Development of an Object Oriented Library based on Java 3D to Facilitate the Construction of Virtual Environments in Driving Simulators

Dr. Talal Al-Shihabi*

Abstract

A driving simulator is a combination of hardware and software that enables the user to navigate through a real-time computer-generated virtual driving environment. Driving simulators are very valuable for conducting driving studies that are infeasible or unethical to conduct on the road. For studies that can be conducted in the real world, driving simulators provide a more cost effective solution and allow a much higher degree of flexibility in terms of changing the conditions of the environment for experimental purposes.

The corner stone when conducting any study on the driving simulator is the construction of a virtual driving environment that serves the purposes of this study. Even though driving environments can considerably vary between different studies, they all share the presence of common components that exist throughout the environment like roads, traffic signs and other components. The objective of this paper is to present the design of an object-oriented library that facilitates the construction of virtual driving environments when conducting studies on a driving simulator. The proposed library uses object orientation to hide implementation details from the designer of the environment which results in making the construction of such environments easier and more productive as well as their modification later if needed. The proposed library is based on the Java programming language and on Java 3D technology which makes it portable and usable on any platform that has an implementation of these two technologies.

Keywords: Virtual Environments, Driving Simulators, Java 3D

* Faculty of Civil Engineering-Damascus University

1. Introduction

A driving simulator is a combination of hardware and software that enables the user to navigate through a real-time computer-generated virtual driving environment. Driving simulators are very valuable for conducting driving studies that are infeasible or unethical to conduct on the road. For studies that can be conducted in the real world, driving simulators provide a more cost effective solution and allow a much higher degree of flexibility in terms of changing the conditions of the environment for experimental purposes.

One of the main conditions for a driving simulator is to have an acceptable level of presence to be regarded as a legitimate representative of the real world. The measure of presence that a driving simulator has on its human operators is complex. It can be summarized as the operators' degree of belief that they are driving in the real world when they actually are operating the simulator [2]. Each element of the driving simulator, be it a hardware or a software element, should be designed in a way that effectively improves the sense of presence within that simulator.

The components of a virtual driving environment have important impact on the results of any study conducted on the simulator since the environment is supposed to serve the purposes of the study. Even though driving environments can vary considerably between different studies, they all share the presence of common components that exist throughout the environment. Since the components of a driving environment are expected to give the user of the simulator the sense of being in a physical environment, it is very difficult to find a driving environment that does not include roads, traffic signs, buildings, trees, grass, walls, and scene background. The objective of this paper is to present the design of an object-oriented library that facilitates the construction of virtual driving environments when conducting studies on a driving simulator. The presented library has classes that simulate the physical components of a driving environment and uses object orientation to hide their implementation details from the designer of the environment which results in making the construction of such environments easier and more productive as well as their modification later if needed. Each component can be added to the virtual driving environment only by instantiating its equivalent class and pass the data needed for construction. The proposed library is based on the Java programming language and on Java 3D technology which makes it portable and usable on any platform that has an implementation of these two technologies. The developed library does not have dynamic components because these components have a specific nature that define their behaviour within the environment and thus they need special treatment.

Before presenting the library, we introduce the concepts of virtual environments and driving simulator to provide the reader with a background about the relevant technologies. After this brief introduction, we describe the overall design of the proposed library and its external dependencies. We then highlight the internal details of the classes within the library.

2. Virtual Environments

The term Virtual Environments (VE) or Virtual Reality (VR) represents a revolutionary approach to the way humans interact with computers. The emergence of VR was celebrated with the promise of replacing conventional input devices, such as the mouse and the keyboard, by others that facilitate direct interaction with computers through movement, speech, touching, and pointing. It turned out, however, that what VR can offer, because of its dependence on other fields of science, is largely determined by the existing technology of the time. The evolving nature of VR makes it very difficult to find a definition for VR that everyone agrees on. Generally speaking, a VR system is a system that consists of hardware and software components that collaborate to create "the illusion of immersion of the user in a computer generated environment" [7].

The roots of VR can be traced to the mid

1960s when Ivan Sutherland envisioned a system that combined position tracking and a head mounted display with computer-generated images that behaved like their physical counterparts [12]. Ivan Sutherland has come to be regarded as the father of modern VR.

Although the scientific community has always preferred the name Virtual Environments (VE), the term Virtual Reality has survived because of its extensive use by the public, the press, and the media. While the terms VR and VE both refer to the same thing, only the term Virtual Environments (VE) will be used throughout this paper.

