Generic 3-Dimensional Visualisation for Distributed Simulations

James Miller
Distributed Simulation Laboratory, University of Ballarat
jg.miller@ballarat.edu.au

Dr David Stratton
Distributed Simulation Laboratory, University of Ballarat
d.stratton@ballarat.edu.au

Abstract. Visualisation is a fundamental aspect of many simulations, providing a layer of abstraction between raw data and an interested human. This layer allows the user to quickly and easily parse the data and comprehend its significance. The lack of any generic visualisation tools has seen the widespread use of visualisation applications designed specifically for individual simulation implementations. The effort required to develop such custom visualisations constrains the development of distributed simulation beyond relatively well-funded defence simulation purposes. A framework that provides generic visualisation capabilities to an arbitrary simulation can potentially relax that constraint. Arguably, the majority of current implementations of distributed simulations model real-world situations that take place in a three-dimensional space. Further, the interest currently shown in distributed simulation architectures by new industries such as video game producers demonstrates the growing demand for 3D visualisation. Therefore, a generic visualisation framework should provide a configurable 3D viewer. This paper describes a conceptual generic 3D visualisation tool that consumes the specialised data provided by a separate visualisation framework that is the subject of a related project.

1. INTRODUCTION

Visualisation is a vital part of many simulations. It is accepted that visualisation takes advantage of the highest bandwidth connection to the brain [15]. The ability to present potentially huge datasets in a visual form allows the user to quickly interpret and understand them, and make decisions. Many simulations model objects that exist in a 3-dimensional space. Being able to "enter" that 3D space and view and interact with the objects within it is an effective way to understand a simulation and obtain results.

A review of existing 3D simulation visualisation tools showed that most are geared towards a single constrained domain. In the past, this was not a problem, since the majority of distributed simulations dealt with military scenarios that were each similar enough that non-generic tools could be easily adapted to accommodate any differences.

With interest growing in the use of the High Level Architecture (HLA) in other domains, such as transport, medicine and entertainment [9, 13, 11], these restricted visualisation tools are no longer applicable without major programmatic modification. There exists, therefore, a need for a highly configurable, generic visualisation tool that can be easily adapted to provide useful 3D visualisation of a range of simulations.

To address this need, this paper describes a conceptual system that provides generic 3D visualisation of distributed simulations. It first defines the functionality that must be included to achieve the aim, and then illustrates how this functionality will be brought together in an integrated design.

2. 3D VISUALISATION AND VIEWERS

Most simulation visualisation tools that include a 3D viewer component are expensive, commercial tools that require the use of a specific and immutable object model. Further, the majority of those are designed for use with military simulations alone. Some examples follow of systems containing such viewers.

2.1 ModIOS

General Dynamics' ModIOS system includes a stealth 3D viewer [8]. The 3D viewer is supported by an extensive framework that provides numerous tools expressly intended for controlling and monitoring a military scenario.

To attain some degree of generality, it relies on the Real-time Platform Reference (RPR) FOM, an industry standard object model specification. The RPR-FOM is also intended and optimised for military simulations, however, and is thus usually unsuitable for business or medical simulations, for example.

2.2 SVS Stealth

Like ModIOS, Advanced Interactive Systems' SVS Stealth 3D viewer is aimed at the military simulation industry [1]. It is highly customisable, and provides a slew of interactivity features, including camera attach modes and an "Individual Combatant" mode, in which the user navigates the scenario with the same constraints a foot soldier would experience.

2.3 JView

JView is a “3D and 2D, runtime configurable and platform independent visualisation program and API” [2]. Developed for the US Air Force, it aims to be a generic visualisation system that provides multiple views. Its extensibility is supplied by plug-in modules that can be written by programmers to handle new data types.

2.4 Major problems with these viewers

ModIOS and SVS Stealth, as well as numerous other visualisation tools, rely on the simulation conforming to—or at least being translatable into—a specific object model. This is fine when most simulations naturally correspond to this object model; however no universal standard has yet been developed that accounts for all possible objects used in any possible simulation. In order to visualise simulations outside this confined domain, a generic viewer must by some means support any relevant object model.

