StellarSim: A Plug-In Architecture for Scientific Visualizations in Virtual Worlds

Amy Henckel and Cristina V. Lopes

University of California, Irvine, Irvine CA 92697, USA
{ahenckel,lopes}@uci.edu

Abstract. More and more researchers in a variety of fields are turning to virtual worlds for 3D simulations and scientific modeling. The use of virtual worlds in this manner offers many benefits. However, the critical task of creating 3D objects for a simulation model is still a manual process, which can be time consuming. Our research concentrates on creating a process that allows for the automatic population of 3D objects in virtual worlds for researchers.

This paper presents a plug-in architecture framework that allows the automatic creation of 3D objects and externalizes the behaviors of the objects. This plug-in architecture makes it possible to utilize the underlying framework of the virtual world platform for the display of arbitrary data, in a straightforward manner. A prototype application was created based off this framework, augmenting the 3D platform OpenSim.

Keywords: virtual environments, content creation, 3D objects, simulation, modeling, astronomical modeling, OpenSim.

1 Introduction

The National Aeronautics and Space Administration (NASA), has announced its interest in modeling mission data from interplanetary probes [1] [2] and developing a Mars and moon virtual habit [1] in Second Life [3], a popular virtual world. This agency has also announced its interest in importing data from the International Space Station Mission and Mars Mission into Second Life [4] [5].

In the field of astronomy, there are additional research interests for the use of virtual worlds. For example, Piet Hut, Institute for Advanced Study, discusses the benefits of using virtual worlds for collaboration among astrophysicists. He started the group Meta-Institute for Computational Astrophysics (MICA) for this purpose [6]. He discusses using virtual spaces as collaboration tools, allowing users to see visual representations of other users (i.e., avatars), and allowing communication through voice or text with other avatars in real time. He believes this method of communication gives the user a sense of being in the same room as other people, which assists with sharing ideas [6] [7].

In addition to these examples, there exists a large number of research interests in the field of astronomy for the utilization of virtual worlds in simulation and modeling. Due to the tremendous amount of data for stellar bodies and

F. Lehmann-Grube and J. Sablatnig (Eds.): FaVE 2009, LNICST 33, pp. 106–120, 2010.

their movements recorded from interplanetary probes and telescopes [8] [9], we examined the process of creating 3D representations of data (3D objects) used for simulations and modeling in virtual worlds. What we observed was an inflexible and time consuming process. For example, the 3D objects themselves must be manually created, and manually customized. Textures (i.e., images that can overlay a 3D object) are appointed to the object. To order to assign a behavior (i.e., controlled movement, change in appearance, etc.), a script defining the movement of an object has to be created and added to the individual 3D object.

We then looked at existing simulation programs used by astrophysicists to discover if they experienced similar problems, to the ones we observed.

We gathered information pertaining to the use of these programs though informal discussions and interviews from astrophysicists involved in the analysis and planning stages of the mission life cycle, as these stages make use of simulation programs.

The results show there are various programs employed for modeling astronomical data. Each program either produces a simulation for one type of data, or requires extensive programming to visualize multiple types of data. This is due to the different behaviors of an object, various aspects of an observed object, and multiple sources of input for an object. The results also revealed only a few of the programs used allow for the addition or modification of a 3D object; these processes involved are manual and time consuming. Many of the astrophysicists we spoke to express the need to use more than one program for a task due to these restrictions.

This paper presents StellarSim, a framework designed to address the weaknesses pointed out above. We show how attributes about a planet (size, texture, name, shape etc.) and modules defining the behavior of a planet can be easily imported into a virtual world. The end result produces simulations consisting of 3D representations of these kinds of data. These representations, or 3D objects, are automatically created and displayed in the virtual world. Our focus was to create a flexible framework for the importing of customized attributes and behaviors of an object into a rendering environment with minimal effort for the user. We refer to this as the automatic population of 3D objects. StellarSim allows for an easy representation of multiple types of data by externalizing the behaviors and attributes of these 3D objects, in particular when applied to astronomy.

This paper focuses on StellarSim's design. Future goals include receiving additional iterative feedback on the design from end-users, adding further requirements and performing a thorough user study in situ. This paper describes some of the motivations, findings, challenges, and future goals.

2 Related Work

Since this framework utilizes virtual worlds in creating a simulation program for astrophysicists, this section will first discuss current 3D simulation programs used in this field, then discuss current research with modeling in virtual worlds.

