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

VRML FOR URBAN VISUALIZATION

Lee A. Belfore, II

Department of Electrical and Computer Engineering
Virginia Modeling, Analysis and Simulation Center
Old Dominion University
Norfolk, VA 23529, U.S.A.

Rajesh Vennam

Digital Fusion, Inc.
11705 Potomac Crossing Way #21
Fairfax, VA 22030, U.S.A.

ABSTRACT

A virtual reality modeling language (VRML) based application has been developed as a marketing tool for a commercial park. VRML is a new web based technology for specifying and delivering three-dimensional interactive visualizations over the internet through a web browser. As a part of its definition, VRML includes primitives that specify geometries, sense different conditions in in the visualization, and allow custom definition of methods. Geometries and conditions may be linked so that the geometries can be modified or added interactively. The visualization features simple operation, an extensive menu structure, dynamic creation of objects, and an arbitration scheme.

1 INTRODUCTION

The Virtual reality modeling language (VRML) (The VRML Consortium 1997; IEEE Computer Society 1999) is a new technology giving users the ability to view three-dimensional interactive visualizations through a web browser. Web technology enables the visualization to be disseminated world wide. This paper summarizes the development of a VRML visualization of an urban landscape. The purpose of the visualization is to serve as a planning and marketing tool for the Department of Economic Development, Portsmouth, Virginia.

VRML includes modeling primitives that are used to create three-dimensional solid models. VRML includes additional primitives for programming user interactions within the visualization and update the properties of constituent objects. This is done by integrating of sensor events and developer defined methods. By creating representations of objects and defining behaviors in the scripts, complex simulations are possible. It is even possible for the visualization to be implemented in such a way that objects and behaviors are dynamically created and inserted into the visualization.

This paper is organized into five sections including an introduction, a brief VRML tutorial, a description of the operation of the visualization, a presentation of the mechanisms for managing objects in the visualization, and a summary.

2 VRML BASICS

Many books are available describing the use and application of VRML, with the language standard presented in the standard reference manual (The VRML Consortium 1997). A VRML program may have three components. The first component defines representations of the objects within the visualization, constructed from a collection of nodes that describe the geometry along with sensors that define interactions. The second component consists of script nodes that implement user defined methods. The third component is the routing of events to communicate updates of node state and parameters between scripts and objects. Interactions among objects are processed by an event driven simulation engine that is an integral part of the VRML browser plug-in. As the user moves through and interacts with the visualization, incident objects and scripts generate events that are processed by the simulation engine and then communicated back to nodes in 1the visualization as defined by declared routes.

VRML provides a rich collection of primitives, termed nodes, for creating worlds. A brief summary of several nodes are described here. Grouping nodes such as `Group` and `Transform` nodes allow the definition of a collection of objects that may be manipulated as one. Fields in the `Transform` node set the scale, position, and orientation of its constituent objects. Primitive shape nodes include the `Box`, `Sphere`, and `Cylinder` nodes. Fields in the nodes define the dimensions of each. More complex and general shape nodes such as the `IndexedFaceSet`, `Extrusion`, and `ElevationGrid` define more intricate shapes. The `IndexedFaceSet` defines the object directly by defining the surface of the object as a collection of facets. The `Extrusion` node works by the analogy that one is extruding a material through a changing cross section, with spine of the extrusion defined by a series of points in space.

The `ElevationGrid` defines a surface as the elevation of individual points on a regular grid. Sensor nodes sense interactions between the user and elements of the visualization. The sensor nodes include the `TouchSensor`, `PlaneSensor`, `CylinderSensor`, `SphereSensor`, `ProximitySensor`, and `TimeSensor`. The `TouchSensor` detects the existence of mouse events such as whether the mouse is over an object or if the mouse has been clicked while over the object. The `PlaneSensor` tracks the mouse position along a plane that is used in drag and drop planar translations of objects. The `CylinderSensor` and `SphereSensor` are used for drag and drop rotations of the object either along an axis or in space. The `ProximitySensor` detects whether the user avatar is within a particular region. The `TimeSensor` generates timing events used in animations and other programmed delays.