JView is designed for its ability to display multiple views in different formats, rather than for generality in object model specification. To support different simulations or models, a programmer must write plug-ins. This approach requires that simulation designers must also be programmers. Such a situation cannot be guaranteed, especially given the effort to introduce distributed simulation into fields where users are more accustomed to visual, drag-and-drop style design. The generic viewer must therefore also provide its support of different object models by means of configuration files that can be manipulated within a visual designer.

3. CONSIDERATIONS

Numerous issues must be considered in the creation of a generic 3D visualisation system. In this investigation, four major factors are identified: the viewer must

1. display the objects that are simulated in a 3D space;
2. depict an environment around the simulation objects;
3. use configuration files to specify both the objects and environment, rather than through reprogramming; and,
4. provide an interface that enables users to obtain useful information from the visualisation.

3.1 Simulation Objects

The main function of a simulation visualisation tool is to display the objects within the simulation in such a way that they impart useful information to the user. This involves portraying the object with a relevant and accurate 3D model. When displaying the simulation objects, most existing visualisation tools include enhancements that enable users to track object paths and view detailed information on their current state.

3.1.1 Object-to-model mapping

Since—at least in the case of the HLA—all simulation objects are explicitly specified in the Federation Object Model (FOM), objects can generally be mapped to 3D models directly from the FED file (which defines the FOM). As will be shown in section 3.3 below, support for dynamic acquisition of models is a technique that could help to reduce the amount of manual configuration that simulation designers are required to perform.

3.1.2 Network Latency

Real-time 3D visualisation requires at least 24 frames per second for the animation to be perceived as smooth [6]. Since there may be latencies in data transfer over a network from a simulation to the viewer, it is unlikely that this ideal can be upheld without some system that accounts for any lapses. One method of overcoming this is called *dead reckoning* [3].

Dead reckoning involves predicting the possible future location of an object, and using that prediction in lieu of actual data. Once the actual location has been discovered, the visualisation can be updated to reflect any difference. If the object is immediately relocated, however, it creates a jumpy and stuttered visualisation, thus defeating the purpose of the dead reckoning. An alternative is to employ a technique called *linear convergence*, whereby once actual data is received the object is gradually brought back into line with its actual position. The degree of positioning error created by dead reckoning is greater the less predictable the movement of the object. For certain simulations, any errors in the visualisation may negatively influence conclusions. In other cases, though, it is more useful for the user to experience smooth animation [6].

3.1.3 Articulated and Animated Models

A simulation may have two objects that in a 3D representation must be somehow joined together. For instance, a tank object with a turret direction attribute should be displayed with a turret that can rotate independently. Here the attributes may either be mapped to different components of a single model, or to two separate models that are “welded” together by the viewer. When “welded”, the relative position and distance between the two models would be maintained.

Additionally, certain objects might perform actions that should transform their model, or even change their representation completely. This change may be either permanent or temporary. For instance, a simulated tank may fire its cannon. This action should be represented in the visualisation in a visible and obvious fashion: in this case, with a bright flash from the barrel.

3.2 Environments

The major obstacle faced when designing a generic 3D visualisation tool is not displaying the objects within a federation; rather, the environment in which the objects exist that is outside the bounds of the simulation poses the greatest challenge. For many existing simulations, a representation of terrain and sky will suffice; indeed, this would cater for most military, transport and gaming simulations. To create a truly generic system, however, other potential environments must be considered. For example, a simulation set in outer space, whilst still a real world environment, has many different visualisation requirements. Even more exotic environments can be imagined for medical simulations that might study anything from cellular-level interactions, to human physiology as a whole [13]. At the other end of the spectrum, simulations such as mechanical testing may need no more environment than a black void.

3.3 Configuration

To uphold the aim of generality, the viewer will require relatively extensive configuration for each simulation it must visualise. However, the benefits of this approach are that: a) programming skills are not required to adapt the viewer to new simulations; and b) much of the configuration can be automated within a visual designer.

