Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

AN INTERACTIVE LAND USE VRML APPLICATION (ILUVA) WITH SERVLET ASSIST

Lee A. Belfore, II
Suresh Chitithoti

Department of Electrical and
Computer Engineering
Old Dominion University
Norfolk, VA 23529, U.S.A.

ABSTRACT

We summarize progress achieved on an interactive land use VRML application (ILUVA) with servlet assist. The purpose of this application is to enable one to take a virtual land area and add buildings, roadways, landscaping and other features. The application is implemented entirely using standard web based technologies to allow fairly universal accessibility. The Virtual Reality Modeling Language (VRML) is a programming language that describes three dimensional objects and defines interactions associated with these objects. In this work, we show how the interactive capabilities can be expanded by employing Java servlets for recording user actions and for restoring prior sessions. The Java servlets offer several powerful capabilities including enabling logging permanent records of user sessions, retrieval of prior sessions, and dynamically generated VRML.

1 INTRODUCTION

The growth of the internet has cultivated a wealth of new technologies. Driving these technologies is the desire to enhance the way internet content is delivered to the user. From a superficial perspective, displaying a web page appears to be a simple matter of displaying text and images. With the introduction of programming capabilities offered by Java (Arnold and Gosling 1996), Javascript, and other programming languages, as well as new web standards such as XML (Bray, et. al. 1998), the web page is a dynamic entity. Paralleling the advances in web technology, advances in computing technology provide capabilities that were until recently beyond the reach of machines generally available. As evidence and in support of these advances, a 3D interactive visualization language, the Virtual Reality Modeling Language (VRML), has been standardized and is widely available (The Web3D Consortium 1998). VRML offers opportunities for demonstrating the value of distributing 3D web content over the world wide web (WWW). In this paper, we present progress on a methodology for delivering VRML based interactive 3D visualizations.

VRML has been applied in several visualization applications. A special issue of *IEEE Graphics & Applications* (IEEE Computer Society 1999) gives several interesting applications that illustrate the power of VRML. The applications range from entertainment (Matsuba and Roehl 1999), to terrain visualization (Reddy et. al. 1999), to simulation (Fishwick 1999). Our own work has been under development for some time and continues to evolve (Belfore and Vennam 1999). Technical challenges include creating an application that works well on typical machine platforms and also identifying those applications that can best benefit with the addition of a third dimension.

The primary contribution of our work summarized here is the development of a collection of Java Servlets that interact with and manage a VRML application. The Interactive Land Use VRML Application (ILUVA) provides a user with the ability to take a land area and populate it with buildings, roadways, and etc. In addition, ILUVA benefits from some of the unique capabilities offered by servlets. More specifically, the servlets provide three important capabilities. First, servlet calls make possible the logging of information generated by a VRML application. Second, servlets can restore the work from a prior session by retrieving information from the log file and emitting the appropriate VRML content. Third, servlets can be used to generate models specified parametricly.

This paper is organized into six sections, including an introduction, a brief tutorial on VRML, an overview of ILUVA, a discussion of interfacing VRML applications and Java servlets, a sample session and a summary.

2 VRML

A VRML application is typically a hierarchical collection of solid models, sensors, light sources, script methods, and grouping primitives assembled in a meaningful fashion. In

VRML, these primitives are termed nodes. Each node can have several fields to define node parameters and also to define inputs to (VRML `eventIn` fields) and outputs from (`eventOut` fields) the node. A node can define a solid model, such as a box; a capability, such as an interactive touch sensor; or arbitrary behavior, such as can be defined in a script. Model reuse is accomplished with prototype definitions that specify user defined nodes. The subset of the VRML nodes used in this application is discussed in this section. Several excellent VRML resources exist (Nandeau 1999; The Web3D Consortium 1998). The nodes are divided into different classes according to their purposes.

2.1 Shape Primitives

Shape geometries can be basic or general. Basic shapes such as boxes, cylinders, cones, and spheres are defined by name with fields to specify the dimensions. More general shape geometries include `Extrusion` and `IndexedFaceSet` geometries enabling specification of fairly general geometries. The `Extrusion` geometry is an analogy with the shape a material, such as modeling clay, makes after being forced through an opening with a varying cross section. The `IndexedFaceSet` defines the form of an the object as a collection of simpler polygons.

