokyo 101-8430, Japan
martinl@nii.ac.jp

Ryota Horiguchi
i-Transport Lab., Co., Ltd.
Jinbo-cho 1-4, Chiyoda-ku
Tokyo 101-0051, Japan
rhoriguchi@i-transportlab.jp

Masao Kuwahara*
Tohoku University
Aoba, Aoba-ku, Sendai
Miyagi 980-8579, Japan
kuwahara@plan.civil.tohoku.ac.jp

ABSTRACT

In recent years, the use of computer-based simulations in the transportation domain has become increasingly important to analyze and test measures for Intelligent Transport System (ITS) policies. Simulators were built to address several aspects of transport, including traffic, driving experience, and pedestrian behavior. However, as the majority of available simulators are single-user stand-alone systems, traffic engineers cannot easily analyze more complex phenomena, such as the interaction between multiple human drivers or pedestrians. Furthermore, this limitation makes it difficult to collect large-scale behavioral data, which is necessary to draw valid conclusions on driving behavior. Emerging virtual world technology offers a convenient alternative. As a networked multi-user framework that allows users to immerse in the virtual world via a graphical self-representation (an 'avatar'), it allows to develop integrated simulation applications that are conveniently accessible by Internet. In this paper, we present OpenEnergySim, a virtual world based visualization application that integrates traffic simulation and immersive multi-user driving.

Categories and Subject Descriptors

I.6.7 [**Simulation and Modeling**]: Simulation Support Systems; I.6.7 [**Simulation and Modeling**]: Types of Simulation—*Animation, Combined, Visual*

General Terms

Design, Human Factors

Keywords

Traffic Simulation, Driving Simulation, Virtual Worlds, OpenSimulator

*This author is also affiliated to the ITS Center, Institute of Industrial Science, The University of Tokyo.

Industry track 2011, March 22-February 22

DOI 10.4108/icst.simutools.2011.238605

1. INTRODUCTION

Simulation of transportation systems has been one of the important application areas of computer simulation for more than 20 years [3, 8]. Recently, the demand for evaluating and testing Intelligent Transport System (ITS) applications has raised even more interest in simulation-based approaches. By using simulators, it is possible to estimate the effects of ITS measures without interfering traffic of the real-world transport network. Microscopic traffic simulation tools are being increasingly applied by traffic engineers and transport professionals to address dynamic and operational traffic problems and to evaluate such systems [21]. Many models for microscopic traffic simulation are now available (see [4] for an extensive list).

However, most available systems operate on rather artificial conditions. For instance, micro-simulation applications, i.e. traffic simulations that represent the trajectories of individual cars, do not consider the driving behaviors of real users as variables for their models. There are two main reasons. First, traffic engineers often follow mainly analytical objectives, and second, the technical implementation of complex traffic scenarios is cost-intensive. Currently, there is no integrated visualization platform capable of handling such dissimilar features as (1) smooth visualization of large-scale data for traffic simulation and (2) realistic physics emulation for users' driving experience.

Virtual world technology is considered as a convenient, cost-effective visualization platform for scientific research [2, 22], and [14] already proposed a virtual world based solution for visualizing the simulation of a Personal Rapid Transit System. However, the implementation of a systematic integrative solution using virtual worlds has not been explored yet. Therefore, in this paper, we present OpenEnergySim, an integrative virtual world based platform that combines traffic simulation and realistic user driving experience with the goal of providing traffic engineers a simulation tool for exploring and testing advanced ITS strategies. OpenEnergySim opens new and exciting ways to explore realistic transport scenarios, in which users can participate as drivers, pedestrians, or traffic engineers. Our solution is created in collaboration with research institutes and industry.

This paper is organized as follows. Section 2 presents an overview of research work on driving and traffic simulators and transportation visualization. Section 3 describes the OpenScience framework, our in-house implemented technology to facilitate application development for virtual worlds. Section 4 provides a detailed description of OpenEnergySim, by explaining its different components (server and client modules, game wheel interface) and their integration. Finally, Section 5 discusses and concludes the paper.

2. RELATED WORK

The visual representation of traffic flow in most currently available traffic simulation tools is two-dimensional (2D) [15, 7, 6] or three-dimensional (3D) [3, 27, 5, 26, 16]. The simulation component and the viewer (viewing program) are either located in the same machine (e.g. [3]), or organized in a client-server architecture (e.g. [27]).

3D visualization becomes an essential feature when there will be human-driven cars interfering with the simulation. Driving simulators are efficient tools to evaluate drivers' behavior. They use virtual reality tools to generate the user's sensations similar to those of driving a real car [28]. There are two kinds of driving simulators: (1) a moving-base simulators [12, 1, 10, 13, 11, 17] and (2) fixed-base simulators [25]. For instance, moving-base simulators are capable of giving motion feedback about how the car is moving in the simulated world.

However, often less advanced simulators are sufficient to obtain realistic data of users' driving behavior. For instance, [29] have implemented an environment for testing the influence of a voice-based command system on the user's driving. User-controlled cars and computer-controlled cars share a virtual space for experimentation. The STSoftware[1] used supports multiple users and easy scenario configuration. In contrast to our method, this approach is based on licensed software and does not provide on a persistent virtual world.