2.1 3D Programs in Use

The programs in use by this group of astrophysicists for 3D modeling and simulations are: Science Opportunity Analyzer (SOA) [10], Satellite Orbit Analysis Program (SOAP) [11], Java Mission-planning and Analysis for Remote Sensing (JMARS) [12], Solar System Simulator [13], Visualization ToolKit (VTK) [14], and Satellite Tool Kit (STK) [15]. As well, some users have built custom simulation programs for particular missions due to one or more missing requirements from available programs.

SOAP and SOA are only available to NASA, JPL, Aerospace Corporation, and affiliates. SOAP is used to assist in projecting and analyzing satellite orbits, including positions and availability of sensors and communication links. It utilizes the Spacecraft, Planet, Instrument, Camera-matrix, and Events toolkit (SPICE) [16] [17], which is produced by NASA. SOA is used to find an ideal time for an observation on a mission. The user can select a specific point in space in which to "view" the surroundings.

JMARS full edition is only available on particular missions. JMARS displays images of 3D terrain and information from stellar instruments, such as maps and image footprints. Solar System Simulator is a website that displays images of projected trajectories of orbiting objects from a defined point of reference and date.

STK and VTK are toolkits which are more flexible, and offer more options. However, they require extensive programming for modeling use. VTK is a rendering tool used to produce 3D images and plots, of any type of data. STK is used to calculate position, orientation, view maps and images, check visibility of a sensor, and project trajectories of satellites and probes.

Each program has its own advantages and disadvantages. Of the programs listed, none allow for the automatic population of 3D objects from within. The task of creating 3D objects in these programs is a manual process, if it is allowed. Certain programs are restrictive on what an end-user can create. The programs listed, except for STK and VTK, focus on modeling only one type of data. For example, a program either focuses on the detailed terrain of a stellar body, or the projected orbit of a stellar body, but not both. STK and VTK are both toolkits and require extensive programming from the user for modeling.

Because of this, most of the users questioned utilize more than one simulation program to accomplish a task. The majority of these users would prefer to utilize one program, mainly to avoid duplicating work. Flexibility is an issue as well. Most users expressed their requirements change from mission to mission. Because of this, the users may switch from one group of programs to another depending on the required tasks for planning a particular mission.

One additional observation was the inability for real-time collaboration with these programs. All data files employed by these programs are stored on the user's PC, and cannot be easily shared or accessed by other users.

2.2 Virtual Worlds

There are many virtual world platforms in existence today and there is much research on simulations and modeling in these virtual worlds. While it is beyond

the scope of this paper to discuss and compare the research in all virtual worlds, this paper will discuss the research taking place in the most popular virtual world platform used for simulation models, Second Life [3], and an open source virtual world with similar user functionality, OpenSim [18].

NASA is involved in several projects within Second Life, and looks to virtual worlds for assistance in future missions. Jessy Cowan-Sharp, who helped create NASA's CoLab island in Second Life [2] [19] [1] sees virtual worlds as a flexible set of tools and useful for building scientifically accurate representations of data from planetary probes. She mentions that collaboration with the members of the virtual world community could add to their tests and sees the collaboration capability of a virtual world as beneficial to this field.

Aside from the astrological simulations mentioned, Second Life is widely popular for creating simulation models for demonstrational, pedagogical, and analytical purposes. Examples include a simulation modeling a Personal Rapid Transit system [20], a demonstration showing how ants find food and leave a pheromone trail [21], a heart murmur simulation [22], a hallucination simulation [23], and a genetic model display [24].

The second virtual world platform discussed in this paper, OpenSim, is an open source project, which employs Second Life's client software to connect to an OpenSim server. For the purpose of this feasibility study, OpenSim proved to offer a more viable solution for our needs than Second Life. Both Second Life and OpenSim were evaluated as a platform for this framework. Second Life had some limitations which prevented our framework from being feasible. OpenSim however produced a feasible and flexible solution.

There are additional open source virtual worlds, such as Sun Microsystem's Wonderland [25] and Darkstar [26], and Croquet [27]. Further studies would be needed to develop and test the operability of the StellarSim framework with such virtual worlds. With the continuous development of 3D virtual worlds, we believe more and more opportunities will arise for further development of simulation models.

3 Usage Scenario

StellarSim provides a method to input customized attributes and assign independent behaviors to 3D objects, which accommodates for greater control over customizing a simulation model on an ad hoc basis. Other modeling applications can be created based off of this framework. This section describes three example scenarios of how StellarSim can be employed.