2.2 Grouping Nodes

The `Group` and `Transform` nodes are two grouping nodes used in this application. The `Group` node groups objects so that the entire child hierarchy can be easily referenced through the root node and can also be collectively associated with one sensor. The `Transform` node gives the same capabilities, but in addition allowing arbitrary scale, translation, and rotations of the child hierarchy. Nodes can be dynamically added and removed from the grouping nodes.

2.3 Sensors Monitor the World and Listen for the User

VRML has seven sensor nodes that generate events when the conditions to which they are sensitive occur. For example, the `TouchSensor` node generates an event when the mouse cursor is over an affected geometry and the mouse button is depressed. The `PlaneSensor` senses mouse drag actions while the mouse button is depressed. Using a `TouchSensor` in conjunction with a `PlaneSensor` enables a mouse click to select a geometry (`TouchSensor`) and a mouse drag to change the position of the geometry (`PlaneSensor`).

2.4 Scripts Implement Arbitrary Behaviors

The `Script` node enables the inclusion of arbitrary methods and behaviors implemented by the developer. The script receives input, does the required processing, and then generates the necessary output events. For example, a script can take events from a `PlaneSensor` that is tracking a mouse drag and convert this into an output event consisting of a set of vertices that defines another geometry.

2.5 Events Communicate Information

Events are generated and received by most nodes. An `eventIn` is used to receive input while an `eventOut` outputs information. For example, a `TouchSensor` node generates an event when a mouse click occurs on the target geometry. This event can be received by a script making a wire frame appear around the selected geometry. `ROUTE` declarations connect output events from one node to input events of another. Routes may be explicitly declared, or may be dynamically created and destroyed in script nodes. Events can be simple, passing a simple value, or complex, passing a node, enabling many interesting capabilities. Finally, it is possible to circumvent explicit routing by directly accessing node input and output events within a script method.

2.6 Prototype Nodes

`PROTO` declarations enable the definition of new node types that can be used in much the same way as the standard VRML nodes. The prototype can be defined either in the same file can be in a separate file declared in an `EXTERNPROTO` declaration. `PROTO` nodes are valuable for supporting model reuse and hiding implementation details.

2.7 The VRML Execution Engine

A VRML world can be viewed schematically as a scene graph where nodes describe geometries, scripts describe behaviors, and routes define information flow. The VRML execution engine is an integral part of VRML browser plug-in that "executes" the scene graph (The Web3D Consortium 1998). Consistent with the existing routes, the execution engine receives events from the scene graph and then delivers the events to the respective destinations, assigning a time-stamp along the way. Events that cascade from an initiating event during the same time delta receive the same time-stamp.

3 AN OVERVIEW OF ILUVA

The predecessor to ILUVA is described in Belfore and Vennam (1999). The primary enhancement has been to link the application to a web server. The server interactions are described in the next section. In this section, we describe the basic operation of ILUVA. ILUVA is a decentralized collection of eight interconnected, concurrently operating modules. A high level overview the ILUVA software architecture is given in Figure 1. These modules are 1) the