Among all those who tried to come up with a definition for VE, Rory Stuart has come up with one of the most descriptive definitions:

"In the most strict sense, a VE system is a human-computer interface that provides 'interactive immersive multi-sensory 3-D synthetic environment'; it uses position tracking and real-time update of visual, auditory, and other displays (e.g. tactile) in response to the user's motions to give the user a sense of being 'in' the environment, and it could be either a single- or a multi-user system" [10].

Roy Kalawsky gave a more visionary than descriptive definition to VE:

"Virtual environments are synthetic sensory experiences that communicate physical and abstract components to a human operator or participant. The synthetic sensory experience is generated by a computer system that one day may present an interface to the human sensory systems that is indistinguishable from the real physical world. Until then, we have to be content with a virtual environment that approximates several attributes of the real world" [5].

While this vision of VE systems that encompass all human senses is still far from realization, many virtual environments have been developed with support for one or two senses and have proven to be immersive ones.

Of all human senses, vision has always been regarded as the most important for any VE system. In most VE systems, the display device is a head mounted display (HMD) with head motion tracking system connected to it. While some has required that using a HMD is a must for any system strives to be a VE system [5], it is widely accepted to use other display devices, including conventional computer monitors, in VE systems [1].

One of the most critical concepts in virtual environments is the concept of immersion or presence. The sense of immersion of presence within a virtual environment is what distinguishes them from other computer applications. Despite the importance of the concept of presence to VE, there are as many definitions to presence as those to VE itself [8]. Presence is a very complicated measure that can be summarized as the degree of belief the users of a VE system have that they are in the real physical environment while they actually are operating with the virtual environment [2]. Sheridan in [8] defines the followings as the main determents of presence within any VE: 1) the ability of the user to navigate through the environment, 2) the ability of the user to modify the environment, and 3) the extent of sensory information provided to the user.

Kalawsky believed that unnatural behaviours of certain objects within a virtual environment might have an effect on feeling within this environment that is more powerful than that of the high quality images. This was evident in certain works that were highly successful in the animation field yet they failed to meet the criteria of VE [5].

3. Driving Simulators

Driving simulation is considered to be a relatively new application of computer technology. The first wave of driving simulators started in 1980s. Compared to the aerospace industry, the automotive industry was slow in adopting simulation technology. This is believed to be partially due to the high cost of driving simulators prior to the 1980s in relation to the cost of motor vehicles especially that driving simulators may require higher-resolution graphics and faster response time than flight simulators which further

increases their cost. With the constant increase of computers' capabilities coupled with the continuing decrease in their cost, it has become more possible than ever to build driving simulators at a very reasonable cost. Driving simulators can be built now at a cost as low as few tens of thousands of dollars like many fixed-base driving simulators that are spread in research labs in many universities in the world or as high as few tens of millions of dollars like the National Advanced Driving Simulator, NADS, at Iowa State University and other simulators built by major auto companies around the world.

Driving simulators are generally divided into two categories: fixed-base simulators and motion-based simulators with the former costs much less than the latter. Fixed-base simulators generally include, but does not require, partial vehicle body with steering, gas, and brake controls connected to one of the simulator's computers. The simulator user navigates through the virtual driving scene while sitting in a seat within the vehicle body. Images are generated by one of the simulator's computers and are displayed on the computer monitor, a Head Mounted Display (HMD), and/or projected on a special screen that can be flat or curvy based on the horizontal field of view supported by the simulator.

The driving simulator of the Virtual Environments Laboratory at Northeastern University is a fixed-base simulator that facilitates the use of either a HMD or a projected screen. Three projectors are used to present a 180 degree field of view on a curved screen and a distortion correction algorithm is used to match the image to the screen's curvature. Figure 1 shows a driving environment with snow conditions generated on this simulator. An edge blending algorithm is used to blend the 3 projector images into one. Users can operate the simulator while sitting in a partial vehicle body through a real-vehicle pedal system and through a force-feedback steering wheel that is integrated with a high-fidelity car dynamics system. The simulator is enhanced with a side-view mirror that is attached to the vehicle body and with a rear-view mirror that is projected on the main driving scene. The simulator may also be used in "turning cabin mode" which provides a more realistic driving experience due to the driver physically turning when using the steering wheel.

The simulator was used for major studies on the divided attention capabilities of elder drivers, the effects of adverse conditions in driving environments on patients after performing refractive laser surgery, a study on simulation sickness in driving simulators [6], and a study on developing scenarios in virtual driving environments [11].

Figure 1. A driving environment with snow conditions generated on the driving simulator at Northeastern University

Motion-based driving simulators provide the simulator operator with a motion feedback and are generally much more expensive. The most prominent example of motion-based simulators is the National Advanced Driving Simulator, NADS. NADS has recently been built by the National Highway Traffic Safety Administration, NHTSA, at Iowa State University.