Any configuration effort will be focussed on three things: translating simulation object types to types suitable for visualisation, mapping simulation objects to models, and specifying the nature of the 3D environment. As discussed in section 4.1 below, reference [7] details a system that obtains data from a distributed simulation, and translates it into a form that is specific to a visualisation tool.

The requirement for simulation designers to pre-configure object-to-model mapping could be lifted by using automatic model acquisition. Work such as that discussed in [12] has investigated the categorisation of 3D models using detailed meta-data. This work expanded upon the Environmental Data Coding Specification (EDCS) that formed part of the DMSO's SEDRIS project. Although this work focussed on objects that are present in the real world, similar standards for other types of 3D models may be developed in the future. Given this meta-data, the viewer could automatically access a central database of uniquely identified models when a new object is encountered, and download the associated files.

3.4 Interface

The user interface is the most important aspect of the viewer, since it is this component that facilitates the user's access to the visualised data. The primary purpose of the visualisation tool is to allow the user to comprehend and understand the relationships within a large dataset. The design for the user interface must therefore support the user in this process. Numerous techniques exist that assist in providing a useful and accessible interface to users.

3.4.1 Dynamic functionality

Since interactivity requirements may change depending upon the type of simulation visualised, the interface functionality may also need to change. For example, the ability to track an object's movement might be useful in a large-scale military simulation, but may not be needed in a smaller-scale medical model. In such a case, removing access to the tracking functionality for users of the medical simulation would reduce screen clutter and improve useability.

Various methods for defining user interfaces have been devised. One such system is called Extensible User Interface Language (XUL) [5]. This technology, based upon the Extensible Modelling Language (XML), allows developers to define a user interface in a non-platform-specific manner. Work is currently taking place on a runtime environment for XUL applications; however this outcome is still some time away. Since XUL uses simple formatted text files for interface specification, its adoption would allow simulation designers to easily redefine existing interfaces or design new ones.

3.4.2 Navigation and depth perception

Navigation through a 3-dimensional space should come easily given that humans exist naturally in such a world; however using 2D input devices makes for challenging navigation [10]. The common human-computer interface (HCI) devices—the mouse and keyboard—are not well suited to 3D navigation, since they were designed for 2-dimensional interaction. Whilst research has been conducted that examines alternative HCI technologies, such as gloves and VR headsets, as of yet none has come into common, widespread use. Furthermore, such hardware is expensive, restricting its use to well-funded projects. The viewer must therefore use techniques to improve user control given the limited available input functionality.

The output generated on a monitor is often referred to as 2.5D [4], since the image only gives the impression of 3D space. This means that users cannot use their brain's depth perception abilities—it is like viewing the world with one eye closed. Many existing tools include support for true 3D via special hardware, such as stereoscopic glasses. These allow users to view computer-generated 3D spaces with the benefit of true depth perception. Provision of the ability to interface with these devices is important, from red-and-blue stereoscopic glasses, to high-tech headsets.

4. DESIGN OF SV3D

To fulfil these requirements, a visualisation system dubbed Simulation Viewer 3D (SV3D) has been designed. The SV3D system will provide the visualisation functionality that existing 3D viewers present, whilst being able to display relevant data from *any* simulation.

The SV3D viewer is based upon an existing framework that allows it to access simulations. The system will initially be a stealth viewer—able only to view simulations, not manipulate them—but it is designed such that it will be easy to integrate interactive features when the necessary infrastructure becomes available.

4.1 NOVA

The Nova Open Visualisation Architecture (NOVA)[1] framework provides generic visualisation data to clients [7]. It uses a highly configurable system that is able to convert simulation data into a form that is useful for specific visualisation systems.

NOVA accesses distributed simulations using software called fedWS[2] [14]. FedWS is designed to provide read-only access to a private HLA Federation through a web service interface. Rather than adding a new viewer federate for every client, it becomes highly scalable by allowing multiple remote clients to access the federation via a single static proxy federate.