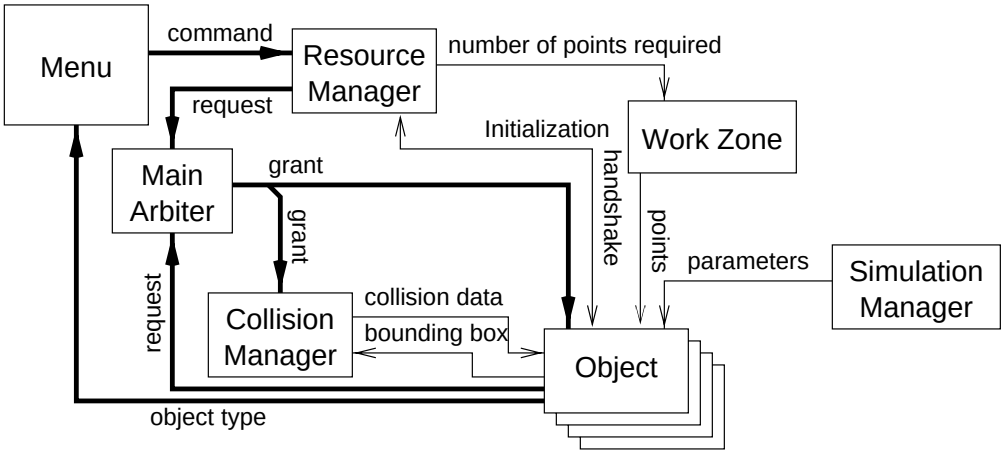


Figure 1: Overview of the ILUVA Software Architecture

Table 1: Major Events

Signal	Purpose	Source
`command`	user request	menu
`objectType`	send information about selected object, usually occurs after mouse click on object	object
`request`	object request for menu control	object, resource manager
`grant`	grants object menu control	main arbiter

menu, 2) the resource manager, 3) the arbiter, 4) the work zone, 5) the collision manager, 6) the fixed landscape (not shown in the figure), 7) a simple simulation manager, and 8) the object models. Object models are self-contained and encapsulate all object functional capabilities. In VRML, the information flow is event driven, with the major events are presented in Table 1. Menu events are encoded into commands that are broadcast to all modules and objects. When objects are selected, an `objectType` event is sent to the menu hierarchy to make the appropriate menu appear. The arbitration `request` and `grant` events are used to manage shared resources. When an object is selected, it sends a `request` event. When a requesting object is recognized, a `grant` event is broadcast to all objects that each monitors and responds accordingly.

The object architecture is designed to serve four purposes. First and most importantly, the object architecture is designed to be independent and autonomous, as much as possible, from the rest of the visualization. This enables objects to be managed and integrated cleanly in a modular fashion. Furthermore, the objects manage their own interactions with the user and the interface to the visualization. This makes possible objects having entirely arbitrary dynamic behavior independent of other elements in the visualization. Second, the object is implemented so that it is contained in and can be handled as a single node to simplify object management. The object, itself, is defined in a file containing a `PROTO` declaration for the object. An instance of the object is inserted into the visualization using a second file that contains an `EXTERNPROTO` declaration for the object followed by a single instantiation of the object. It is this coding formulation that guarantees that the instantiated object is contained in a single `SFNode` field when the second file is used to dynamically instantiate the object using the `createVrmlFromURL` browser method. Third, the object must be capable of reporting all changes in geometry and appearance to a web server. The monitor is the interface between the object and the server. The monitor takes updates and communicates the changes through servlet calls. Fourth, the node architecture enables static instantiation of the node as is required for restoring a prior session. Specific fields in the node prototype are used to both signal static instantiation and configure the object. Sufficient fields are included to recreate all information from a prior session.

In order to achieve these diverse purposes, the object architecture is partitioned into layers that isolate functions that interface the object to the visualization, from those that change the object geometry, and from those that define the pure behavior of the object. This organization permits modular design of the architecture and provides a straight forward framework for adding new objects with a variety of capabilities. As such, the object architecture is organized into three layers, the interaction layer, the edit layer, and the model layer, as shown in Figure 2. The interaction layer manages the interface with the visualization, static instantiation, and any user interactions that affect the object as a whole, such as object drag and rotation. The edit layer defines the nature and appearance of controls and provides an interface between the model and the interaction layer.