Scenario 1 - Projected Path: Emma is required to calculate the projected path of a shuttle and make adjustments to that path. She has to: (a) input data for the shuttle, (b) increase and decrease the speed or orbit of the shuttle to discover if the projected path will collide with other objects, (c) if the projected path will place the shuttle in the right place on a specified date, and (d) if adjustments are needed, modify the projected path accordingly.

Scenario 2 - Collaboration: After calculating the appropriate path of the shuttle, Emma now needs to: (a) share her calculations with a coworker and (b) both will have to make adjustments to the simulation model, as appropriate.

Scenario 3 - Change Perspective: Jorge has received specifications on a new mission involving a probe to Jupiter. He will need to (a) input data involving the probe (b) monitor the projected path of the probe from Earth to Jupiter then (c) monitor the projected orbit of the probe around Jupiter up close to see if any other object, i.e., one of Jupiter's moons, will interfere with the probe's lens and its predetermined target.

Details on the use of StellarSim for these three usage scenarios are described in the Evaluation section.

Data Model. Current simulation programs are strongly coupled with the data they represent; behaviors are not dynamically assigned to 3D objects. In order to effectively create 3D models of multiple types of data, attributes (size, shape, texture, etc) and behaviors (controlling factor of an object's movement) of the objects must be external to the main program. Figure 1 shows an illustration of distinct types of objects that can have representation in StellarSim and their structure within the virtual world platform.

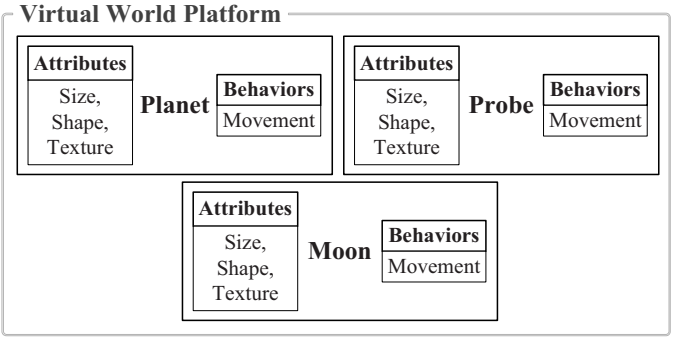

Fig. 1. Representations of stellar objects within StellarSim

The design of StellarSim allows for the externalization of the attributes and behaviors of the 3D objects through the use of configuration files and assemblies in Dynamic-link library (DLL) files. Once the attributes and behaviors are defined, the object appears to behave as expected to the end-user.

Dynamics Modeling. Much research and progress have been made to better understand and model our universe. The progress is remarkable, and beyond the scope of this paper to fully address the outcome. There are many possible methods for calculating planetary positions in the virtual world environment. For this framework, algorithms simplified by Paul Schlyter [28], are used to calculate

the coordinates of the planets. To reference Paul Schlyter on the accuracy of his algorithms, "The accuracy of the computed positions is a fraction of an arc minute for the sun and the inner planets, about one arc minute for the outer planets." [28]

This method was chosen for the initial design of the system for the reason that it supports the externalization of the behaviors of objects. DLLs utilizing these algorithms are assigned dynamically to the objects to calculate their positions.

4 Architecture and Implementation

The framework for StellarSim provides a plug-in architecture that applies an object's attributes and behaviors from external files. The object's attributes and behaviors are specified in configuration files and C# classes which are independently compiled into DLL files. StellarSim loads those independently developed components and executes them. This allows for the external data and behavior to be instantiated in the generic virtual world. First it allows for greater flexibility in modeling various types of objects. Second, it allows the systemic utilization of the underlying virtual world platform across a variety of applications.

This architecture provides numerous benefits for an application design within virtual worlds. The benefits of using OpenSim for our framework include a direct access to the backend of the virtual world server for dynamic additions and modifications of 3D objects, use of the system timer for orbit simulations, and registration of events (discussed in more detail later in this section).

The plug-in architecture was implemented using OpenSim Region modules. Region modules are collections of classes that implement the interface IRegionModule (DLLs themselves). There are many Region modules standard with OpenSim. A new Region module, OpenSim.Region.StellarSim (StellarSim module), was created for this application.

The StellarSim module reads in attributes for new objects and loads the appropriate behavior modules designed to calculate the object's position. It then calls on these behavior modules and uses an existing module in OpenSim, OpenSim.Region.Environment (Environment module), to create the 3D objects and update their positions in the virtual world. The StellarSim module also hosts web services used for a web form interface for our prototype application. It listens for http requests and executes appropriate functions for these requests. Figure 2 shows the architecture of the StellarSim framework.