For evaluating an ITS application known as Intellidrive[2], the U.S. Department of Transportation used a driving simulator based in the Unreal game engine[3]. They claim that within a development period of four months, a demo with very high quality graphics could be developed. However, the simulator can only handle a single user at a time and not available to the general public.

[1]http://www.stsoftware.nl/
[2]http://www.intellidriveusa.org/
[3]http://www.udk.com/

Unlike the available methods, our approach is based on the networked OpenSim world simulator, which just requires a computer, Internet connection, and optionally, a game wheel for realistic driving. The OpenSim viewer is free for download. OpenSim supports persistent virtual worlds, where researchers may alter the test scenario collaboratively and in real-time, and the changes will be reflected in a central database for future use.

3. THE OPENSCIENCE FRAMEWORK

Our OpenScience virtual world framework API (Application Programming Interface) is based on OpenSim, an open source virtual world server. It was conceived to help developers implement OpenSim-based applications in a quick and easy way. The framework extends the functionality of libraries such as EML3D [20], MPML3D [24] and LibOpenMetaverse[4] by providing new features or more efficient versions of current existing ones, while maintaining platform independency for application development.

The OpenScience framework consists of two main components (see Fig. 1):

- *OpenLibrary* plugs directly into the OpenSim server and provides low level access to simple functions such as the creation and deletion of prims (primitive graphical elements in the simulator) or the subscription to specific virtual world events through a TCP/IP-based interface.
- *OpenAppCore* provides an abstraction layer on top of OpenLibrary, which enables developers to encapsulate complex virtual world objects to simplify their use in subsequent application developments. This layer also implements an access wrapper for OpenLibrary called OLWrapper to facilitate function calls to the library.

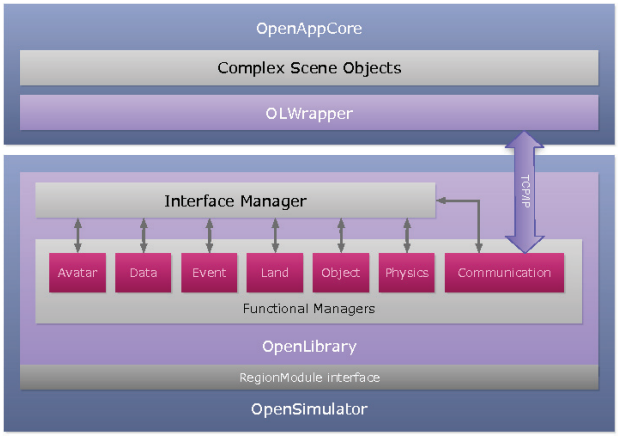

Figure 1: Functional diagram of the OpenScience framework.

3.1 OpenSim

OpenSim is a virtual world server technology that was introduced as an open source alternative to Linden Labs' popular "Second Life" service[5]. OpenSim allows users to build

[4]http://lib.openmetaverse.org/
[5]http://secondlife.com/

content and interact with each other inside a virtual world using a Second Life compatible client application. OpenSim maintains a large community of users and developers are actively working on it. At the time of writing of this paper, most of the functionality provided by Second Life has been implemented or even extended in OpenSim. OpenSim has several advantages over Second Life:

- *Extensibility.* OpenSim server functionality can be extended through the use of plug-ins (application modules that are automatically recognized and loaded by OpenSim). These plug-ins have complete access to OpenSim's internal structures, allowing developers to create highly customized solutions.

- *Server Control.* Using OpenSim allows developers to have full control of the whole application implementation process. In other virtual worlds such as Second Life, developers are constrained by the limitations imposed by the owners of the servers in which the virtual worlds run. With OpenSim, developers can overcome those limitations and find out the real limits for their implementations based on their own infrastructure.

- *Content Ownership.* The use of OpenSim also guarantees the complete control of the content deployed and generated in the virtual world, easing the fears of users regarding the ownership of their creations.

3.2 OpenLibrary

OpenLibrary[6] was implemented as a plug-in based on OpenSim's extensible architecture. It is loaded and instantiated by OpenSim during startup through .NET's reflection mechanism. The library features a similarly modular design approach and, therefore, allows the automatic discovery, loading and activation of functional units called managers during the server's startup. The current version is composed of a central Interface Manager and a set of core managers, which implement the functionality of OpenLibrary itself.

3.2.1 The Interface Manager

The Interface Manager acts as the central administration component in OpenLibrary and is responsible for loading the core managers and delegating commands to them (see Fig. 1). In order to provide access to the virtual world functionality to the core managers, the Interface Manager receives as initialization parameters references to virtual world regions (called Scenes in OpenSim) and to OpenSim configuration values. When the Interface Manager locates and loads the core managers, a reference of itself is passed to these managers, allowing not only manager development independency, but also a unified way to access OpenSim's shared resources.