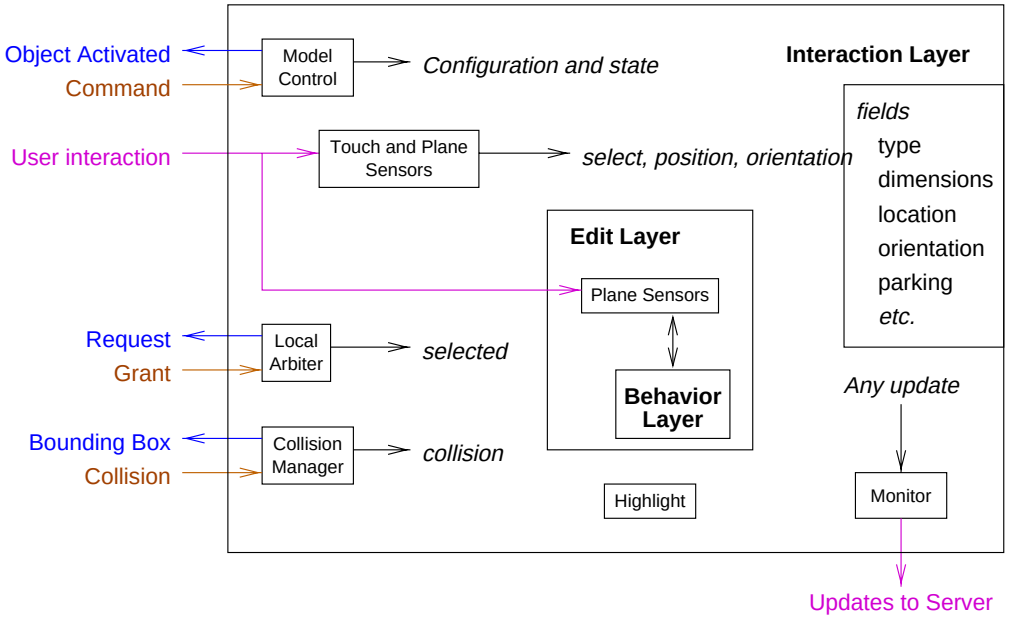


Figure 2: Object Architecture

At the lowest layer, the model layer, the appearance of the object is defined as is any characterizing behaviors, such as the ability to change appearance or shape. Presently, a new object can be included in the visualization by using templates for the interaction and edit layers. Indeed, the interaction layer is sufficiently well defined that it is automatically generated.

4 ILUVA AND JAVA SERVLETS

Java Servlets enable the integration of Java programs that run on the web server. The servlets can manage information from the client browser that reside on the server and can, in addition, produce web content of any type. In order to run servlets, the web server must be servlet enabled. For this project, we employed the Apache web server version 3.1.12 (The Apache Project 2000), SUN servlet Development Kit version 2.0 (Sun Microsystems 2000), and the Apache servlet engine Apache JServ version 1.1 (The Apache Project 2000). A high level view of the interactions is illustrated in Figure 3. Servlets were employed for three reasons 1) reading and writing files, 2) generating VRML to restore a session, and 3) generating a VRML object for a session.

First, the web browser has security limitations in that files on the client machine cannot be arbitrarily read and written. Furthermore, the server can be easily configured to read and write files on the server file system. The mechanism for logging a session requires interaction between servlets ECMAscripts in the VRML application. A simple method for passing information is for the VRML application to load the universal resource locator (URL) for the servlet. Arguments passed to the servlet URL supply the information to be logged. This is accomplished in the VRML application using the `loadURL` browser method. In this fashion, information from the VRML session is logged on the server. For example, we employ a `Record` servlet in this capacity. In ECMAscript, the following shows how a two dimensional coordinate stored in the `SFVec2d` variable `point` is passed to the server:

```
urls[0]='http://host/zone/Record?'+
        point[0]'+':'+point[1];
parameters[0]='target=updateFrame';
Browser.loadUrl(urls,parameters);
```

Additions and updates are recorded chronologically in the log file, with the most recent modifications representing the final appearance. Figure 3 illustrates the interaction between these processes. Note that the server and client machines are not necessarily different machines. For example, a stand alone system can be configured to run Apache while at the same time the client browser can request documents from the server. Table 2 lists the different log file record types. The format is straight forward and easy to parse. The *id* field is the serial number for the object and is used

Table 2: Log File Record Formats

Type	Format
New Entry	*id*:*type*
Update, simple	*id*:*type*:(*field*):-:*value*
Update, x,y	*id*:*type*:(*field*):-:*x-value*:*y-value*
Update, array	*id*:*type*:(*field*):-:*n*:-:$value_1$:-:$value_2$:-: $\cdots$:-:$value_n$