The interface IRegionModule, listed below, requires that the functions listed within it are included in all classes which implement this interface. During the initialization of the OpenSim server, the working directory of OpenSim (opensim/bin) and the scriptengines directory (opensim/ScriptEngines) are scanned for DLL files containing classes which implement IRegionModule. Once an appropriate DLL file is located, OpenSim loads this file and executes the Initialise and PostInitialise functions within it. After Region modules are loaded into the OpenSim server, they remain active until the server is shutdown. Region modules are flexible in nature, and can perform a variety of tasks, including creating

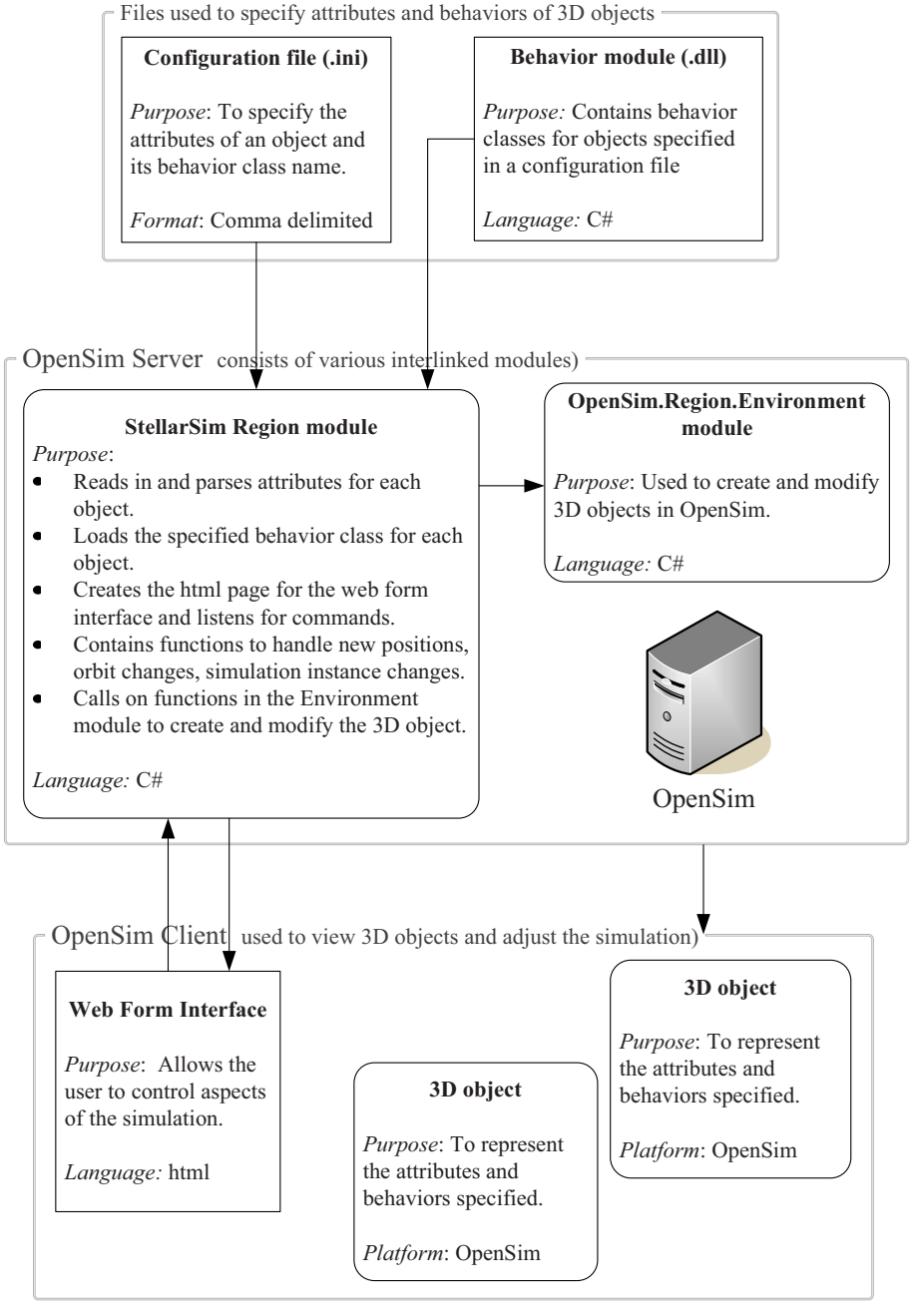

Fig. 2. Diagram of the architecture of StellarSim

objects, modifying objects (such as updating the positions of objects), using the system clock timer and registering for events (i.e., listening to chat messages, user logins, http requests, texture transfers, etc.). Registration for events allows an action to occur within the registered module in response to an event.