Custom methods appear in the `Script` node. Scripts nodes may have `field`s that hold the state of the script, `eventIn`s that receive events from other nodes, and `eventOut`s that send events to other nodes. The scripts are implemented using either ECMA scripts or Java classes. All script nodes in the visualization use ECMA scripts so we focus our attention to these. ECMA scripts are descendent from Javascript used to add dynamic capabilities to ordinary web pages. In ECMA scripts, the script processes an input event by defining a function with the same name as a declared `eventIn` field having. with an argument, the input event. The function is called by the simulation engine when the event occurs, passing the value of the event as the argument to the function. A second optional argument is the absolute time when the event occurred. An event is generated when a value is assigned to the `eventOut` variable. Furthermore, ECMA scripts have several built-in methods to control the browser, to dynamically add objects, and to dynamically add routes.

The paths defining the flow of events among nodes are defined by `ROUTE` declarations. A `ROUTE` declaration names the node and `eventOut` field that generates the event and links this to the incident node and `eventIn` field. The types must match exactly, otherwise an error is reported. Routes may also be dynamically added and deleted in script nodes using the `addRoute` and `deleteRoute` methods.

User defined nodes are defined with the `PROTO` declaration. The `PROTO` consists of an interface definition and a collection of nodes that are part of the VRML standard or that have been previously defined. `EXTERNPROTO` declarations allow the prototype to be defined in a separate file and allows straight forward reuse of a custom defined node.

3 OPERATION OF THE VISUALIZATION

The visualization is designed to be interactive and easily usable by personnel without significant technical computing expertise. To meet this goal, the visualization is designed to operate in an intuitive fashion. A menu hierarchy is built into the visualization, giving the user access to the full functionality of the tool and guiding the user through reasonable command sequences. Object manipulations are designed to be intuitive, clearly prompting the user for required input and showing the user controls when object geometries may be modified. Figure 1 shows the opening scene after invoking the visualization. Note that the image here is black and white; however, the actual visualization is in color. As noted, an integral feature of the application is the menu that serves as the primary user interface. To select a menu option, the user moves the cursor over the desired button and clicks the (left) mouse button. Menu operation includes full navigation through the menu structure as well as an express menu path that moves the user efficiently across menus, representing the anticipated usage. A typical operation is the addition of an object to the visualization. Figure 2 shows the visualization and the menu state after the addition and placement of the building. Objects are added to the visualization by clicking on the desired point in the visualization when signaled to do so. Objects that have been added to the visualization can be modified naturally through mouse clicks and drags.

4 MANAGEMENT OF OBJECTS

Managing objects requires the interaction of four components within the visualization: menus, objects, arbitration, and resource allocator. In addition, some browser dependent accommodations were necessary. As a result, a consistent interface has been defined to enable all operations to be carried out in a decentralized fashion.

4.1 Menu Hierarchy

The menu hierarchy is the primary component of the user interface. The hierarchy is constructed from a collection of menu geometries. During normal operation, one menu is made visible while the remaining menus are made invisible. The operation of menu hierarchy includes a command event that is "daisy chained" from lower menus to the top menu that communicates menu selections to the rest of the visualization. In addition, an object activated event generated by clicking on an object is distributed from top-level menus to lower level menus, ultimately resulting in the display of the appropriate menu. For this organization to work, each class of objects is assigned the same class identifier so that when the object is clicked, the appropriate menu appears by detecting the appropriate class identifier. Within the menu hierarchy, the user may move freely among menus linked in the hierarchy. Each menu directly controls the visibility of the menu beneath it in the hierarchy. The user may move up the menu hierarchy through the linkage between

