*Proceedings of the 1999 Winter Simulation Conference*
*P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds.*

# WEB-BASED SIMULATION VISUALIZATION USING JAVA3D

Chad F. Salisbury
Steven D. Farr
Jason A. Moore

Air Force Research Laboratory / IFSB
525 Brooks Rd
Rome, NY 13441-4505, U.S.A.

## ABSTRACT

The coupling of Java3D and applet technologies has the potential to revolutionize web-based simulation visualization. Applets can enable the dynamic and distributed instantiation and elimination of viewers that until now was not possible. A visualizer based on these technologies is under development at the Air Force Research Laboratory's Information Directorate. Unlike existing visualizers that must be running at the simulation's start time, this tool allows users to come and go as they please - effectively allowing one to peer into the state of a simulation at a place, perspective, and time that is of specific interest to them.

Intense 3D graphics have been difficult to program and distribute among heterogeneous environments until the inception of Java3D. Sun Microsystems' Java3D provides users the best acceleration their platform can support while the WWW provides the transmission and communication infrastructure. This paper discusses a government owned, browser-based simulation visualizer capable of displaying simulated entities to any number of distributed sites.

## 1 MOTIVATION

Historically, simulation visualizers were stand-alone applications running on a single workstation. People interested in monitoring a simulation using most existing visualization tools must be present at the inception of the simulation so that the visualizer can initialize itself with the simulation. If the starting time is missed, the real-time progress of the simulation can not be observed. This scenario was the case with the Distributed Interactive Simulation (DIS) paradigm. Preferably one should be allowed to 'peer' in and out of the execution of a simulation as desired.[1] It is often the case that one is only interested in observing a relatively small time slice of the simulation. If this time period of interest occurs near the end of the overall simulation the observer will have to join the simulation at its very onset, tying up both the machine and the user for the duration of the simulation.

A second motivation is the desire to easily bring visualization to those who only wish to observe the simulation execution. Participation in HLA exercises requires extensive off-line involvement prior to the actual running of the simulation. Visualization packages must first be made HLA compliant (i.e., be able to communicate with the HLA runtime infrastructure) and then must be folded into the interface specification. Distributed passive observation of a simulation without the burden participating in the HLA exercise 'set-up' process should significantly increase the number of users willing to analyze simulation results.

Lastly, legacy visualizers were constrained to execution on the machine for which the application was written. When systems were updated, often the simulation viewer had to be at least partially modified to get it to run on the new configuration. Inherent to the machine dependent problem was the requirement to maintain a separate implementation for each of the architectures the visualizer supports. The number of source code variations expands rapidly, and the application gets exponentially difficult to support.

## 2 BACKGROUND

JView is a modular program designed to assist engineers and analysts with visualization of models and simulations. The core segment of the program relies on Sun Microsystems' Java 3D architecture to render an image. In order to support the simulation community's diverse set of visualization requirements, JView allows one to create Java Beans that interface with the main viewing window. These small pieces of code enable JView to display objects from any data source (i.e., simulation). A list of viewable objects would include not only aircraft, tanks, and ships, but also radar coverage, sound, thermal coverage, and other non-visible phenomena. The basic JView package is

provided with a core set of beans that allow reading and replaying of simulation input data. Whether this data is real-time through DIS or HLA or recorded to an input file, JView can display the forces and energies depicted in any given simulation.

JView-lite leverages the same display technology but relies on the Web for distribution. Unlike JView, JView-lite will be a holistic applet that provides a window into a simulation (i.e., one cannot develop beans to work in conjunction with the JView-lite applet). The only simulation feed available to JView-lite will be provided from an HLA simulation. JView-lite won't have the plug and play capabilities of JView. However, it will utilize the same graphics engine as JView. It will also support all of the same viewing and scene manipulation features as its big brother.

## 3 SYSTEM ARCHITECTURE

The client server architecture depicted in Figure 1 below has two principal software components: the Simulation Monitor and the Simulation Viewer (a.k.a. JView-lite).

The Simulation Monitor is an HLA-compliant module (i.e., a federate in HLA-speak) which has registered to receive position data from all the objects in the federation. The monitor is initialized upon startup of the HLA exercise and participates in the simulation as an unobtrusive observer for the duration of the exercise. The monitor passes object information to each client. The number of supportable clients is still under consideration and may be a function of the total number of viewed objects requested by the set of clients.