3.2.2 The Core Managers

In addition to the Interface Manager, OpenLibrary includes a set of core managers that encapsulate the actual functionality of the library. Currently, seven core managers have been implemented: *Communication*, *Avatar*, *Event*, *Data*, *Land*, *Physics* and *Object*. Each manager is responsible for one specific set of tasks as described below:

[6]http://www.prendingerlab.net/globallab/technology/

- The *Communication Manager* is used for communication between OpenLibrary and external applications. It provides the ability to handle socket communication.

- The *Avatar Manager* implements functions related to the creation and manipulation of avatars, including functions to log avatars in and out, move avatars, and send chat messages.

- The *Object Manager* implements functions to create and manipulate objects in the virtual world, except objects considered part of nature such as trees or grass. Various types of primitive objects such as boxes, spheres or cylinders can be built and connected. Objects' color, texture and other physical properties can be modified as well.

- The *Land Manager* implements functions to create objects related to nature (e.g. trees, clouds, smoke) and to modify terrains.

- The *Data Manager* implements functions to connect external data sources to objects in OpenSim and vice versa. For instance, an external temperature data stream can be mapped to the height of an object in the virtual world or moving an object located inside the virtual world could trigger actions in the real world.

- The *Event Manager* implements event handling in OpenLibrary. For instance, an external application can subscribe for chat messages or object updates (e.g. changes in position, deletion, and so on).

- The *Physics Manager* implements functions to manipulate the physical properties of virtual world objects (e.g. apply forces to objects).

3.3 OpenAppCore

OpenAppCore is a software layer implemented over OpenLibrary. Its main objective is to provide virtual world based application programmers with a simple, flexible, and fully functional framework to interconnect and reuse runtime components that operate with entities in the virtual world. In OpenAppCore, programmers can develop customized, virtual world based components by using OpenAppCore's access point to OpenLibrary (the OLWrapper component) to facilitate function callings and event subscriptions. Once the components are implemented, other developers can use those components by instantiating them using OpenAppCore's component management processes. In this way, we are able to ensure that programmers can modularize their work and increase their development performance through software reusability. For instance, a user driven car component involves the implementation of several complex functions such as "accelerate", "turn lights on", and so on, and each of these functions involves, in turn, the implementation of several complex OpenLibrary function calls. Without OpenAppCore, each developer that desires to implement a user driven car would have to adapt or reimplement the same code for his/her own solution, creating problems for software maintenance and versioning. On the other hand, by using OpenAppCore, developers that merely want to use a user driven car would just need to copy the component to their own solution and instantiate it, saving precious development time.

Most of the objects managed by OpenEnergySim were implemented as OpenAppCore components, giving us the possibility to modularize our work and considerably reduce the application's overall implementation time.

4. INTEGRATING TRAFFIC SIMULATION AND MULTI-USER DRIVING

In this section, we will describe OpenEnergySim, a platform for behavioral driver studies in virtual worlds based on the OpenScience framework. OpenEnergySim allows us to integrate and synchronize simulation of microscopic traffic with the driving simulation of multiple drivers. The two main modules of OpenEnergySim are:

- *Server Module.* This module is responsible for (1) controlling the visualization of the traffic simulation, (2) managing the interactive virtual world based interface, (3) synchronizing the traffic simulation with the online driving behavior from users.
- *Client Module.* This module provides users the means to drive virtual cars using either a keyboard interface or a gaming wheel device.

The main components of both Server and Client modules will be explained in the following sections.

4.1 Server Module

The main purpose of the Server Module is to connect and synchronize the traffic simulator and the OpenSim virtual world, by performing the following tasks: (1) Receive information about virtual objects and/or vehicles controlled by the traffic simulator and create or update their corresponding graphical representations in OpenSim, and (2) receive information about user interaction with objects (such as vehicles) in OpenSim and forward it to the traffic simulator.

The Server Module is composed of five main components (see Fig. 2):

- *Core* component, whose main function is to instantiate the other components when the application starts.
- *User Interface* component, which is in charge of managing the virtual objects created by OpenEnergySim.
- *Traffic Simulator* component, which generates microscopic traffic, i.e. trajectories of individual cars. Currently, we use the X-Roads traffic simulator [18, 9].
- *Data Rate Converter* component, which handles the data synchronization between the traffic simulator and the user driven vehicles in the virtual world.
- *Event Logging* component, which stores attributes of user-controlled and computer-controlled cars (position, velocity) in a central database on a per frame basis.

OpenEnergySim uses the OpenLibrary and OpenAppCore components of our OpenScience framework (Sect. 3) that helps programmers to implement virtual world based applications easily. While OpenLibrary provides low level access to simple functions such as 'create' and 'delete' prims (primitive graphical elements) and listen to specific in-world events, OpenAppCore allows programmers to create and distribute complex objects based on OpenLibrary's functionality.

When the application is executed, the Server Module creates an instance of the OpenEnergySim Core component, which in turn instantiates the X-Roads Traffic Simulator component, the Data Rate Converter component, the User Interface component, and the Event Logging Module component.

In the next two subsections, we are just going to describe in greater detail the two most important components that implement the core functionality of the Server Module: the Data Rate Converter and the User Interface.