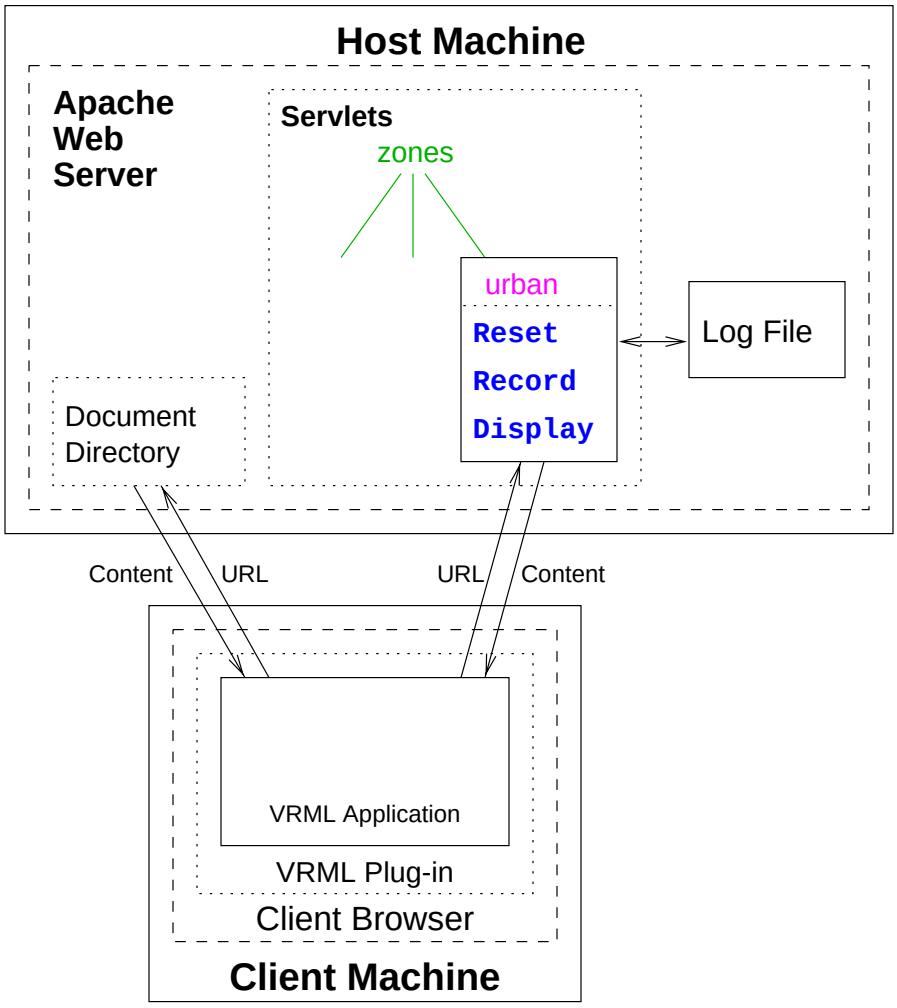


Figure 3: Interaction of Client and Host Machines

to associate subsequent updates to that object. The *type* field identifies what type of object is being manipulated. New entries provide this information only and signal the insertion of a new object into the visualization session. Updates to the geometry require, in addition to the serial number, an identifier for the field updated followed by the update. Fields can be single values, coordinates, or arrays of either. To restore a session, the log file is read and the last value recorded for a particular field is used. In the event a field is not modified, reasonable defaults, defined in the object model, are used.

Second, servlets can be used to restore prior sessions. This is as much a function of the servlet method as it is of the structure of object architecture within the visualization. The servlet must be able to scan the log file described in the previous paragraph, collect a list of all objects that were saved, and retrieve the most recent set of updates for each. Next, a VRML object is created that encapsulates the collection of restored objects. The object architecture is designed so that by appropriate instantiation, the object is configured so that it appears identically to the prior session. Currently, the application only supports restoring a prior session, without an editing capability for the restored objects.