An important fact that the simulator user should be aware of is that he should agree to suspend his disbelief in the simulation and to act and react in the same ways he would on a real road [2].

Driving simulators can provide support to more than one user simultaneously. Each user in a multi-user driving simulator generally operates from a separate computer. All computers have copies of the simulator environment and are networked together.

Each computer sends to other machines on the network the updates on the environment that was done by the user of that computer. Multi-user driving simulators predominantly use peer-to-peer distributed simulation. The most widely established implementation of distributed driving simulators is the Distributed Interactive Simulation (DIS) protocol, IEEE standard 1278-1993 [3].

4. The Java 3D Technology

The Java 3D API is described as: "an application programming interface used for writing three-dimensional graphics applications and applets. It gives developers high-level constructs for creating and manipulating 3D geometry and for constructing the structures used in rendering that geometry. Application developers can describe very large virtual worlds using these constructs, which provide Java 3D with enough information to render these worlds efficiently." [9]

Java 3D extends the Java programming language with advanced 3D graphics and imaging capabilities [4]. It is platform independent, so it allows developers of 3D graphics to "Write Once, Run Anywhere". This feature enables users to reduce application development time and cost, by writing one version of an application for running on multiple types of platforms. In addition, Java 3D provides a set of object-oriented interfaces for applications that require high-performance 3D graphics. Object-oriented programming is easier to understand and make additions to. This feature can greatly reduce development time for improvements or additions to a Java 3D application.

Objects within a Java 3D based virtual environment are contained within virtual universe and compose what is called a "scene graph" in Java terminology. A scene graph represents instances of Java 3D classes in a graph-like data structure presented as nodes and arcs. The nodes in the scene graph are the instances of Java 3D classes while the arcs represent the relationships between these instances. The scene graph is composed of two main branches, the content branch and the view branch. The content branch contains all the objects in the environment, while the view branch contains everything necessary for appropriate viewing of the virtual environment.

Figure 2 shows the content side of the virtual driving environment that is consistent with the components that could be created using the proposed library. The VirtualUniverse is composed of a locale. The locale is composed of a BranchGroup object, called objRoot in the figure, that serves as the root pointer of all other nodes in the scene graph. BranchGroup objects are the only objects that can be inserted into a Locale's set of objects. objRoot has another BranchGroup as its child, called bgColorBG in the figure, as its child that has bgNode, a Shape3D, as its child. This object is responsible for presenting the driving scene background.

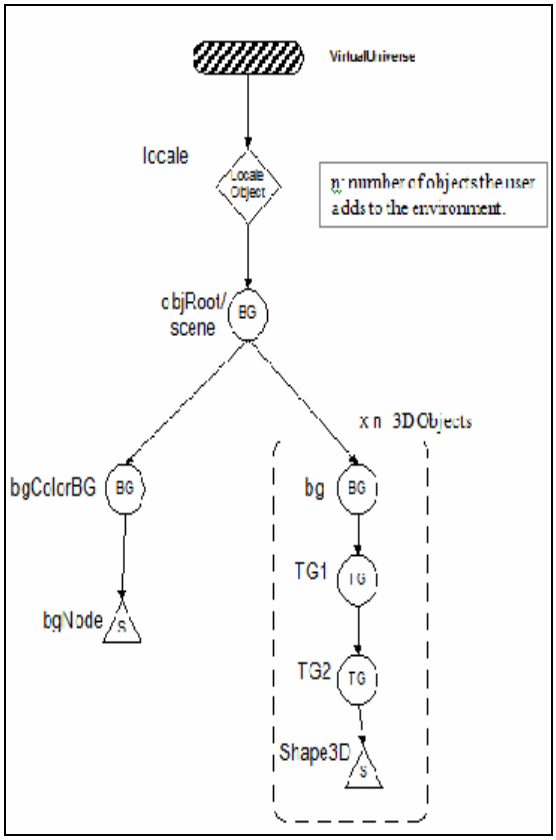

Figure 2. Content side of the scene graph

5. An Object-Oriented based design of a library of objects in a Java 3D virtual driving environment

5.1. The library description

An object oriented library was designed to facilitate the construction of virtual driving environments. The proposed modular design uses the object-oriented features of object orientation to hide implementation details from the user. This allows the construction and the modification of virtual driving environments easily and effectively. Since