Because fedWS makes its data available by means of a web service, a bottleneck may exist where bandwidth is limited over an Internet connection. NOVA attempts to compensate for this by creating its own object store that caches federation data. The cache reduces the amount of traffic that must pass between it and the simulation by drawing upon the stored data when providing updates to clients.

The SV3D viewer will use NOVA and fedWS to relay simulation data from the distributed simulation, and to convert the data into a form that is relevant and understandable.

4.2 Configuration

SV3D will need configuration by simulation designers to work with different simulations. The extent to which this must take place depends on the degree of outlandishness of the object model. As will be shown, various techniques are used to avoid unnecessary steps in the definition and design of a visualisation.

4.2.1 Configuring NOVA

NOVA allows visualisation tool designers to specify the simulation inputs required, and any data transformation that must take place for simulation data to become understandable to the viewer. The NOVA framework uses a transformation pipe system that performs any translations of simulation data by means of user-definable methods. These methods may, for example, perform unit conversion, offsetting or scaling of data.

[1] For more information, see http://hsv.littlebluefrog.com/

[2] For more information, see http://fedws.littlebluefrog.com/

4.2.2 Configuring SV3D

SV3D will use XML files to configure its operation and interaction. These can be edited manually, allowing full control over the visualisation, but this process is potentially time-consuming, so it is planned that SV3D will have an inline scene composer. This will facilitate rapid design practices, such as combining pre-existing elements visually to create a new environment. This allows simulation designers to ignore menial tasks that can be handled automatically.

It is likely that most object-to-model mapping will be conducted at design time, with appropriate models selected manually by the designer. As mentioned in section 3.3 above, however, efforts to design a framework for a 3D model acquisition system have been made. In SV3D, model acquisition is expected to occur at three times: 1) during the design process, where FOM data can be used to establish model requirements; 2) when a simulation is first viewed on a client machine and model files have not been supplied by the simulation designer; and, 3) a federation that supports some form of FOM agility may allow new object types to be merged dynamically. In each case, until a suitable model can be obtained, a temporary avatar such as a simple box will be displayed. Unfortunately, automatic model acquisition is not yet viable given the lack of a well-stocked, publicly accessible database. SV3D is designed such that this functionality will be easy to incorporate at a later date.

4.3 Use-case scenario

As an example, say a simulation designer was working with a simulation of race cars driving around a track. In this simple scenario, each car has direction and speed attributes, and races on a flat, oval shaped track. The designer's first step is to set up fedWS and NOVA so that simulation data is transformed into relevant data for SV3D. Next, they locate appropriate 3D models to represent the cars, and then, through the configuration files, map these models to objects or attributes. They would then create the track by integrating 3D primitives (e.g. cubes or spheres) with custom models using SV3D's inline scene composer. For this example, this involves creating the track surface with primitive planes, and adding pre-existing tree models and a sky. Once this is completed, execution of the simulation could begin. SV3D displays the car models on the track (accurately thanks to NOVA's transformation pipe), and moves them as their simulated position changes. If the designer specifies dead reckoning, SV3D would predict the next location of each car depending upon its previous actions. As the simulation runs, the viewer position can be moved around to get different angles on the action.

4.4 The SV3D Architecture

SV3D will be constructed with a modular approach, as shown in Figure 1: below, allowing for individual components to be easy to replace as old technologies become outdated or new functionality is required.

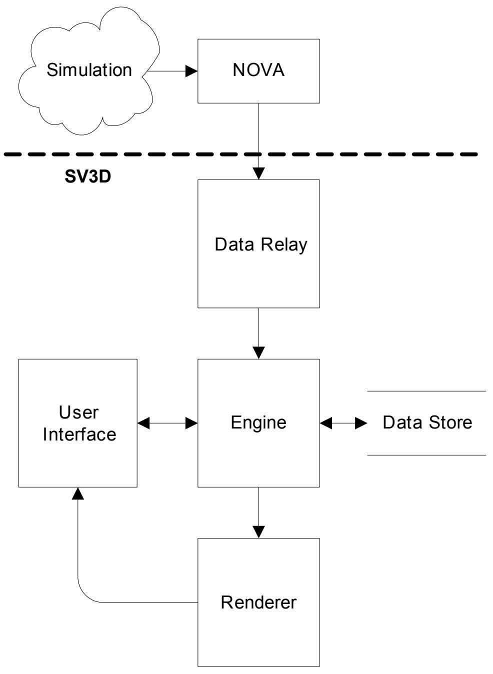

Figure 1: The SV3D architecture.