Third, the servlets are a powerful mechanism for dynamically generating VRML content. For example, parking lots associated with office buildings are sized according to the building square footage. Per city building codes, parking lots are required to have a particular canopy cover fraction, i.e. trees. The number of trees varies with the parking lot size. In the application, one parking lot model represents all parking lot instances and therefore, the requisite number of trees must be generated dynamically. In earlier versions of the visualization, the trees were generated using ECMAscript. While functional, this approach has resulted in instabilities and resource problems. Presently, a servlet has been designed to generate an object containing the appropriate number of trees. This has resulted in faster and more reliable operation.

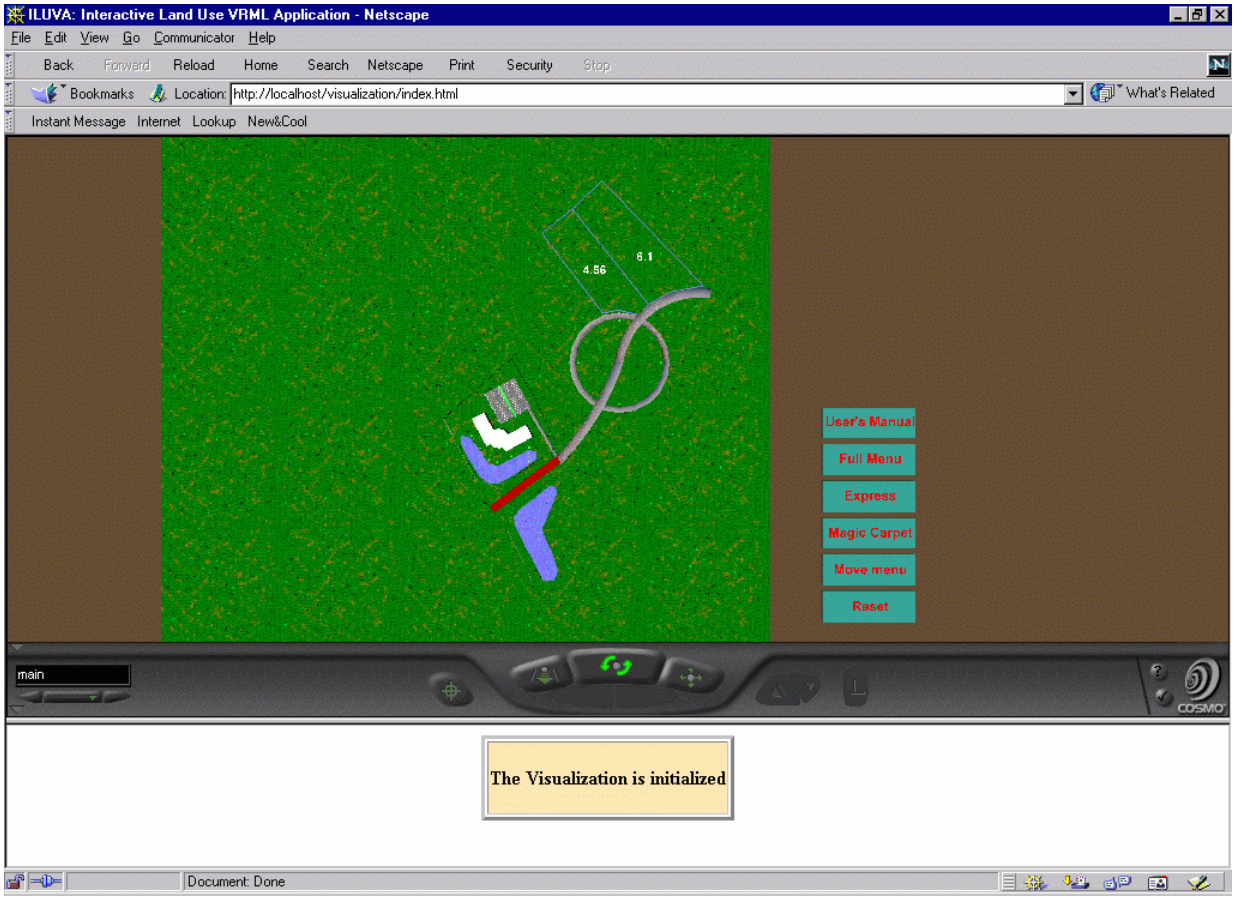
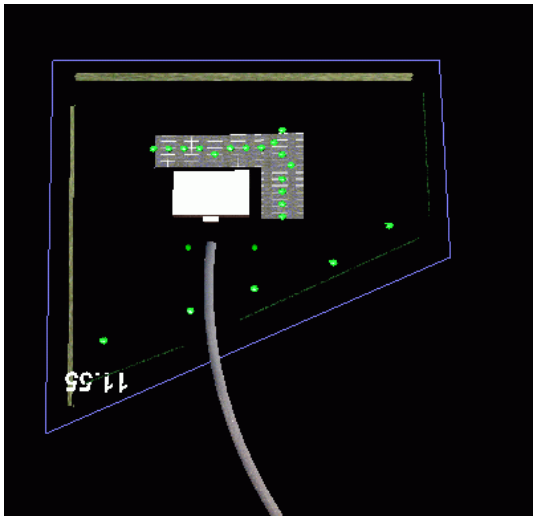
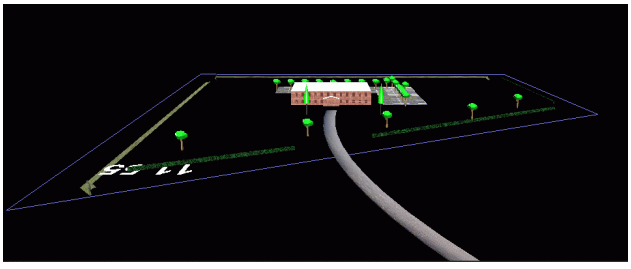