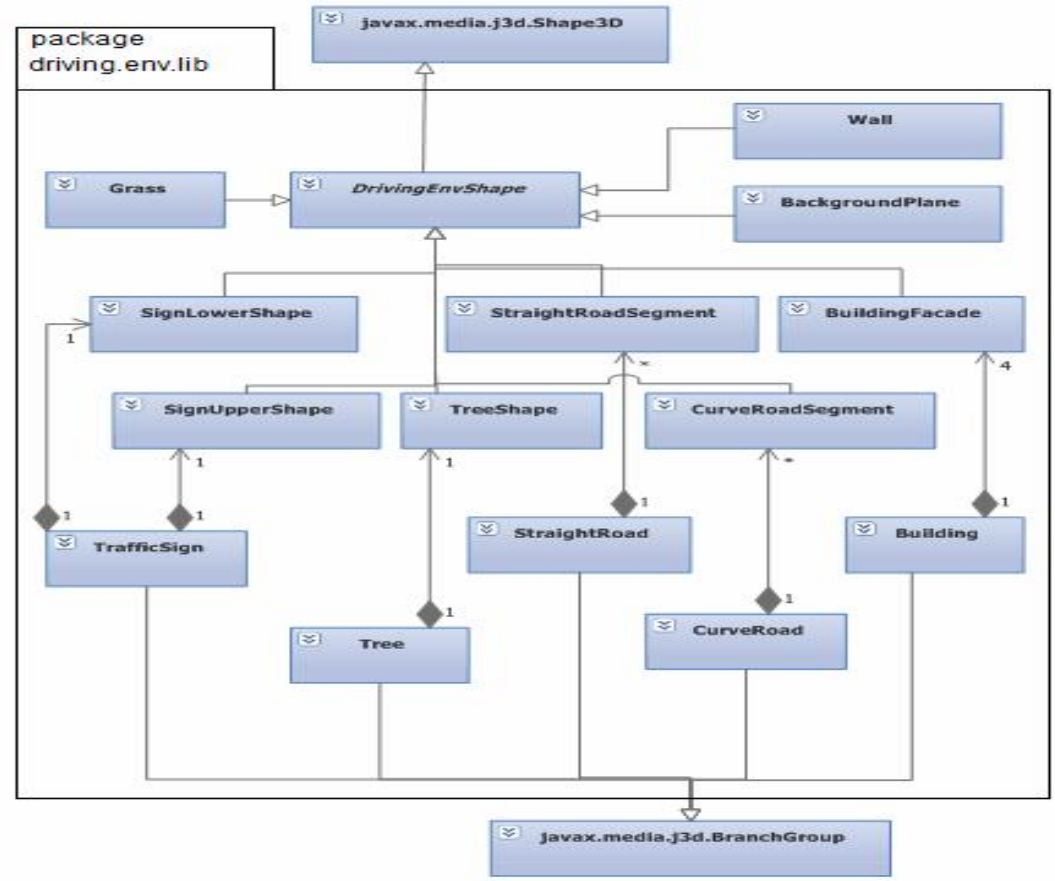

Figure 3. Library classes and their relationships

the implementation of the library is based on Java and Java 3D technology, this library is portable and can be used on any platform that supports these two technologies.

The main advantage of having such a library is the modularity and flexibility in constructing the building blocks of a virtual driving environment within the simulator.

The classes included in this library and their relationships are shown in Figure 3. The external dependencies include two classes

from the Java 3D API, the javax. media. j3d. Shape3D abstract class and the javax. media. j3d. BranchGroup class.
The general design approach in building this library consists of mapping the characteristics of the physical elements in a driving environment into logical objects in a corresponding virtual driving environment.

5.2. The library classes

The *DrivingEnvShape* Class
This class is an abstract class that extends the "*javax.media.j3d.Shape3D*" class from the Java 3D API and adds functionalities that characterize components of a virtual driving environment. The class detailed design is shown in Figure 4.

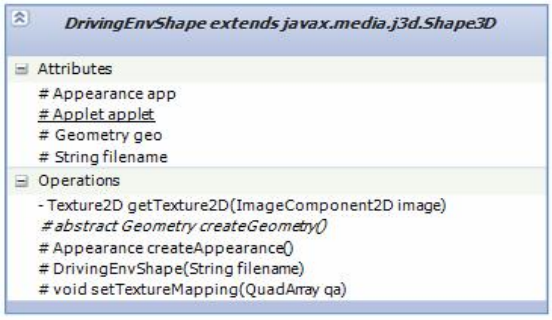

Figure 4. Detailed class diagram of the *DrivingEnvShape* class

Every other class in the library that represents a shape has to extend this library and thus implements the abstract methods: *createGeometry*, and possibly override the method: *createAppearance*.
The *BackgroundPlane* component class
This class represents the background of the virtual driving scene. The background plane is supposed to face the user of the simulator as he navigates through the virtual driving environment. Only one object of this class is instantiated in the environment. This object is added to the bgColorBG as shown in Figure 2 above. Code Listing 1 shows the implementation of the *createGeometry* method in this class. A detailed class diagram for this class is shown in Figure 5.