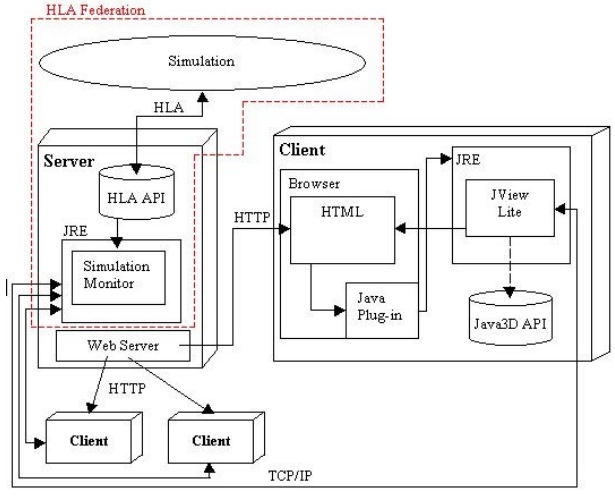


Figure 1: Web-Based Simulation Visualization Architecture

The JVIEW-lite simulation viewer is an applet executing within the client's web browser. Three-dimensional rendering is enabled using the Java Plug-in and the Java3D API (more on these later). The applet allows the user to view the objects in the simulation from any desired perspective. It is also possible to launch multiple browser windows, each with their own applet, to allow for multiple views into the same simulation. Currently, the clients are provided all the information necessary to display all of the objects in the simulation.

The viewing process starts when the client browser requests, via HTTP, to view a simulation. The web server hosting the link will send the web page and the embedded viewer applet. The browser then initializes the viewer applet, effectively making the browser a 3D-simulation viewer. The viewer, in turn, makes contact with the simulation monitor that is residing on the same machine as the web server. Although not necessary, we have elected to have this component run off the web server to ease communication with the clients. Housing the monitor and the web-server on separate machines requires signing and validating the viewer classes so the applet can communicate with a system other than its originator. Once the connection is made, the simulation monitor brings the viewer up to speed by informing the viewer about the number, locations, and vector information of all the simulation participants. The simulation viewer can then render the simulation based on this information.

When a viewer is closed down, a message is sent to the simulation monitor to stop the transmission of information to that particular client. If the simulation should end while viewers are still connected, the simulation monitor will inform those clients that the simulation has ended and no further information will be sent. The simulation monitor will then shut down.

## 4 WEB-BASED 3D GRAPHICS

Our approach was to use an existing 3-D graphics standard as the means to visualize simulations within a web browser. This venue was deemed more efficient and cost-effective than implementing a proprietary or, in our case, government-owned, 3D renderer. It also allowed for a shorter development cycle. Support from the API's creator in terms of bug fixes and incremental improvements was also a strong incentive. An API that has corporate backing and a large user base often sees much more in the way of enhancements and innovations than a proprietary, government owned product that is only used within one system.

The API of choice was Java3D. This API was attractive because it has strong corporate backing and a large user base, providing strong indicators that the product should have a long and prosperous lifecycle. It is also built upon Java, a language that focuses on portability and web-based architectures. None of this should discount the API's power and relative ease of use, which were also factors in the decision.

### 4.1 Showing 3D Content on the Web

Java3D is fairly new and is not part of the standard Java language distribution. It is an extension, an additional part of Java whose classes have to be downloaded separately from the standard Java distribution. There are a couple of reasons for these extensions. One is to keep the standard Java distribution from growing exponentially large with a multitude of options, avoiding "kitchen sink syndrome". Incorporating every new idea into the language would fast make Java a bloated, unwieldy beast that would be impossible to download and use. Extensions allow users to utilize only those extra parts of Java they need. The second reason is that the Java3D API utilizes some hardware level graphics facilities for speed. Access is via OpenGL support built into the video card (There is a beta release of Java3D utilizing DirectX for video cards that support DirectX). These calls aren't supported by all video cards, making Java3D somewhat less portable than most of Java.

Creating 3D applications using Java3D is fairly straightforward matter; but the creation of 3D applets that run within a web page is not. Java3D is a new and changing technology that is ahead of most of the Java interpreters incorporated in today's web browsers. As of this date, Netscape's Communicator and Microsoft's Internet Explorer can't support all the new features being added to Java. Fortunately, Sun Microsystems provides a mechanism for running Java code through the browser using the latest and greatest in Java interpreters. This is done with the Java Plug-in.