4.1.1 Data Rate Converter

Since the majority of traffic simulators such as X-Roads was primarily designed to run on stand-alone platforms, the traffic processing data rate is approximately 100 frames/sec. A frame is defined as the attribute information (position, velocity) of all the vehicles at a specific moment during the simulation. This update rate is extremely difficult to maintain in networked environments such as client/server based virtual worlds, because of the lag imposed by the network infrastructure.

To accommodate for the specific requirements of virtual world technology, we developed the Data Rate Converter component, which is responsible for maintaining coherent synchronization between the information generated by the traffic simulator and the information generated in OpenSim. The main purpose of the component is to collect all the asynchronous update events from X-Roads and the User Interface component, and synchronously forward them to their respective recipient in fixed-time intervals. Since both X-Roads and OpenSim have very different update rates, the Data Rate Converter automatically performs two procedures to ensure data interchange consistency:

- *Prioritization of Updates.* Prioritization means that updates that are more critical to the execution of the visualization are passed first. For instance, a VEHICLE DELETE command has a higher priority than a VEHICLE UPDATE POSITION command since the traffic simulator may try to update attributes of an object that does not exist in the virtual world anymore. Note that most traffic simulators follow an O–D (Origin–Destination) scheme, where cars are created at some initial location, and removed if outside of the area determined by the traffic simulator.
- *Merging of Updates.* Merging means that updates that are irrelevant because they were superseded by other updates are eliminated. For instance, a VEHICLE UPDATE POSITION command can be deleted if another VEHICLE UPDATE POSITION for the same vehicle arrived.

The functionality of the Data Rate Converter provided a smoother visualization of the car trajectories generated by the traffic simulator.

4.1.2 User Interface – Connecting to OpenSim

The User Interface component is in charge of (1) the creation, deletion and manipulation of any user interface object defined inside the virtual world, and (2) the handling the events generated by user interaction. The component interacts with OpenSim through the use of our OpenAppCore framework component solution. There are two types of user interface objects:

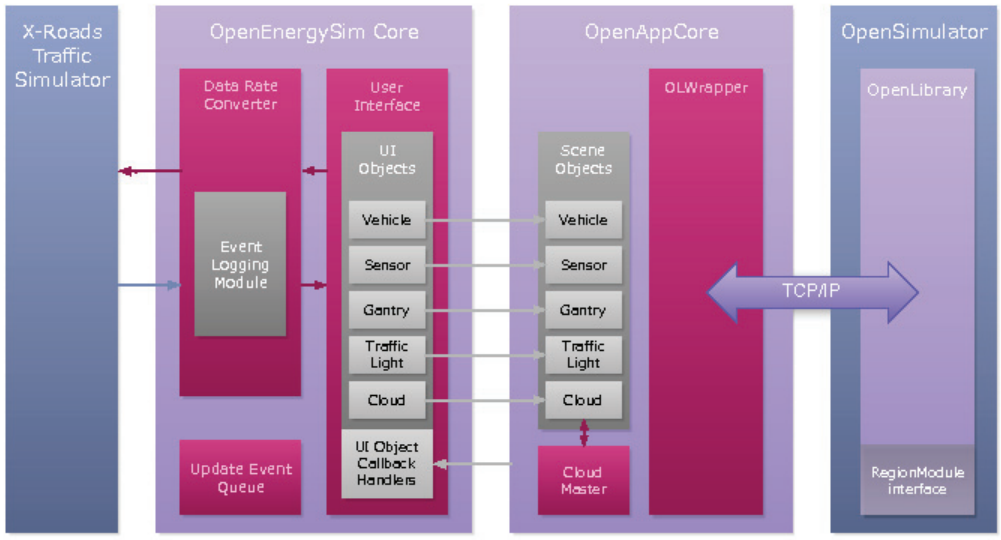

Figure 2: Functional diagram of server Module of OpenEnergySim.

- Objects that are dynamically manipulated by the traffic simulator, including vehicles, traffic lights, sensors, and Variable Message Signs (VMS).
- Objects that are used by users to manipulate the first type of objects, specifically button array objects for sensor manipulation.

Figure 3 shows the virtual world representation of the objects currently manipulated by X-Roads. Figure 4 shows the button array interface for instantiation of user driven vehicles and sensor creation. Users can create a car that is controllable by a wheel (bottom button), or controllable by a keyboard if a wheel is not present (above-bottom button). Sensor creation and deletion functions use the current position of the avatar that clicked the button to determine where the sensor is to be created or which sensor to remove.

Some User Interface objects are considered static and always present in the virtual world (e.g. traffic lights), wheras others have to be created at runtime, such as vehicles or sensors controlled by OpenEnergySim. The creation of an object in OpenSim is achieved in one of two ways:

- The object is loaded into the virtual world using an XML description file.
- The object is instantiated based on an inventory asset that is already stored in the virtual world asset manager.

In case of the first method, users do not need to know about the virtual world asset manager and can instantiate the same object independent of the virtual world environment. However, streaming XML description files into OpenSim could affect the server's performance if overused. Therefore, we use the first method for operations that are not so frequent, such as sensor creation or user driven vehicle instantiation. The second option is much faster and currently used to create computer-controlled vehicles.