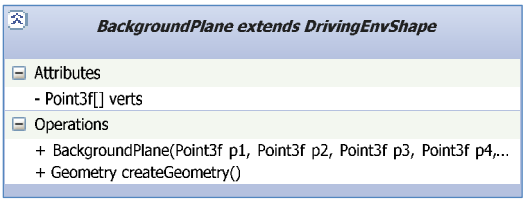

Figure 5. Detailed class diagram of the *BackgroundPlane* class

```
public Geometry createGeometry() {
  QuadArray bkgrnd = new QuadArray(4,
                QuadArray.COORDINAT
                ES                  |
                QuadArray.COLOR_3   |
                QuadArray.TEXTURE_C
                OORDINATE_2);

  bkgrnd.setCoordinates(0, verts);
  setTextureMapping(bkgrnd);
  return bkgrnd; }
```

Code Listing 1. the implementation of the createGeometry method in the BackgroundPlane class

The *Wall* component class
This class represents a wall component in a virtual driving scene. The wall object is a very small object whose geometry and appearance are very similar to the *BackgroundPlane* object except that it is not connected to the view object. Figure 6 shows a detailed class diagram for this class.

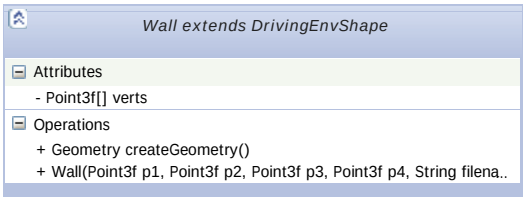

Figure 6. Detailed class diagram of the *Wall* class

The *Grass* component class
This class represents a plane of grass which is a very common component and repeatable in a virtual driving scene. The grass object is defined as a plane geometry with an image mapped as a texture to make its

appearance realistic and at the same time maintain the efficiency in viewing this object. Figure 7 shows a detailed class diagram for this class.

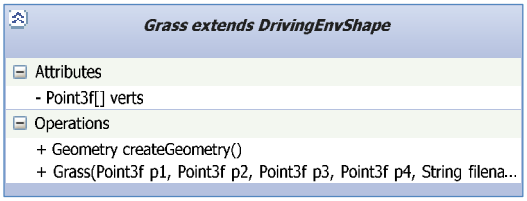

Figure 7. Detailed class diagram of the *Grass* class

The *Building* component classes

The addition of a "*Building*" component to a virtual driving environment requires the collaboration of two objects, the *BuildingFacade* and the *Building* objects.

The *BuildingFacade* class represents one façade of the building. A *Building* object is composed using four *BuildingFacade* objects. A building object is constructed using the location, dimensions and orientation of the building as well as an array of four images that are used for the four facades as explained in Figure 9 while the internal implementation of the Building class is shown in its class diagram in Figure 8.

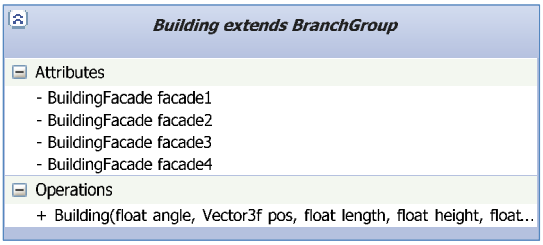

Figure 8. Detailed class diagram of the *Building* class

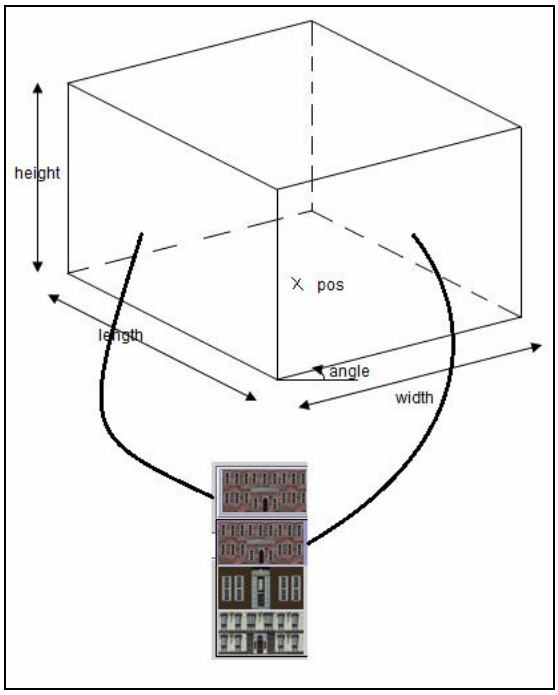

Figure 9. Characteristics of a *Building* component

The *Tree* component classes:

The addition of a *Tree* component to a virtual driving environment requires the collaboration of the *TreeShape* and the *Tree* classes.