IRegionModule interface

```
public interface IRegionModule {
  void Initialise(Scene scene, IConfig config);
  void PostInitialise();
  void Close();
  string Name { get; }
  bool IsSharedModule { get; }
}
```

The StellarSim module includes a new interface, IAstronomicalModule. IAstronomicalModule is designed to be implemented by external behavior modules for specifying the position of a 3D object. The StellarSim module reads in attributes from comma delimited text files with the extension of .ini (configuration files). These files reside under the StellarSim main directory (opensim/bin/StellarSim). For each object listed in a configuration file, a class name must be provided. This referenced class must implement the IAstronomicalModule interface and exist in a DLL file under the StellarSim lib directory (opensim/bin/StellarSim/lib). The StellarSim module loads the specified class and associates it with the object's attributes from the configuration file. It then uses the Environment module to create a 3D object based on the provided information. Once the 3D objects are created, they can be viewed via logging into the virtual world.

Listed below is the format of the configuration file, and the IAstronomicalModule interface. Examples using these files are shown in the Evaluation section.

*Format of the configuration file, *.ini*

```
ObjectName, ClassName, Size.x, Size.y, Size.z, Shape, Texture
```

The IAstronomicalModule interface implemented by classes defining movement of a 3D object

```
using System;
using OpenMetaverse;
namespace OpenSim.Region.StellarSim.Interfaces {
  public interface IAstronomicalModule {
    Vector3 PositionFromDate(DateTime date);
  }
}
```

By using C# interfaces in this manner, certain functionality is then guaranteed to exist in loaded modules. For example, the IAstronomicalModule interface requires that the function 'Vector3 PositionFromDate(DateTime date)' exists in

an object's behavior module. This ensures that the StellarSim module can call on a function PositionFromDate from a specified class, give it a date (in DateTime format), and receive a position vector (in Vector3 format).

After the position vector is received, the StellarSim module scales the information appropriately to fit within the simulation region limits. It then calls on the Environment module to update the position of the 3D object.

The orbit of a 3D object in the virtual world is controlled here by continuously updating that object's position. The OpenSim system timer is used; the object's position is recalculated and adjusted every second.

Next, the StellarSim module registers for http request events. After a request is made through the web form interface, the StellarSim module will call on appropriate functions within itself to respond to the request.

The end result is as such: the end-user can launch the web form interface of StellarSim and control the objects in the simulation. Figure 3 shows StellarSim's web form interface through the virtual world client. (This interface can also be used through a web browser.) For example, to view the objects on a particular date, a user makes a request from the web form interface and the StellarSim module updates the objects' positions based on information from the behavior modules.

Using Region modules allowed for the dynamic additions of and modifications to the 3D objects, use of the system timer for the orbit simulation, and the registration of events. By using this approach, the creation and modification time of each object is relatively small. This allows for a smooth simulated orbit.

The code for StellarSim is written in C# with 501 lines of code for the main module (OpenSim.Region.StellarSim), 7 lines of code for the interface class (IAstronomicalModule), and 735 lines of code for two example simulations (described next).

5 Evaluation: StellarSim

This section first shows two example simulations implemented with StellarSim. The first example simulation displays the planets in the solar system. The second example simulation displays Jupiter and its moons. These two simulations are created in the same region. When switching between simulations all 3D objects of the previous simulation are deleted then all 3D objects of the new simulation are created in the same space.

Next, this section discusses previously defined usage scenarios and their application with these two example simulations.

5.1 Applications

Simulation 1 - Solar System: There are many 3D simulation programs that model the planets in our solar system, as this is a necessity for planning a mission in our solar system.

The Solar System simulation was implemented by adding a configuration file and a DLL file. Shown below are sections of the configuration file and class library

files used to create the DLL file that will specify the attributes and behaviors, respectively, of the 3D objects in this simulation.