Figure 4: Opening View



(a) Top View



(b) Front View

Figure 5: Saved Session

5 EXAMPLE SESSION

Parts of an example session are presented in this section. The opening of the visualization shown in Figure 4 is a view from about 2,000 feet (about 600 meters) above the work zone. In addition, the menu appears in the right part of the view. Figure 5 shows content that was the result of a prior session. The object specifications were stored in the log file during the session and then retrieved by a servlet that generated the appropriate geometry specifications

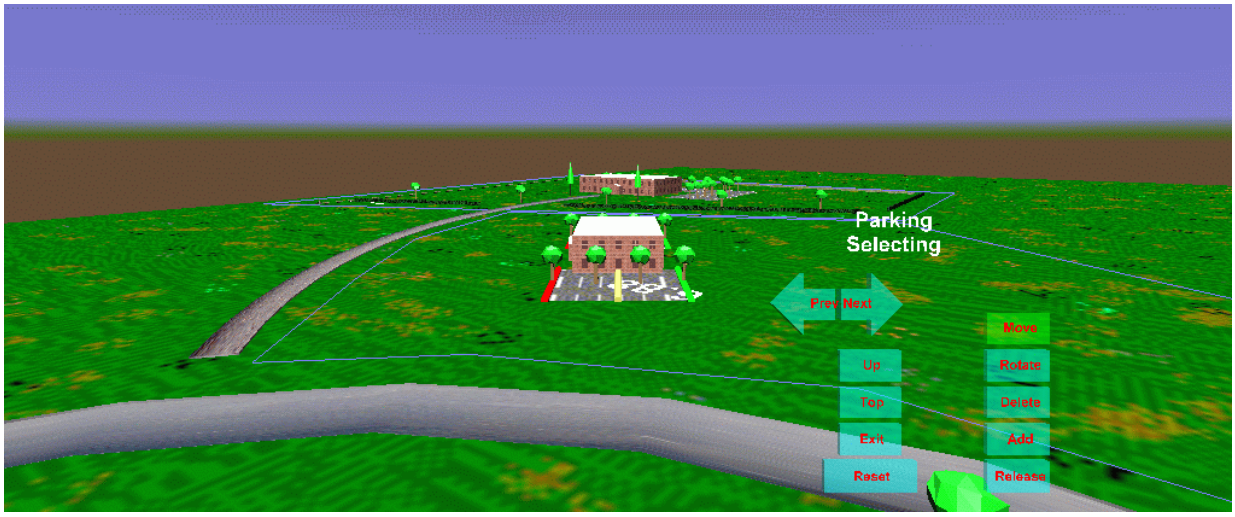


Figure 6: Continue in New Session

for each object. Figure 6 shows work continuing after restoring a session. A link to ILUVA can be found at `<http://www.lions.odu.edu/~lbelfore/ /urbanVisualization>` and presently only operates under CosmoPlayer.

6 SUMMARY AND FUTURE WORK

In this paper, we have described extensions to Belfore and Vennam (1999) that allow a visualization session to be logged and later restored. A limitation in the restore capability is that currently restored objects cannot be edited. The interface to the server required that each geometry or configuration update be communicated using the VRML `loadURL` method, where the URL is a servlet call. The session restore feature required that all objects in the application have an instantiation mode that faithfully recreates the object appearance from a prior session.

We envision several directions for future work. First, the application architecture can be expanded to allow objects to be fully editable in the restored session. Second, the servlet provides unlimited capabilities in terms of generating custom VRML modules. The modules can be synthesized from data bases or by user request. The framework presented here and in prior work allow any such geometry to be an integral part of the application.

ACKNOWLEDGMENTS

We would like to acknowledge the Virginia Modeling, Analysis and Simulation Center (VMASC) for providing facilities and other support for this project.

REFERENCES

The Apache Project Software Foundation. 2000. The Apache Server Project. Available as `<http://www.apache.org/httpd.html>`. [accessed April 1, 2000].