For static objects, which are typically created before the execution of the application, a list containing the identifiers

Figure 3: Virtual representations of objects manipulated by X-Roads: a gantry for customized message display (top), a traffic light (bottom-left), and a sensor (bottom-right).

of those objects in the virtual world is passed to the User Interface component, so that their non-visual counterparts can be instantiated and recognized by OpenEnergySim. Although some objects can be retrieved by providing their name, others have to be retrieved by providing the actual identification number known as UUID (Universal Unique Identifier). This information has to be part of the XML-based configuration file before the Server Module starts its execution. An example configuration file is shown in Fig. 5.

4.2 Client Module

The virtual driving experience in most car related games is based on the implementation of two major components:

```
<?xml version="1.0" encoding="utf-8" ?>
<SceneObjects name="Kashiwa">
    <TrafficLights>
        ...
        <TrafficLight id="1" type="vehicle">traffic lights2-2</TrafficLight>
        <TrafficLight id="2" type="vehicle">traffic lights2-5</TrafficLight>
        <TrafficLight id="3" type="vehicle">traffic lights2-4</TrafficLight>
        <TrafficLight id="4" type="vehicle">traffic lights2-6</TrafficLight>
        ...
    </TrafficLights>
    <Gantries>
        <Gantry id="0">traffic sign2-5</Gantry>
        <Gantry id="1">traffic sign1-5</Gantry>
    </Gantries>
    <Vehicles>
        ...
        <Vehicle id="1" type="computer" name="Type_A_Blue">bdbd47e1-aa30-4b5d-8f83-e7f89704edad</Vehicle>
        <Vehicle id="2" type="computer" name="Type_A_Red">d2cb3976-8f9e-4fdc-812b-36f570b7832c</Vehicle>
        <Vehicle id="3" type="computer" name="Type_A_Silver">da72a6ee-e53f-4845-b190-14676d8fd6a1</Vehicle>
        <Vehicle id="4" type="computer" name="Type_A_White">ba321d5e-2d70-4bf8-9c3a-021fe26a3daf</Vehicle>
        ...
    </Vehicles>
</SceneObjects>
```

Figure 5: Sample configuration file for static objects. The UUIDS of traffic lights and VMS can be retrieved by their names, while vehicle templates for traffic simulation have to be specified directly with their UUIDs.

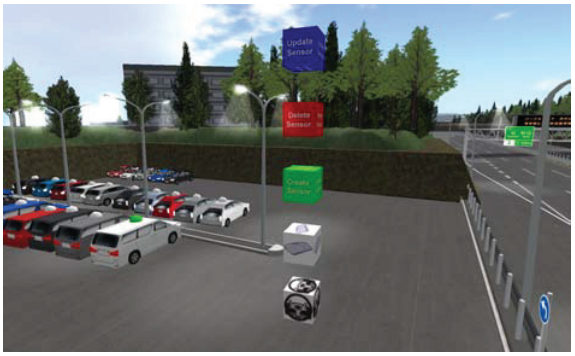

Figure 4: A button array interface displays the menu of options for sensor manipulation (top three buttons) and user driven vehicle instantiation (two buttons at bottom).

- The *vehicle physics engine* which governs the vehicle's motion dynamics, and
- the *driving user interface* which handles the hardware drivers use to interact with the vehicle.

Users' satisfaction of their driving experience is mainly determined by the stability and accuracy of the interaction between these two components. Games and other standalone applications enjoy the benefit of having both components in the same machine. In the networked environment like OpenSim, on the other hand, a generic physics called Open Dynamics Engine (ODE)[7] is located on the server and controls the physical processes of the entire environment.

[7]http://www.ode.org/

Therefore, a virtual world based solution has to support a two-stage physics interaction, in which the dynamic force values calculated by the vehicle's physics engine (client side) are passed to the generic physics engine (server side). The vehicle physics engine calculates the longitudinal force to be applied to the vehicle based on the user interface input and a vehicle profile. It then passes this value to the OpenSim physics engine which will move the vehicle inside the virtual world. Figure 6 shows the main components of vehicle motion in OpenSim.

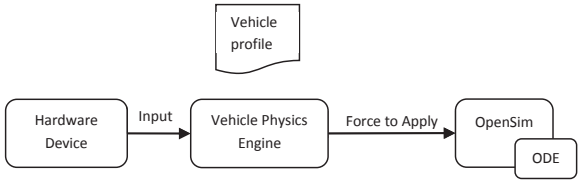

Figure 6: Client vehicle motion in OpenSim.

4.2.1 User Vehicle

A car in OpenSim is implemented as a frictionless object, composed of prims. The car is manipulated through direct intervention of the physics engine or through commands specified by programming scripts inside the prims that can change its overall status.

In OpenEnergySim, a car is defined by a main prim and a set of functional prims. The main prim is used to manipulate the entire motion of the car, whereas the functional prims are used to specify the individual elements of the car. The status of functional prims can change over time and affect their visualization in the virtual worlds, such as break lights

or CO_2 indicator.