### 4.2 The Java Plug-in

The Java Plug-in comes with the Java Development Kit (JDK 1.2) and now with the JRE1.2.1. The Java Plug-in works with Netscape Navigator version 3.0 or greater. It also works with Microsoft Internet Explorer version 3.0.2 or higher [see Java Plug-in web site reference]. Both utilize the plug-in in a slightly different manner; our focus is restricted to the Netscape browser since it is currently the browser used at the Air Force Research Laboratory.

The Java Plug-in (Sun Microsystems 1999a) utilizes Netscape's Plug-in technology to run the JRE (Java Runtime Environment) within the browser, bypassing the browser's native Java interpreter. It doesn't replace the native interpreter, the JRE is just called in place of the native interpreter. This is accomplished by changing the HTML in which the applet is contained. Netscape uses the <EMBED> tag to indicate that the browser should use an

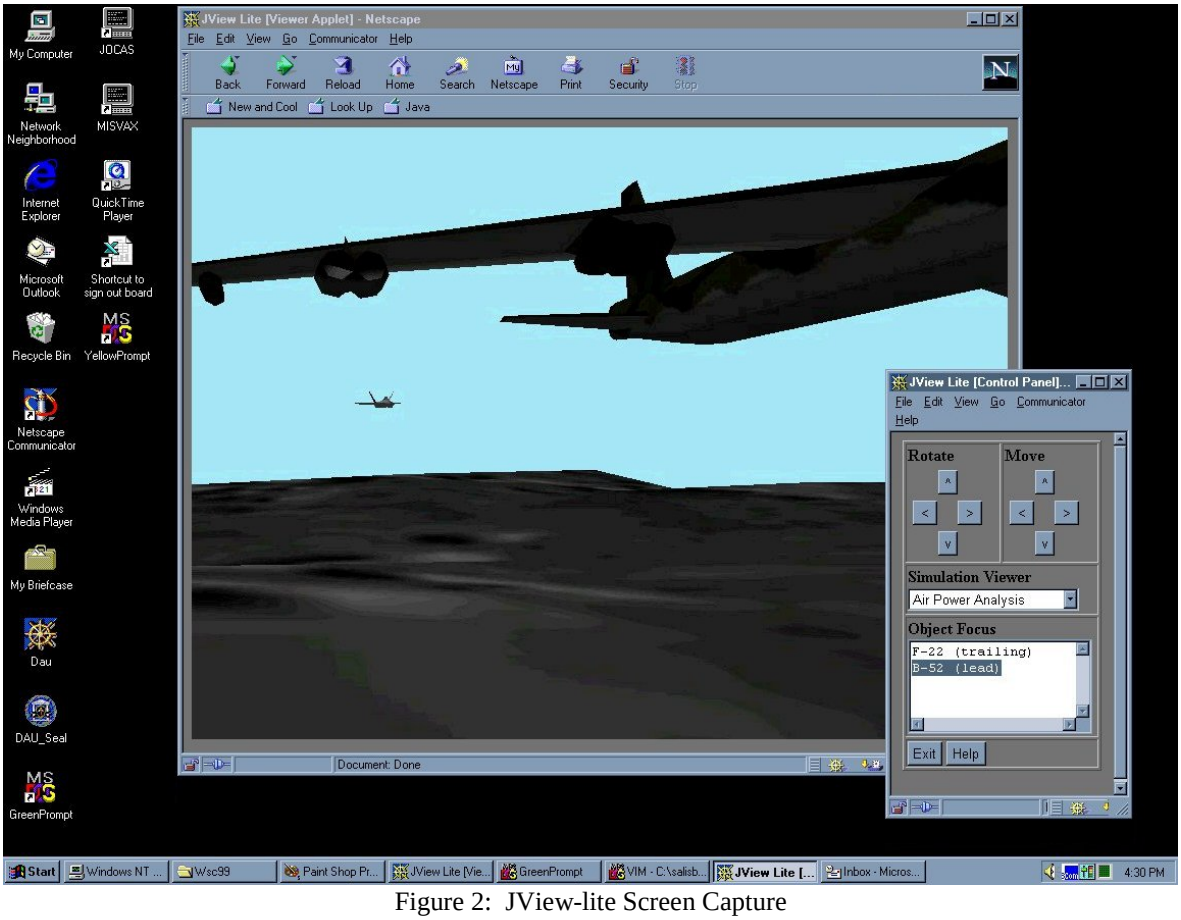

Figure 2: JView-lite Screen Capture

alternate JRE. (Sun Microsystems 1999b) In addition, Sun provides an HTML Converter that handles the HTML conversion automatically. This is particularly useful if there is a significant number of existing web pages that need conversion.