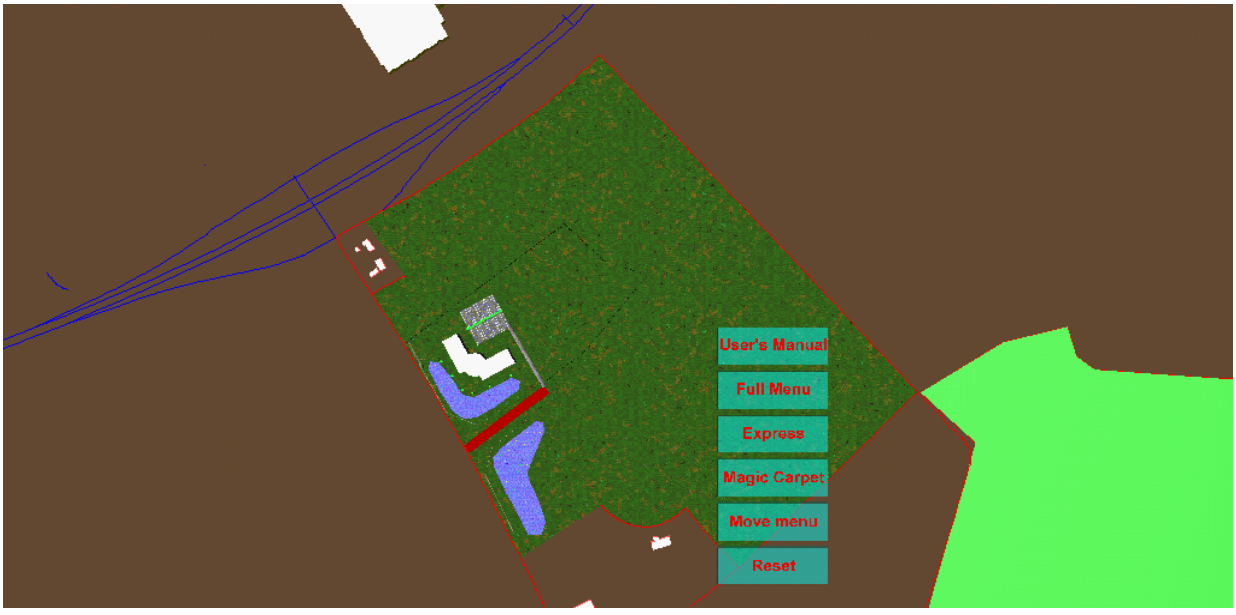

Figure 1: Opening Visualization Scene

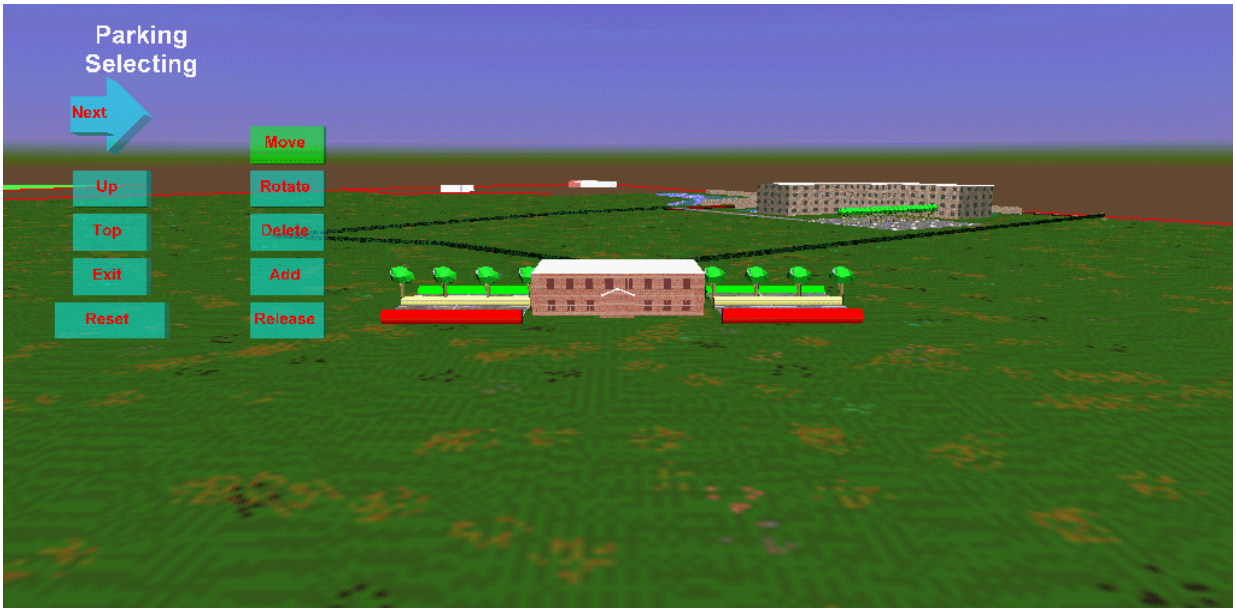

Figure 2: Visualization After Adding Building

child and parent menus in the hierarchy. Figure 3 gives the structure of the menu hierarchy. In addition to allowing user movement through the hierarchy, an express path is also included that allows movement across the hierarchy between major object classes.