SolarSystem.ini

```
Mercury,...,http://maps.jpl.nasa.gov/pix/mer0muu2.jpg
Venus,...,http://maps.jpl.nasa.gov/pix/ven0ajj2.jpg
Earth,...,http://maps.jpl.nasa.gov/pix/ear0xuu2.jpg
Mars,...,http://maps.jpl.nasa.gov/pix/mar0kuu2.jpg
Jupiter,...,http://maps.jpl.nasa.gov/pix/jup0vss1.jpg
Saturn,...,http://maps.jpl.nasa.gov/pix/sat0fds1.jpg
Uranus,...,http://maps.jpl.nasa.gov/pix/ura0fss1.jpg
Neptune,...,http://maps.jpl.nasa.gov/pix/nep0fds1.jpg
Sun,...,http://solarviews.com/raw/sun/suncyl1.jpg
```

Examples.SolarSystem:Behavior.cs

```
using System;
using OpenSim.Region.StellarSim.Interfaces;
using OpenMetaverse;
namespace Examples.SolarSystem{
  public class Earth : IAstronomicalModule{
    #region IAstronomicalModule Members
    Vector3 IAstronomicalModule.PositionFromDate(...){
      Planet earth = new Planet();
      int d = earth.convertTime(date);
      Vector3 newPos = earth.CalculateEarthPosition(d);
      return newPos;
    }
    #endregion
  }
  public class Sun : IAstronomicalModule{ ... }
  ...
  public class Neptune : IAstronomicalModule{ ... }
}
```

Examples.SolarSystem: Planet.cs shows sections of the class "Planet", which was used in Examples.SolarSystem:Behavior.cs, listed above. Combined, they return a position in Vector3 format for any object they define when given a Julian date.

Examples.SolarSystem:Planet.cs

```
using System;
using OpenMetaverse;
namespace Examples.SolarSystem{
  public class Planet{
    public Vector3 CalculateSunPosition(int d){
      ...
```

```
      Vector3 sunPos = new Vector3((float)sunx,...);
      return sunPos;
    }
    public Vector3 CalculateMercuryPosition(int d){
      ...
      calculateXYZ(...);
      Vector3 planetPos = new Vector3((float)xeclip,...);
      return planetPos;
    }
    public Vector3 CalculateEarthPosition(int d){ ... }
    ...
    public Vector3 CalculateNeptunePosition(int d){ ... }
    public int convertTime(DateTime date){ ... }
  }
}
```

Remark on Scale. To accurately display a model of our solar system to scale, allowing the smallest planet Mercury the smallest representation possible in OpenSim, 73 regions of virtual land in diameter are required for a full orbit around the sun for the farthest planet, Neptune. For the sake of this example simulation (and to view more than one planet in a screen shot), the distances between the planets have been scaled down.

Simulation 2 - Jupiter and moons: To switch from a general view to a detailed view, the web form interface is used. Figure 3 shows the web form interface and 3D objects in the Jupiter simulation. The moons shown are: Io, Europa, Callisto, and Ganymede.

The Jupiter simulation was implemented in the same fashion as the Solar System simulation, with a configuration file and a DLL file. Shown below are sections of these files.

Jupiter.ini

```
Callisto,...,http://solarviews.com/raw/jup/callistocyl2.jpg
Europa,...,http://solarviews.com/raw/jup/europacyl2.jpg
Ganymede,...,http://solarviews.com/raw/jup/ganymedecyl2.jpg
Io,...,http://solarviews.com/raw/jup/iocyl2.jpg
Jupiter,...,http://solarviews.com/browse/jup/jupitercyl1.jpg
```

Examples.Jupiter:Behavior.cs

```
using System;
using OpenSim.Region.StellarSim.Interfaces;
using OpenMetaverse;
namespace Examples.Jupiter{
  public class Jupiter : IAstronomicalModule{
    #region IAstronomicalModule Members
    Vector3 IAstronomicalModule.PositionFromDate(...){
```

```
      Planet Jupiter = new Planet();
      int d = jupiter.convertTime(date);
      Vector3 newPos = jupiter.CalculateJupiterPosition(d);
      return newPos;
    }
    #endregion
  }
  public class Callisto : IAstronomicalModule{ ... }
  ...
}
```