The *TreeShape* class defines the geometry of the tree as two perpendicular planes with the same image mapped to both to create the illusion of a 3D tree in the scene. A *Tree* object is composed of one *TreeShape* object. A Tree object is constructed using the location, width and height of the tree as well as the image that will be mapped to each plane in the *TreeShape* object as explained in Figure 10.

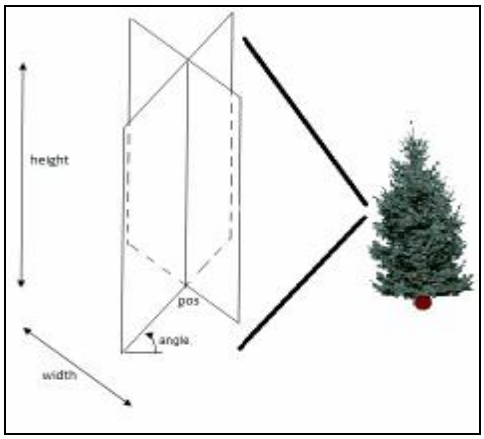

Figure 10. Characteristics of a *Tree* component

The construction of a *Tree* object is shown in Code Listing 2.

```
public Tree(float angle, Vector3f pos, float
            height, float width, String
            filename) {
    tree= new TreeShape(width, height,
            filename);
    Transform3D rot=new Transform3D();
    rot.rotY(angle);
    TransformGroup TG1= new
            TransformGroup(rot);
    TG1.addChild(tree);
    Transform3D trans=new
            Transform3D();
    trans.set(new Vector3f(pos.x, pos.y,
            pos.z));
    TransformGroup TG2= new
            TransformGroup(trans);
    TG2.addChild(TG1);
    this.addChild(TG2);
    this.compile();}
```

Code Listing 2. The construction of a Tree object

The *TrafficSign* component classes:

The addition of a *TrafficSign* component to a virtual driving environment requires the collaboration of three objects, a *SignLowerShape* object, a *SignUpperShape* object, and a *TrafficSign* object.

The *SignLowerShape* class represents the lower part of a traffic sign. An object of this class is represented as a colored rectangle. The *SignUpperShape* class represents the upper part of a traffic sign. An object of this class is represented as a rectangle with an image mapped on it. A *TrafficSign* object is composed of a *SignLowerShape* object and a *SignUpperShape* object. A *TrafficSign* object is constructed using the location, dimensions and orientation of the traffic sign as well as the image of the traffic sign upper part and the color of its lower part as explained in Figure 11.

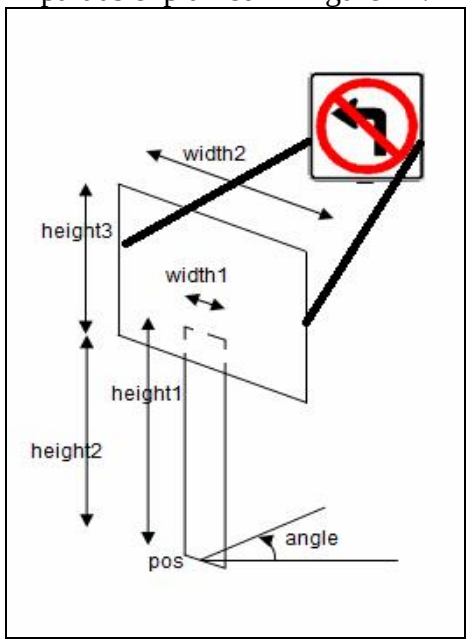

Figure 11. Characteristics of a *TrafficSign* component

The construction of a *TrafficSign* object is shown in Code Listing 3.

Code Listing 3. The construction of a TrafficSign object

The detailed class diagram for class *TrafficSign* is shown in Figure 12.

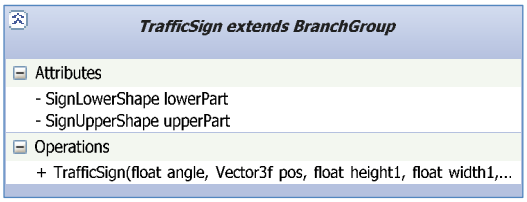

Figure 12. Detailed class diagram of the *TrafficSign* class

The *StraigntRoad* component classes

The addition of a straight road component to a virtual driving environment requires the collaboration of two objects, a *StraigntRoadSegment* object and a *StraigntRoad* object.