4.2 Object Manipulations

Objects may be manipulated in several ways, including changing position, changing orientation, or changing geometry. Initially, the position of each object is defined by clicking at the desired location(s). After its addition, an object can be moved to any position within the working area. Other manipulations are geometry specific. Roadways and hedges are defined by points sampled along their extent. For these objects, controls appear when the object is selected into the editing context. Geometries are modified by dragging and clicking the controls. Objects that are not defined by control points and that are asymmetrical may be rotated with click

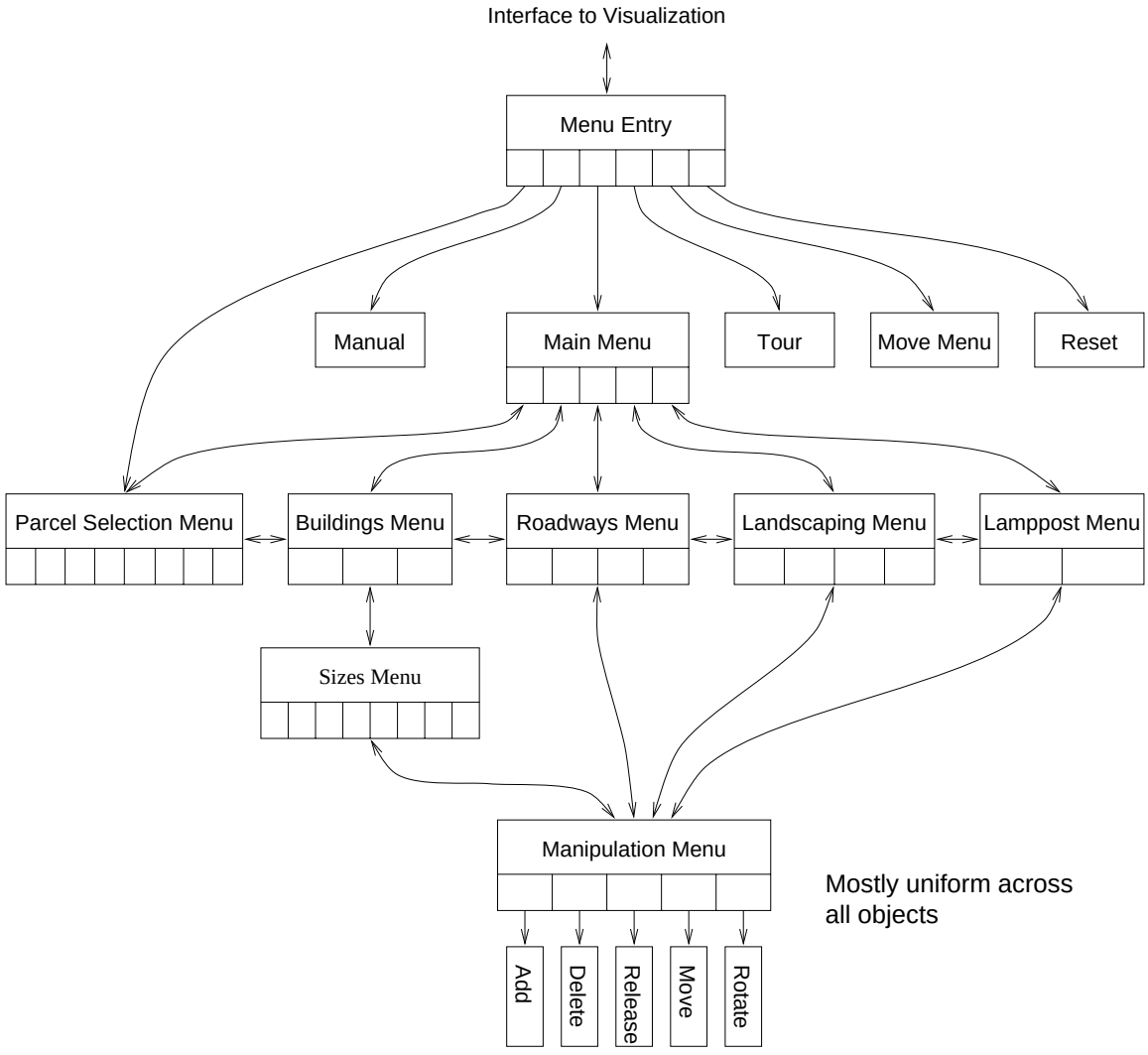

Figure 3: Menu Organization.

and drag operations. Constant area manipulations are possible for parking lots. Trees are defined by their location, and no modification of the geometry is possible.

4.3 Arbitration

Arbitration is necessary when one or more resources or contexts must be shared among a collection of other concurrent processes. In this work, the menu hierarchy and the editing context are shared among the objects that have been added to the visualization. The arbitration process requires that each object be assigned an unique serial number. A centralized arbiter has a `requestIndex eventIn` that is received from all objects and has a `grantIndex eventOut` that is transmitted to all objects. When an object is clicked, it communicates its serial number through its `requestIndex eventOut` that is received by the central arbiter. The central arbiter first communicates that a change in context shall occur, forcing the current object to surrender any resources. After a suitable delay (0.2 seconds), the serial number for the winning object is communicated through the `grantIndex eventOut` to all objects that have been added in the current session. The object whose serial number matches the `grantIndex` event communicates information about itself by transmitting an `objectActivated` event to the main menu hierarchy. The remaining objects in the visualization become quiescent, not generating any events unless later selected by a mouse click. The `command` event received by the main menu is used to call up the menu for the object, that includes taking into account any variations in the menus resulting from user selected modifications to the object. Figure 4 schematically presents the arbitration mechanism.