Installing the Java Plug-in in Netscape is simply a matter of pointing the browser at a converted web page. The plug-in will download and install automatically (upon user authorization). From this point on, any pages requesting use of the Java Plug-in will run whatever JRE the Plug-in is configured to use. An alternate method of installing the Java Plug-in is to download and install the JDK or JRE manually, since both now come with the Java Plug-in. The browser now has the capability to run any Java classes corresponding to the Java 2.0 language specification.

### 4.3 Additional System Requirements

Currently the Java Plug-in requires Windows95/98/NT 4.0 or Solaris operating systems. A 90 MHz Pentium processor, 25 megabytes of disk space, and 16 megabytes of RAM are also necessary. Sun recommends at least 24 megabytes of RAM for better performance.

### 4.4 The Java3D Classes

At this point, the browser is capable of supporting Java 2.0. However, the JRE that the plug-in is using has to be able to handle the Java3D API calls made by the JView-lite applet. This requires installing the Java3D API on the machine which the browser, and subsequently the JRE, is executed. The API can be downloaded from the Sun Microsystems website and installed under the JRE that the Java Plug-in is using. Once the Java3D classes are within reach of the JRE, any calls an applet makes to them will be completed.

### 4.5 Providing 3D Content

Now the browser is capable of displaying 3D content. Providing that content however, requires a bit more effort. First, the code has to be written to create the three-dimensional world that the browser will be viewing. The resultant classes must then be accessible from a web server and called from an HTML document. The document must specify that the Java Plug-in is to be used to run the applet (see above for details on how to do this). Once this is complete, it should be possible to point the Java3D-enabled browser at the appropriate URL and have a Java3D applet appear within the browser.

## 5 FUTURE WORK

Figure 2 on the previous page shows a screen capture of the current version of simulation viewer. While functional, the applet is slated for several enhancements that include:

(1) Client selectable object sets – In order to better facilitate user-specific needs, the application will be improved to allow each client to select the objects or classes of objects (e.g., fix-winged aircraft) that they would like to visualize.
(2) Geographically constrained views – Large-scale HLA exercises, especially at the campaign or theatre level, may play out of a geographically large area. We propose to modify the current viewer to allow user selectable geographic boundaries of interest.
(3) Object constrained views - Similar to the modification above, this would constrain the view to a region around a certain object. For example, the user may only wish to view aircraft within 200 nautical miles of a ground-based radar site.
(4) Object view – This option will allow user's to view from the perspective of the object (e.g., a pilot's eye view). Views such as this will allow for the incorporation of heads-up-displays or other object specific characteristics.
(5) Object characteristics – Currently only object position data is passed to the clients. Future versions will allow for the display altimeters, fuel gages, speedometers, etc.

## REFERENCES

Fishwick, P.A. , 1996. "Web-based Simulation: Some Personal Observations", *Proceedings of the 1996 Winter Simulation Conference*, Dec 96: 772-779.

Sun Microsystems, 1999a. Java Plug-in Product web page. http://java.sun.com/products/plugin/index.html

Sun Microsystems, 1999a. Java plug-in HTML specification web page, http://java.sun.com/products/plugin/1.2/docs/tags.html

## AUTHOR BIOGRAPHIES

**CHAD SALISBURY** has a Bachelors degree in Computer Science from SUNY Institute of Technology in Utica, New York. Past experience includes Web-based applications using Perl, Java, and C. He has also been a developer on a Network Management SBIR and has a network analyzer utilizing neural networks. He's currently working on

projects involving 3D visualization and distributed collaboration.

**STEVE FARR** is the Chief of the C4ISR Modeling and Simulation Branch, Information Systems Division, in the Air Force Research Laboratory's Information Directorate at Rome NY. Steve received his B.S. from the University of Connecticut in 1983, and M.B.A. from Rensaelear Polytechnic Institute in 1986 and an M.S. from Syracuse University in 1993. His professional responsibilities include the application of emerging technologies to specific modeling and simulation requirements. He has been actively applying his modeling and simulation research interests to numerous Air Force programs since 1984.

**JASON MOORE**, a recent graduate of the State University of New York at Binghamton, began his professional career with the United States Air Force Research Laboratory in 1998. Prior to receiving his Masters degree in Computer Science, he worked part time at the same research institution working on advanced displays and intelligent interfaces. While his full time appointment is with a different group of engineers, he carries his Human Computer Interface background with him. Most recently, Jason is working on increasing the amount of visual tools that are available to the Modeling and Simulation community utilizing inexpensive commercial off the shelf hardware.