Arnold, K., and Gosling, J. 1996. *The Java programming language* Reading, Massachusetts: Addison-Wesley Publishing Company, Inc.

Belfore, L. A., and Vennam, R. VRML for urban visualization. *Proceedings of the 1999 Winter Simulation Conference*. 1454-1459.

Bray T., Paoli J., and Sperberg-McQueen, C. M. 1998. Extensible markup language (XML) 1.0 specification, 10 February 1998. Available as `<http://www.w3.org/TR/REC-xml>`.

Fishwick, P. A. 1999. A hybrid visual environment for models and objects. *Proceedings of the 1999 Winter Simulation Conference*. 1417-1424.

IEEE Computer Society. 1999. Special Issue. *IEEE Computer Graphics & Applications*. 18 (2).

Matsuba S. N., and Roehl, B. 1999 "Bottom, thou art translated": The making of VRML dream. *IEEE Computer Graphics & Applications* 19 (2): 45-51.

The Java Apache Project. 2000. The Apache JServ Project. Available as `<http://java.apache.org/jserv/index.html>`. [accessed April 1, 2000].

Nadeau, D. R. 1999 Tutorial: Building virtual worlds with VRML. *IEEE Computer Graphics & Applications* 19 (2): 18-29.

Reddy, M., Leclerc, Y., Iverson, L.,and Bletter, N. 1999. TerraVision II: Visualizing massive terrain databases in VRML. *IEEE Computer Graphics & Applications* 19 (2): 30-38.

Sun Microsystems, Inc. 2000. The Java^{TM} servlet API. Available as <http://java.sun.com/products/servlet>. [accessed April 1, 2000].

The Web3D Consortium. 1998. The virtual reality modeling language. Available as <http://www.web3d.org/Specifications/VRML97>. [accessed February 1, 2000].

AUTHOR BIOGRAPHIES

LEE A. BELFORE, II is an Assistant Professor of Electrical and Computer Engineering at Old Dominion University in Norfolk, Virginia. He holds a Ph.D. in Electrical Engineering from the University of Virginia. His research interests include internet based visualization, artificial neural networks, and data compression.

SURESH CHITITHOTI is a Research Assistant in the Electrical and Computer Engineering Department. He is currently working towards an M.S in Electrical Engineering from He received the B.E. in Electronics and Communications from Jawaharlal Nehru Technological University, Hyperabad, India.