The car management is divided into two main components. The LandTransportBasic component runs on the server as part of OpenEnergySim, and the LandTransportCont component that runs on the client. Both components make use of our OpenLibrary and OpenAppCore libraries (see Sect. 3). Figure 7 shows the component architecture diagram and the interaction of server and client components. The LandTransportCont component utilizes the WheelInterface to connect to the Logitech G27 Wheel[8] and uses the SpeedOMeterApp component to display a car's speed. On the other hand, the LandTransportBasic component is used to connect the car to OpenEnergySim.

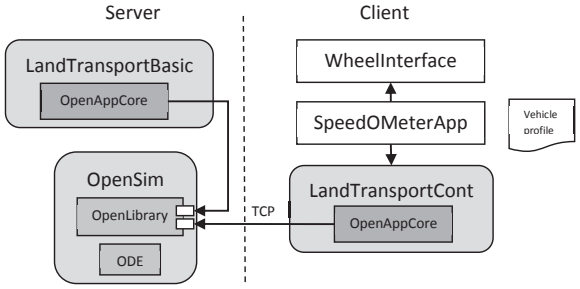

Figure 7: Component architecture for client vehicle motion in OpenEnergySim.

The LandTransportBasic component instantiates the car object in the virtual world and provides a basic interface to receive the car's properties, such as current velocity and position, and to set attributes such as lights (on/off). To instantiate a car, the component needs a XML file with the description of the car's visual elements (prims) or an UUID of an existing car inside the world. With this information, OpenEnergySim manages the car's creation and deletion, as well as the logging of its properties into the central database.

To drive the car, users must provide values (via driving) for linear force and moment applied to the car object. The LandTransportCont component allows users to control three car parameters:

- *Speed*: applies a linear force to go forward or backward.
- *Steering*: applies moment to turn right or left.
- *Driver's seat*: sets the position of the avatar inside the vehicle.

The LandTransportCont component was designed with a generic interface to support the default interaction paradigm of virtual world clients, which is the keyboard.

4.2.2 Wheel Interface

Since our goal was to provide a driving interaction device that mimics real-world driving, we developed a component called WheelInterface, which provides the connection between LandTransportCont and a Logitech's G27 Wheel [9] (see Fig. 8).

[8]http://www.logitech.com/

[9]http://www.logitech.com/en-us/gaming/wheels/devices/5184. Last Access: 2010/09/08

Figure 8: Driving a virtual car in OpenEnergySim using the Logitech Gaming Wheel G27.

The WheelInterface component retrieves the following values from the device: (1) Gas pedal pressure, (2) brake pedal pressure, (3) steering angle, and (4) push button events (e.g. gear change). These values are used by the vehicle physics engine to calculate the linear force and moment that will be applied to the car in OpenSim. In the vehicle physics engine, we use a model that takes two types of force into account [19]: Traction, i.e. force generated by the engine, and drag force, i.e. force created by friction and air resistance). With these values, the vehicle physics engine calculates the longitudinal force, and then a linear force is sent to OpenSim.

To obtain the moment, the physics model in [19] uses the angle of the front wheels as a parameter. However, in a real car, the angle of the steering wheel is not the same as the angle of the front wheels, thus it will thus be calculated. To obtain angle of the front wheels, there two values were taken into account: The steering ratio, i.e. the relation between the angle of the steering wheel and the angle of the front wheels, and the lock-to-lock (LTL) value, i.e. the number of turns a steering wheel takes to reach from the maximal right position starting from the maximal left position.

In a real car, the steering ratio is normally between 12:1 and 20:1. (A $n:1$ ratio means that a n degree angle in the steering wheel will turn the front wheels 1 degree). In addition, the LTL value is calculated taking into account the maximal angle of deflection of the wheels δ and the steering ratio ρ [10]. Equation 1 shows the formula for LTL calculation. For a graphical explanation of the relationship between angular values and wheel motion, see Fig. 9.

$$LTL = 2 \times (\delta \times \rho) \quad (1)$$

When driving a virtual car, the LTL value stays constant but the wheels' deflection and steering ratio change depending on the car speed. The basic principle is that with a constant steering wheel angle, the car turning rate is inversely proportional to its speed. For instance, if the car's speed is 100 km/h, a driver has to steer the wheel more in order to have the same turning angle than if the car's speed

[10]http://www.carbibles.com/steering_bible.html. Last Access: 2010/09/08

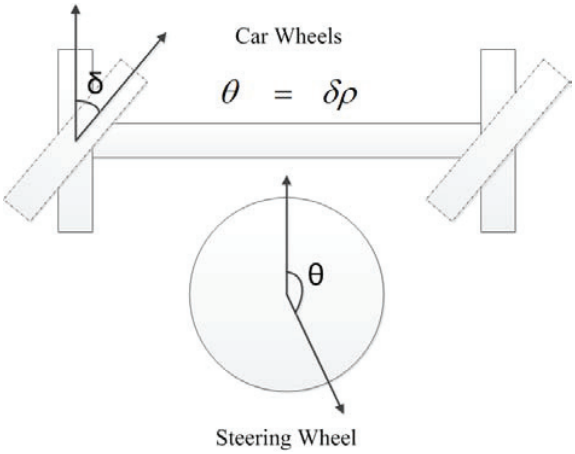

Figure 9: Graphical representation of the angular values used for the Wheel Interface component. In order to turn a car's wheels δ degrees, the steering wheel has to be turned θ degrees according to the formula shown in the diagram. The LTL value is, then, computed when δ reaches its maximum value, i.e. the maximum angle that the car's wheels are allowed to turn.

is 10 km/h. This is summarized in Table 1.