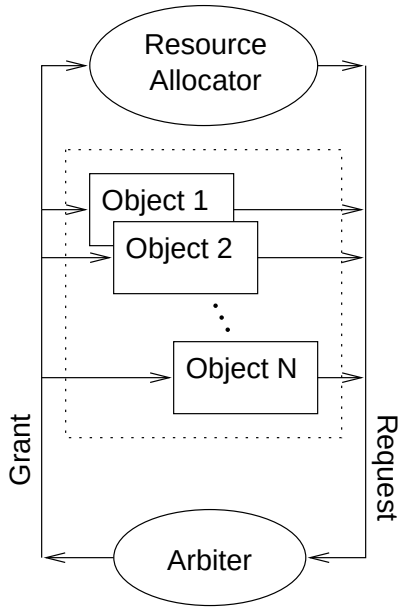

Figure 4: Object Arbitration

4.4 Dynamic Object Creation and Management

In order to more effectively utilize machine resources, objects that become a part of the visualization are created dynamically. The initial version of the visualization statically allocated a limited number of all objects that were invisible until inserted into the visualization. Because the internal data structures associated with some objects were large, the amount of memory required was larger than the physical memory present in the computer. This resulted in large startup times resulting primarily from disk swap thrashing on the machine. On lower performance machines, the visualization operated unreliably and even failed to operate on these slower machines. For these reasons, we implemented a mechanism to allocate all objects dynamically. VRML provides two methods for dynamically creating objects initiated by browser method calls from script nodes. For simple objects, the `createVrmlFromString` method was employed. For complex objects, the `createVrmlFromURL` was employed and used extensively for inserting the complex objects. The method has three arguments where the first is the URL for the object, the second and third arguments are the node that receives the new object and the name of the node's `eventIn`, respectively. The URL contains a `PROTO` definition for the object followed by a single instantiation of the newly defined node. In this work, the recipient node is a script that subsequently adds the object to the visualization and routes to the new object with the `addRoute` browser method. Once the object is added to the visualization, events are sent to the object customizing the object for the desired initial conditions. Figure 5 outlines the object creation mechanism.