4.4.1 Engine

The Engine is the central organiser. It is responsible for processing incoming data, obtaining necessary support files such as 3D models or terrain from databases, and coordinating data flow amongst the other components. The Engine also intercepts data between the Data Store and Renderer, and performs any further transformations required, such as dead reckoning.

4.4.2 Data Relay

The Data Relay module is responsible for obtaining the simulation data from a provider, which will in this case be the NOVA framework. This module is passive, simply exposing an interface and shipping data to the Engine module and into the Data Store.

Separating this component allows for the support of different communication methods. For example, one version may support an XML-based transfer such as the Simple Object Access Protocol (SOAP), another may implement a proprietary TCP protocol, and another yet may use UDP for high-speed, unreliable transport. Furthermore, the ability to replace the Data Relay lends the system data source independence. Were it the case that NOVA was deemed unsuitable for a particular situation, a new Data Relay module could be written that interfaced with a different data provider. It is also possible that a Data Relay implementation could draw its data from multiple sources. If, for instance, by changing the User Interface module to include a 2D map view, the Data Relay module could also be modified to accept data from another NOVA view server that supplied specific map data. These modification techniques would require programming skills, however, and are thus not intended for regular use by designers.

4.4.3 Data Store

The Data Store's primary function is to keep track of the current state of objects, but it also provides a limited caching service that supports the dead reckoning functions. This involves storing at least the previous update’s object state. Objects that do not have dead reckoning enabled need only have their current state maintained.

4.4.4 Renderer

The Renderer is the component that creates the 3D visualisation. It must be able to provide a useable and visually appealing representation of the supplied data, including optimisation for large-scale, high polygon-count scenes. The Renderer must allow the user to dynamically reposition the view cameras.

The Renderer ideally must provide support for as many commonly used 3D model data formats as possible. Numerous data formats are standard within the simulation community, such as MultiGen’s OpenFlight, Terrain Experts Incorporated’s TerraPage, and SEDRIS. In order to increase the range of available 3D models, other widely used data formats such as 3D Studio Max, MD2, Wavefront and LightWave should be supported.

4.4.5 User Interface

The User Interface provides a graphical control mechanism for human interaction with the viewer. At start-up, it will provide a graphical interface for initiating communications between the SV3D viewer and the simulation. Once the viewer is connected, the interface will present users with tools that modify and interact with the visualisation, allowing them to change their viewing position, or obtain information about a particular object.

The interface gives the user access to controls that modify the camera position and orientation. It is important that these controls be useful despite the complications inherent with 3D navigation using 2D input devices (as exposed in section 3.4.2 above). To automate navigation to some degree, the User Interface component will allow the user to constrain camera movement in various ways, as is done in other visualisation tools. This includes attaching the camera to an object and following it, restricting altitude or distance from an object, and a first-person view from an object.

4.5 Implementation

The SV3D software will be developed using existing tools and libraries where possible, and preferably with open-source or free software. It will use Object Oriented development processes to facilitate the modularity of the framework. The system will make use of platform-independent coding techniques and libraries so that it will work on as many architectures as possible. The resulting software will be released under an open-source license and be freely available for anyone to use and modify.

5. STATUS & FUTURE WORK

Development is currently taking place on a prototype implementation of the SV3D viewer, following the guidelines set out in this paper. Initially, the development will focus on subsets of the grander design. Later versions will include support for all the features specified in the original design.

Whilst this prototype will be purely stealth, future versions of NOVA and fedWS may allow interaction between the client viewer and the simulation. This would allow for interesting possibilities such as the creation of a game using SV3D as the renderer and user interface.