The *StraigntRoadSegment* class defines the geometry and appearance of a straight rod. An object of type *StraigntRoad* is constructed using its start point, its end point and its width as illustrated in Figure 13.

The *StraightRoad* class serves as a wrapper over the *StraightRoadSegment* to facilitate the addition of straight road components to the virtual driving environment and to unify the addition interface of straight roads and curvy roads.

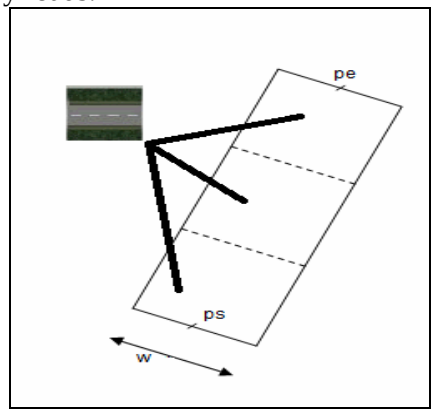

Figure 13. Characteristics of a *StraightRoad* component

The *CurveRoad* component classes

The addition of a curvy road component to a virtual driving environment requires the collaboration of two objects, a *CurveRoadSegment* object and a *CurveRoad* object.

The *CurveRoadSegment* class defines the geometry and appearance of one segment of a curvy road.

An object of type *CurveRoad* is constructed using its start point, its end point, its center, its width, and the number of segments throughout the curve. Figure 14 shows the characteristics of such an object.

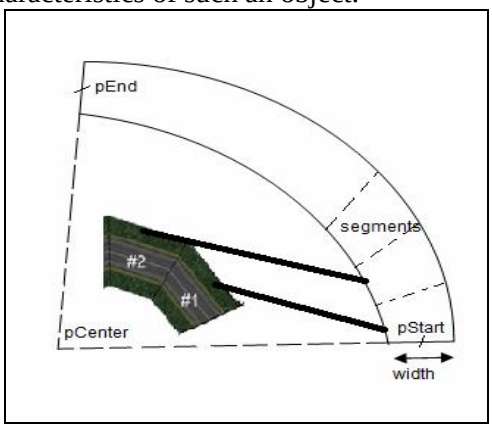

Figure 14. Characteristics of a *CurveRoad* component

5.3. An Example

As stated before, the purpose of this library is to facilitate and assist in building virtual driving environments easily and effectively through hiding the internal details of the components from the environment designer. To demonstrate the use of the library, a part of a driving environment is shown in Figure 15. The use of the library components for constructing this part is shown in code listing 4 below.