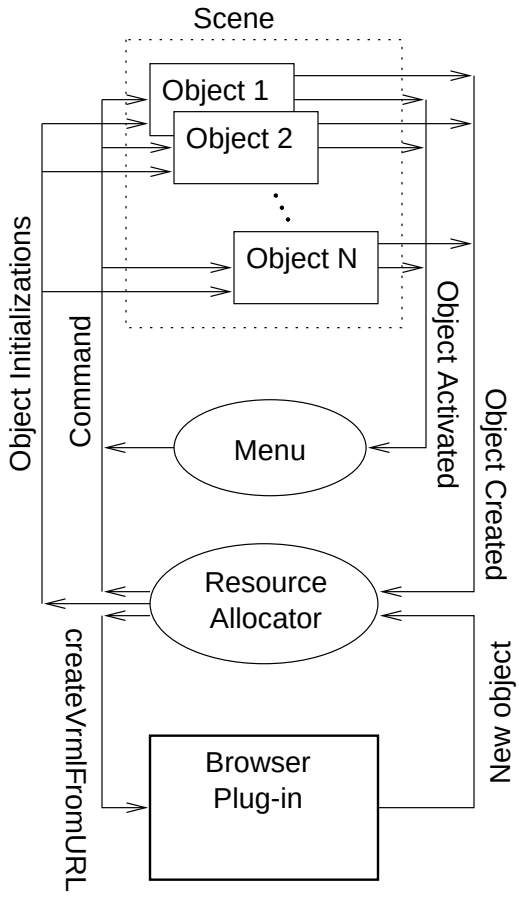

Figure 5: Dynamic Object Creation

4.5 VRML Browser Dependent Accommodations

VRML technology is fairly robust and well defined when all that is necessary is static visualization geometries or if geometry changes are simple animations. When this project was begun, few browsers had the capabilities to perform the tasks described in this work at which time CosmoPlayer provided the most extensive implementation of VRML97. Because VRML technology is still relatively immature, aspects of the VRML standard are poorly defined, the browsers may not fully implement the standard, or the browser may include features that are outside the standard. One significant accommodation comes as a result of a shortcoming in the standard. When an object is dynamically created, no standard mechanism exists to determine whether the new object has been fully loaded and initialized. When an `eventIn` is transmitted to an object that is partially loaded, the event is either be caught or not. Compensation is possible by repeatedly sending the event until the newly created object acknowledges receipt of the event. Unfortunately in this case, CosmoPlayer occasionally freezes forcing the user to reload the visualization from the beginning or to restart the browser. This likely results

from a bug in CosmoPlayer that causes the corruption of some internal data structures. The solution implemented in this work is to insert delays between the creation of the object and subsequent events. Furthermore, receipt of an acknowledgment is required from the new object before proceeding with any potentially destabilizing actions.

5 SUMMARY AND FUTURE WORK

In this paper, a visualization was presented to serve as an aid to marketing an urban commercial park. The visualization featured the ability to add new objects to the visualization and to view the visualization from different perspectives. The implementation featured an extensive menu hierarchy, an object arbitration scheme, and dynamic addition of new geometries. A shortcoming of the visualization is the inability to save a session that is a result of security features that are built into the visualization. A server based mechanism is under development to enable save and restore operation.

ACKNOWLEDGMENTS

The authors would like to thank the City of Portsmouth, Virginia for funding this work. The authors would also like to acknowledgment of the Virginia Modeling, Analysis and Simulation Center (VMASC) and in particular Drs. Thomas Mastaglio and Roland Mielke for their support of this work.

REFERENCES

The VRML Consortium, Inc., "The Virtual Reality Modeling Language," http://www.web3d.org/Specifications.

The IEEE Computer Society, *IEEE Computer Graphics & Applications*, vol. 19, no. 2, March/April 1999, Special issue on VRML.

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. He is a member of IEEE, ASEE, and Sigma Xi.

RAJESH VENNAM is presently a Developer at Digital Fusion, Inc., Chantilly Virginia. He holds an M.S in Electrical Engineering from Old Dominion University and a B.E. in Electronics and Communications from Osmania University, Hyperabad, India.