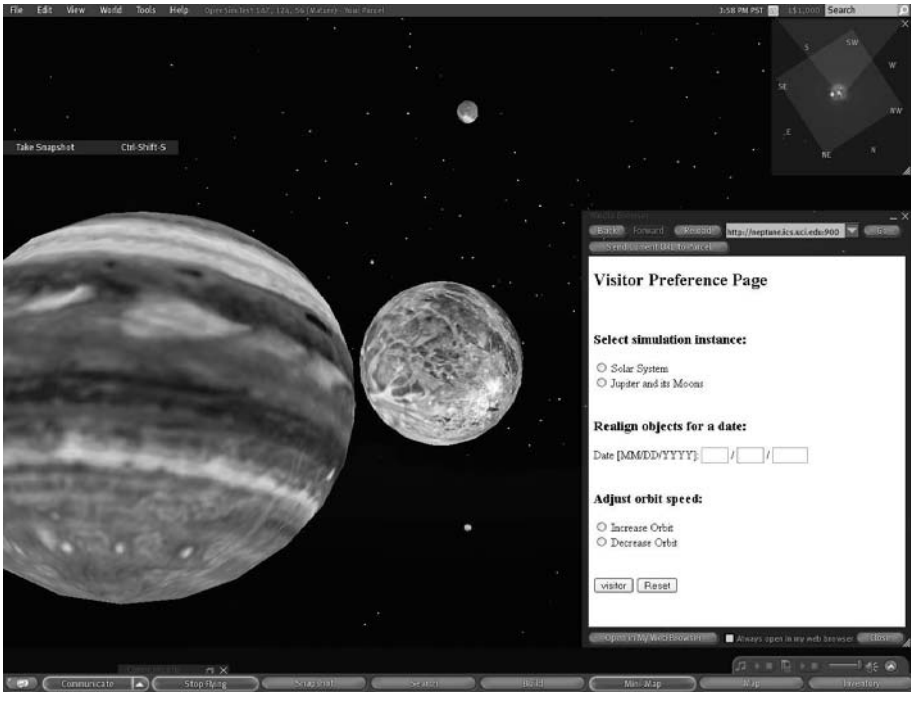

Fig. 3. Image from the virtual world client showing Jupiter and its moons with StellarSim. StellarSim's web form interface is shown on the right

5.2 Usage Scenarios with StellarSim

Scenario 1 - Projected Path: (a) To input new data for a shuttle within the Solar System simulation, Emma can create or modify a behavior module adding a class which implements the interface IAstronomicalModule. Next, information on the shuttle's attributes and a reference for the new class is added to the configuration file under OpenSim/bin/StellarSim for the Solar System simulation (SolarSystem.ini). Then, by using the web form interface and selecting the instance "Solar

System" Emma can now see the new shuttle along with the planets configured in the simulation. (b) To increase and/or decrease the speed of the shuttle, Emma will again use the web form interface and select "Increase Orbit" or "Decrease Orbit" accordingly. She can then view the shuttle with respect to other objects within the Solar System simulation to look for any potential collisions. (c) To align objects in the simulation corresponding to a particular date, Emma can use the web form interface and enter a date under "Realign objects for a Date:". (d) If any adjustments are needed, Emma can modify the behavior module for the shuttle then reselect the Solar System simulation and her new changes will take effect immediately.

Scenario 2 - Collaboration: (a) Emma can share her simulation with anyone who has access to log into the OpenSim server hosting the simulation. (b) Both Emma and her coworker can modify the behavior module for the shuttle, reselect the Solar System simulation and see their changes immediately.

Scenario 3 - Change Perspective: (a) Jorge can input new data about the probe in the same manner as Emma in scenario 1. (b) By viewing the Solar System simulation, and using functionality listed in scenario 1, Jorge can follow the projected path of the probe to Jupiter. (c) To switch to a more detailed view of Jupiter and its moons, Jorge can either add a new behavior module for the shuttle to depict its movement in orbit around Jupiter or use the behavior module from the Solar System simulation. Next he can modify the configuration file under OpenSim/bin/StellarSim for the Jupiter simulation (Jupiter.ini), adding a line for the shuttle's attributes and referencing the desired behavior class name. By using the web form interface and selecting the instance "Jupiter and its Moons" he can now see Jupiter in a more detailed view and monitor if one of Jupiter's moons will interfere with the probe's objective.

5.3 User Feedback

Our prototype application was shown informally to several astrophysicists from our user group and a couple of suggestions came up after.

The first suggestion was to add the ability for the user to obtain a set of real rectangular coordinate points, for any point on the screen. Currently the rectangular coordinates shown through the virtual world client refer to a location within an area in the virtual world and not the rectangular coordinates that correspond to a location within the space being simulated.

The second suggestion was to add SPICE [17] toolkit to the backend of the StellarSim framework. Currently its use is implemented with the programs SOA and SOAP. Implementing this within StellarSim is feasible and discussed in the next section.

6 Conclusions and Future Work

Virtual worlds are being used in research for simulations and modeling more and more. The advantages of these virtual worlds make them attractive for modeling and simulations.

In view of the increasing interest from the field of astronomy to utilize virtual worlds in simulations and modeling, and the large amounts of data typically involved with this field, we looked at the process of creating 3D objects in virtual worlds. We found this process to be arduous.

We then looked to the existing simulation and modeling programs used by astrophysicists to see if a more automated process existed there.