For more information on the current status of SV3D, please visit the project website at http://hsv.littlebluefrog.com.

6. CONCLUSION

There is a lack of generic visualisation tools for distributed simulations. Whilst numerous 3D visualisation systems exist, they are all restricted to viewing a specific domain, predominantly military scenarios.

This paper has described the requirements and presented an outline of a generic 3D visualisation tool for distributed simulations. It is designed so that any simulation that models relevant situations may be viewed without reprogramming an existing viewer or the simulation itself.

REFERENCES

1. Advanced Interactive Systems (formerly Reality by Design). (2001) "RBD Stealth Pro Product Information." Orlando, Florida [Online: http://www.ais-sim.com/svs]

2. Air Force Research Lab / IFSB. (2003) "JView Website." [Online: http://www.rl.af.mil/tech/programs/JVIEW/]

3. Batista, H.; Costa, V. & Pereira, J.M. (2001) "Games of War and Peace: Large Scale Simulation over the Internet." *Seventh International Conference on Virtual Systems and Multimedia*. Berkeley, USA [Online: http://vasc.home.sapo.pt/]

4. Blocher, T.W. & VonPlinsky, M. (2002) "Information Visualization in a Distributed Virtual Decision Support Environment." *Proceedings of the 2002 Spring Simulation Interoperability Workshop.* [Online: http://www.sisostds.org] 02S-SIW-119

5. Bojanic, P. (2003) "The Joy of XUL" [Online: http://www.mozilla.org/projects/xul/joy-of-xul.html]

6. Cheshire, S. (1996) "Latency and the Quest for Interactivity." *Commissioned by Volpe Welty Asset Management, LLC, for the Synchronous Person-to-Person Interactive Computing Environments Meeting*. San Francisco. [Online: http://www.simoncheshire.org]

7. Fraser, M.R. & Stratton, D. (2004) "A General Purpose Visualisation Architecture for Distributed Simulation." *Submitted to the 2004 SimTecT Conference*. Canberra.

8. General Dynamics. (2003) "ModIOS System Overview." [Online: http://www.gd-decisionsystems.com/modios/]

9. Klein, U.; Schulze, T.; Strassburger, S. & Menzler, H.-P. (1998) "Distributed Traffic Simulation based on the High Level Architecture." *Proceedings of the 1998 Fall Simulation Interoperability Workshop.* Orlando, Florida. [Online: http://www.sisostds.org] 98F-SIW-016

10. Li, T.-Y. & Ting, H.-K. (2000) "An Intelligent User Interface with Motion Planning for 3D Navigation." *Proceedings of the IEEE Virtual Reality 2000.* [Online: http://citeseer.nj.nec.com/li00intelligent.html]

11. Loughran, J. (1999) "Simulation Across the Spectrum: The Island of Entertainment Simulations." *SISO News*. Volume 1 Issue 4. [Online: http://www.sisostds.org/webletter/siso/iss_18/art_133.htm]

12. Miller, D.D.; Janett, A.; Salemann, L.; Farsai, S.; Miller, E.Y. & Birkel, P.A. (2001) "An Environmental Data Model for 3D Models," *Proceedings of the 2001 Fall Simulation Interoperability Workshop*. Orlando, Florida. [Online: http://www.sisostds.org] 01S-SIW-071

13. Murphy, S.; Coolahan, J.; Lutz, R.; Saunders, R.; Kovalchik, J. & Feldman, A. (2003) "Human Physiology Simulation Integration Using the HLA: ExerFed1516." *Proceedings of the 2003 Spring Simulation Interoperability Workshop*. [Online: http://www.sisostds.org] 03S-SIW-092

14. Pokorny, T. (2002) "Dynamic Web Access to Active HLA Simulations." Honours Thesis. University of Ballarat: Australia.

15. van Dam, A.; Forsberg, A.S.; Laidlaw, D.H.; LaViola Jr, J.J. & Simpson, R.M. (2000) "Immersive VR for Scientific Visualisation: A Progress Report." *IEEE Computer Graphics and Applications*. November/December. 26-52.