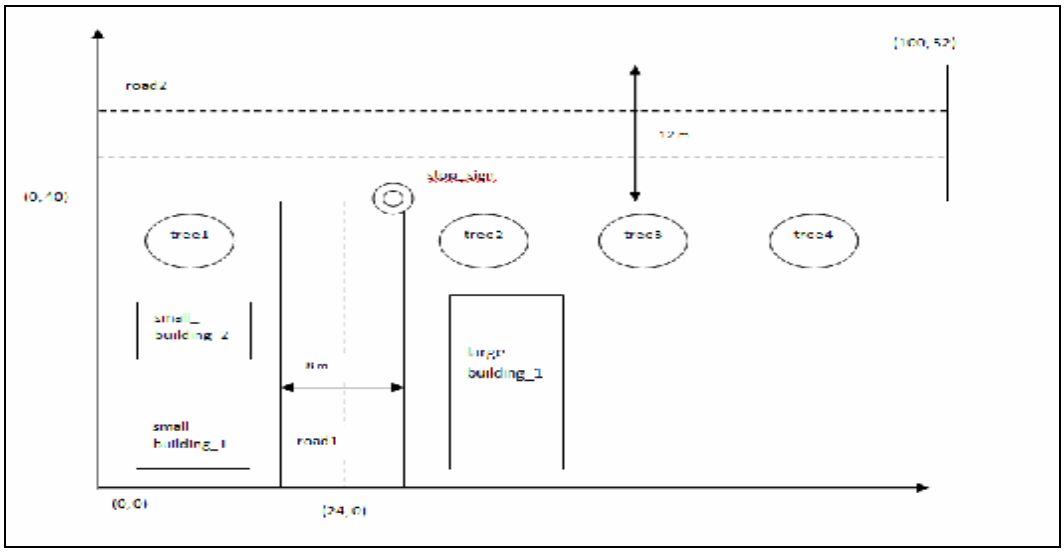

Figure 15. Part of a driving environment

```
// spreading the grass throughout the environment at level -0.1
Grass grass = new Grass(new Point3f(0,0,-0.1f), new Point3f(100,0,-0.1f), new
Point3f(100,52,-0.1f),new Point3f(0,52,-0.1f), "grass.jpg");
// building the two road segments as defined
StraightRoad road1 = new StraightRoad( new Point3f(24,0,0), new Point3f(24,40,0), 8,
"road_pattern_1.jpg");
StraightRoad road2 = new StraightRoad( new Point3f(0,46,0), new Point3f(100,46,0), 12,
"road_pattern_2.jpg");
// putting a Stop Sign at the end of road1
TrafficSign stop_sign = new TrafficSign(0.0f, new Vector3f (28f,40f,0f), 2f, 0.1f, new Color3f
(0.7f,0.7f,0.7f), 0.6f, 1.9f, 0.6f, "stop_sign.jpg");
// putting the three buildings around road1
String[] facades1= {"b1_f1.jpg", "b1_f2.jpg", "b1_f3.jpg", "b1_f3.jpg"};
String[] facades2= {"b2_f1.jpg", "b2_f2.jpg", "b2_f3.jpg", "b2_f3.jpg"};
Building small_building_1 = new Building(0.0f, new Vector3f (10f,6f,0f), 10f, 8f, 5f,
facades1);
Building small_building_2 = new Building(0.0f, new Vector3f (10f,25f,0f), 10f, 8f, 5f,
facades1);
Building large_building_1 = new Building(0.0f, new Vector3f (40f,20f,0f), 30f, 15f, 8f,
facades2);
// putting the four trees on the right side of road2
Tree tree1 = new Tree(0.0f, new Vector3f (10f,35f,0f), 15f, 5f, "tree1.jpg");
Tree tree2 = new Tree(0.0f, new Vector3f (38f,35f,0f), 20f, 8f, "tree2.jpg");
Tree tree3 = new Tree(0.0f, new Vector3f (60f,35f,0f), 16f, 7f, "tree2.jpg");
Tree tree4 = new Tree(0.0f, new Vector3f (80f,35f,0f), 22f, 9f, "tree1.jpg");
```

Code Listing 4. Code for building the contents of Figure 15.

6. Discussion and conclusion

In this paper, we described a library that can be used to create virtual driving environments easily and effectively. The use of the object-oriented methodology has helped greatly in hiding the implementation details of the environment and in transforming the process of constructing virtual environments into a high level process. The use of the library can yield elegant and quality code and improve usability.

The reliance of the Java and Java 3D technology make the virtual environments produced using the described library portable and can be used on any platform that supports these two technologies.

REFERENCES:*

[1] Arther, K.W., Booth, K.S., and Ware,C. (1993). Evaluating 3D Task Performance for Fish Tank Virtual Worlds. ACM Transactions for Information systems, 11(3). 239-265.

[2] Hein, C. M. (1993). Driving Simulators: Six Years of Hands – on Experience at Hughes Aircraft Company. In proceedings of the Human Factors and Ergonomics society 37th Annual Meeting. Santa Monica, CA, 607-611.

[3] IEEE , Institute of election and electronics engineer. (1993). International standard, ANSI, IEEE Std 1278-1993, Standard for Information Technology, Protocols for Distributed Interactive Simulation. New York, NY.

[4] Java 3D API, https://java3d.dev.java.net

[5] Kalawsky, R. S. (1993). The Science of Virtual Realty and Virtual Environment. Wokingham, UK:Addison-Wesley.

[6] Mourant, R. R. and Zhishuai Yin. (2010). "A Turning Cabin Simulator to Reduce Simulator Sickness." Proceeding of IS&T/SPIE Electronic Imaging 2010, San Jose, CA, pp. 17-21.

[7] Paush, R., van Dam, A., Robinett, w. and Bryson, S. (1994). CHI' 94 Tutorial Notes: Implementing Virtual Realty. Boston, MA: Association for Computing Machinery.

[8] Sheridan, T.B (1992). Musings on Telepresence and Virtual Presence. Precence 1(1). 120-125.

[9] Sowizral H., Rushforth K., Deering M. (1997) The Java 3D API Specification. Addison-Wesley Pub Co.

[10] Stuart, R. (1996) The Design of Virtual Environments, McGraw-Hill, New York.

[11] Suresh, Piriyakala and Mourant, Ronald R. (2005). "A Tile Manager for Deploying Scenarios in Virtual Driving Environments". Proceedings of the Driving Simulation Conference North America, pages 21-29.

[12] Sutherland, I. (1965). The Ultimate Display. In proceeding of the IFIP congress, Vol. 2, 506-508.

* Received 11/11/2012