Not only did we find a similar problem among current simulation programs, but we also discovered these programs had further limitations including the lack of structure to enable collaboration with others. This particular problem is remedied through the use of a virtual world; other issues are addressed through the use of StellarSim.

The framework of StellarSim was designed to be flexible in nature, utilizing the plug-in modular structure of OpenSim. It allows for the automated process of 3D object population and ad hoc modifications to the 3D objects. By externalizing the attributes and behaviors of 3D objects, this framework generates an application independent of the type of data being modeled which in turn makes the application usable for more than one type of data.

The application, StellarSim, was designed for the use of astrophysicists during the analysis and planning stages. It is currently a prototype and online connected to UCIGrid.

Future versions of StellarSim can implement additional functionality features. Features such as obtaining real rectangular coordinate points and adding the use of SPICE in the backend of the framework. Implementing the SPICE toolkit would allow for the use of SPICE available functionality within StellarSim which includes the use of SPICE-hosted ephemeredes (tables of values that provide positions of astronomical objects at a given time) in determining the movements of 3D objects. This functionality would allow for a greater accurancy in computed positions of planets, moons, probes, satellites, etc.

Our framework presented here can be extended to other fields, and the prototype application for StellarSim can be modified to incorporate additional functionality. This approach utilizes an open source virtual platform to produce real-time 3D models of planetary objects. This framework provides instant shared access to a 3D simulation created in real-time and facilitating collaborative tools that enable scientists to review and discuss these simulations.

References

1. Boyle, A.: Virtual-space gurus build final frontier (March 2007), `http://www.msnbc.msn.com/id/17841125/`
2. Holden, K.: NASA dreams of an interplanetary 'Second Life' for mars crew. Wired (January 2008)
3. SecondLife, `http://www.secondlife.com`
4. David, L.: NASA ames' Second Life blends cyberspace with outer space (May 2007), `http://www.space.com/adastra/070526_isdc_second_life.html`
5. Taran. NASA in SecondLife: Plans for a synthetic world in 2007 (November 2006)
6. Hut, P.: Virtual laboratories and virtual worlds. In: Proceedings of the International Astronomical Union, 3(Symposium S246), pp. 447–456 (2007)

7. Hut, P.: Virtual laboratories. Progress of Theoretical Physics 164, 38 (2007)
8. Schechter, B.: Telescopes of the world, unite! a cosmic database emerges. The New York Times (May 2003)
9. Sloan digital sky survey, http://www.sdss.org/
10. Streiftert, B.A., Polanskey, C.A., O'Reilly, T., Colwell, J.: Science opportunity analyzer – a multi-mission approach to science planning (March 2003)
11. Stodden, D.Y., Galasso, G.D.: Space system visualization and analysis using the satellite orbit analysis program (soap), vol. 1, pp. 369–387 (February 1995)
12. JMARS, http://jmars.asu.edu
13. Solar System Simulator, http://space.jpl.nasa.gov/
14. The Visualization ToolKit (VTK), http://www.vtk.org
15. Satellite ToolKit (STK), http://www.stk.com
16. SPICE toolkit, http://naif.jpl.nasa.gov/naif/toolkit.html
17. Acton, C.H.: Ancillary data services of nasa's navigation and ancillary information facility. Planetary and Space Science 44(1), 65–70 (1996); Planetary data system
18. OpenSim, http://opensimulator.org
19. CoLab Virtual Overview - NASA CoLab, http://colab.arc.nasa.gov/virtual
20. Lopes, C., Kan, L., Popov, A., Morla, R.: PRT simulation in an immersive virtual world. In: SIMUTools 2008, First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems, Marseille, France (March 2008)
21. Ant simulation in Second Life, http://andrewcantino.com/sl/ants/
22. CDB Barkely. Heart murmur sim, assessment of learning in sl- interview with a man in a surgical mask (September 2006), http://sl.nmc.org/2006/09/25/jeremy-kemp/
23. Yellowlees, P.M., Cook, J.N.: Education about hallucinations using an internet virtual reality system: A qualitative survey. Acad. Psychiatry 30(6), 534–539 (2006)
24. Mesko, B.: Genetics in Second Life (April 2007), http://scienceroll.com/2007/04/11/genetics-in-second-life/
25. Project Wonderland, https://lg3d-wonderland.dev.java.net/
26. Project Darkstar, http://projectdarkstar.com/
27. Croquet Consortium, http://opencroquet.org
28. Schlyter, P.: Computing planetary positions - a tutorial with worked examples, http://www.stjarnhimlen.se/comp/tutorial.html