Table 1: Steering ratio mapping using LTL = 900 degrees. Wheel deflection values are taken from the actual Logitech G27 documentation.

Speed (km/h)	*Wheel Deflection δ*	*Steering Ratio ρ*
1	60°	7.5:1
100	12.5°	36:1

By taking into consideration the values in Table 1, we defined a linear interpolation equation to calculate the steering ratio ρ based on the car's speed ω. This is shown in Eq. 2.

$$\rho = 0.2878\omega + 7.2122 \quad (2)$$

Using this steering ratio value, we calculate the car's wheel rotation angle α based on the current car speed ω and the steering wheel rotation angle β (Eq. 3). The vehicle physics engine uses α to calculate the moment applied to the car.

$$\alpha = \frac{\beta}{\rho} = \frac{\beta}{0.2878\omega + 7.2122} \quad (3)$$

5. DISCUSSION AND CONCLUSIONS

In this paper, we have presented OpenEnergySim, an integrative multi-user traffic application solution based on virtual world technology. Our preliminary tests of the system show very promising results in terms of traffic visualization and user driving experience. In [23], we describe a car-following experiment that was conducted in OpenEnergySim. The results obtained in the virtual world could be shown were very similar to both real-world data on car-following and a car-following model.

However, the emergent nature of the virtual world technology created several challenges during the implementation of OpenEnergySim. We will briefly discuss two of them. First, the visualization of traffic simulation suffers from the overflowing amount of data from the traffic simulator. We had to manually fine-tune the rate of information transmission so that the visualization would not lag too much, based on trial and error. We found that this value depends mostly on the server's hardware and network conditions rather than the visual client. Techniques such as client side prediction algorithms might reduce the traffic visualization lag, but current visual client applications are extremely hard to extend. In the future, we plan to address this problem by experimenting with other, more flexible, visual clients such as Naali from RealXtend.[11]

HUD (Heads-up Display) interfaces would be preferable to let users manipulate in-world menu objects, but they proved to be extremely difficult to implement. While the use of 3D world buttons would allow simultaneous access for all users, it is impractical in a context in which avatars need to access the menu no matter where they are. We will be exploring ways to enhance client side user interfaces by implementing new functionality in our OpenScience framework.

Despite the technical difficulties faced during the development of OpenEnergySim, we are convinced that virtual worlds will become the most convenient platforms to test integrative solutions for the transportation domain. The main reasons are low production cost and high accessibility. Several complex challenges will arise in this endeavor since currently there is still a gap between social interaction (which is the initial main objective of the technology) and simulation of large-scale data. Nevertheless, we believe that the flexibility and openness of virtual world implementations such as OpenSim will allow us to close that gap. OpenEnergySim will allow us to provide engineers the tools they need to handle complex scenarios that closely resemble real-world problems.

6. ADDITIONAL AUTHORS

Additional authors: Juan Camilo Ibarra (National Institute of Informatics, email: `juanibarral@gmail.com`). Kugamoorthy Gajananan (National Institute of Informatics, email: `k-gajananan@nii.ac.jp`) Ricardo Mendes (National Institute of Informatics, email: `ricardo.tiago.mendes@gmail.com`), and Marconi Madruga (National Institute of Informatics, email: `marconi@nii.ac.jp`).

7. REFERENCES

[1] H. Arioui, S. Hima, and L. Nehaoua. 2 DOF low cost platform for driving simulator: Modeling and control. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2009)*, pages 1206–1211, 2009.

[2] W. Bainbridge. The scientific research potential of virtual worlds. *Science*, 317:472–476, 2007.

[3] J. Barceló, E. Codina, J. Casas, J. L. Ferrer, and D. García. Microscopic traffic simulation: A tool for the design, analysis and evaluation of intelligent transport systems. *Journal of Intelligent & Robotic Systems*, 41(2):173–203, 2005.

[11] http://www.realxtend.org. Last Access: 2010/12/02

[4] S. A. Boxill and L. Yu. An evaluation of traffic simulation models for supporting its development. Technical Report SWUTC/00/167602-1, Center for Transportation Training and Research, Texas Southern University, 2000.

[5] B. C. da Silva, A. Bazzan, G. Andriotti, F. Lopes, and D. de Oliveira. ITSUMO: An intelligent transportation system for urban mobility. In T. Bhme, V. L. Rosillo, H. Unger, and H. Unger, editors, *Innovative Internet Community Systems*, volume 3473 of *Lecture Notes in Computer Science*, pages 224–235. Springer Berlin Heidelberg, 2006.

[6] P. Hidas. Modelling lane changing and merging in microscopic traffic simulation. *Transportation Research Part C: Emerging Technologies*, 10(5–6):351–371, 2002.

[7] R. Horiguchi, M. Kuwahara, M. Katakura, H. Akahane, and H. Ozaki. A network simulation model for impact studies of traffic management avenue ver.2. In *Proceedings 3rd Annual World Congress on ITS*, 1996.

[8] R. G. Hughes. Visualization in transportation: Current practice and future directions. *Transportation Research Record*, 1899:167–174, 2004.

[9] T. Jiang, M. Miska, M. Kuwahara, A. Nakasone, and H. Prendinger. Microscopic simulation for virtual worlds with self-driving avatars. In *Proceedings 13th International IEEE Conference on Intelligent Transportation Systems (ITSC'10)*. IEEE, 2010.

[10] H. S. Kang, M. K. A. Jalil, and M. Mailah. A pc-based driving simulator using virtual reality technology. In *Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, VRCAI '04, pages 273–277. ACM, 2004.

[11] N. Kuge, T. Yamamura, and O. Shimoyama. A driver behavior recognition method based on a driver model framework. *Society of Automotive Engineers Transactions*, 109(6):469–476, 2000.

[12] J. Kuhl, D. Evans, Y. Papelis, R. Romano, and G. Watson. The Iowa driving simulator: An immersive research environment. *Computer*, 28(7):35–41, 1995.

[13] W.-S. Lee, J.-H. Kim, and J.-H. Cho. A driving simulator as a virtual reality tool. In *Proceedings IEEE International Conference on Robotics and Automation*, volume 1, pages 71–76, 1998.

[14] C. Lopes, L. Kan, A. Popov, and R. Morla. Prt simulation in an immersive virtual world. In *1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems and Workshops*, 2008.

[15] P. Lorenz, T. Schulze, and T. Schriber. The design, implementation, application and comparison of two highly automated traffic simulators. In *Simulation Conference Proceedings, 1994. Winter*, pages 1084–1092, 1994.

[16] J. Maroto, E. Delso, J. Felez, and J. Cabanellas. Real-time traffic simulation with a microscopic model. *IEEE Transactions on Intelligent Transportation Systems*, 7(4):513–527, 2006.

[17] M. Maza, S. Val, and S. Baselga. Control architecture of parallel and spherical driving simulators and related human factors. In *Proceedings 10th IEEE International Workshop on Robot and Human Interactive Communication*, pages 541–545, 2001.

[18] M. Miska. Real time traffic management by microscopic online simulation. Technical Report T2007/1, TRAIL Thesis Series, The Netherlands, 2007.

[19] M. Miska and M. Kuwahara. Nanoscopic traffic simulation with integrated driving simulator to investigate the sag curve phenomenon. In *Proceedings 9th ITS Symposium*, 2010.

[20] A. Nakasone and H. Prendinger. EML3D: An XML based markup language for 3D object manipulation in Second Life. In *Proceedings 9th International Symposium on Smart Graphics (SG'09)*, pages 263–272. Springer LNCS 5531, 2009.

[21] S. Panwai and H. F. Dia. Comparative evaluation of microscopic car-following behavior. *IEEE Transactions on Intelligent Transportation Systems*, 6(3):314–325, 2006.

[22] H. Prendinger. The Global Lab: Towards a virtual mobility platform for an eco-friendly society. *Transactions of the Virtual Reality Society of Japan*, 14(2):163–170, 2009.

[23] H. Prendinger, A. Nakasone, M. Miska, and M. Kuwahara. OpenEnergySim: Conducting behavioral studies in virtual worlds for sustainable transportation. In *IEEE Forum on Integrated and Sustainable Transportation Systems (FISTS'11)*, 2011. Submitted.

[24] H. Prendinger, S. Ullrich, A. Nakasone, and M. Ishizuka. MPML3D: Scripting agents for the 3D Internet. *IEEE Transactions on Visualization and Computer Graphics*, 2010. In press.

[25] M. P. Reed and P. A. Green. Comparison of driving performance on-road and in a low-cost simulator using a concurrent telephone. *Ergonomics*, 42(8):1015–1037, 1999.

[26] D. Rieck, B. Schünemann, I. Radusch, and C. Meinel. Efficient traffic simulator coupling in a distributed V2X simulation environment. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, number 72 in SIMUTools'10, pages 1–9. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.

[27] T. Wang, S. Tang, and P. Pang. 3D urban traffic system simulation based on geo-data. In *2nd International Conference on Information Technology: Research and Education (ITRE-04)*, pages 59–63, 2004.

[28] Y. Yang, J. Hu, and D. Chen. Research on driving knowledge expert system of distributed vehicle driving simulator. In *11th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2007)*, pages 693–697, 2007.

[29] A.-U.-H. Yasar, Y. Berbers, D. Preuveneers, and A. Jameel. A computational analysis of driving variations on distributed multiuser driving simulators. In *Proceedings of the 19th IASTED International Conference on Modelling and Simulation*, MS '08, pages 178–186. ACTA Press, 2008.