XJL – an XML Schema for the Rapid Development of Advanced Synthetic Environments

Timothy Griepp, Carolina Cruz-Neira
Virtual Reality Applications Center
tgriepp@acm.org, cruz@iastate.edu

Abstract

Virtual reality is a tremendous tool and a powerful catalyst of modern scientific and design achievements. To fully take advantage of these achievements, users are expected to have a highly advanced technical expertise. This paper discusses an approach to simplify the creation of immersive applications. We present the design and implementation of an interpreted language for the rapid development of virtual reality applications. The design is based on a specialized XML schema. The intended end users are digital artists and designers. Our goal is to remove the complexity of writing and compiling traditional code and provide the artist a more usable method of developing a full-featured application.

Our design has been built upon VR Juggler; an open source development environment focused on abstracting applications from the hardware and devices used in their run time execution. Upon this foundation we have created an interface through which the user may define object animation, navigation algorithms, object transformations, and environmental settings.

1 Introduction

The advancements of visualization are becoming more accessible with each passing day as the rapid growth of computational and display technologies push their way into the industry. Not twenty years ago, computers filled entire rooms and could still perform only rudimentary tasks. Today the same tasks could be performed by the microprocessor in a wristwatch [1]. This is further illustrated, in that the graphic and computational capabilities of supercomputers from the mid-90's are now matched by desktop computers; but at a significantly reduced cost.

A leading trend of computer technology is the advancement of graphics hardware and its availability at the desktop level. Computer based 3D graphics are becoming very affordable and are thus accessible to a community that extends beyond the confines of computer science laboratories. There are, nevertheless, not many people that can develop applications that capitalize on these graphical advancements. Such applications require both technical and artistic knowledge and there are few people who are skilled computer scientists and skilled artists at the same time. As graphics technology advances and gets more complex, there is a pressing need to develop methods and tools to hide this complexity to provide access to a wider set of users. In this way, artists can use the technology and exploit it to its full potential.

The issues of increased complexity and the need of hiding that complexity become even more critical in the area of virtual reality. By reducing the programming requirements of advanced virtual world development, a significant number of digital artists, previously held at bay from such objectives, would now have the means to develop powerful and creative virtual environments. Using a plain text or graphical interface, the artist would be able to create highly detailed applications without a background in computer programming. With such an interface a set of rules or instructions could be created that defines an applications behavior and the means through which the user is to interact with the application. The instruction set would also define geometry animation, level-of-detail, and switch nodes; passing instructions to the actual program as to how these objects are to be manipulated. As a separate entity, these instructions would isolate the developer from the complex programming necessary to create the virtual environment. As a result, advanced synthetic environment applications could be developed without requiring the artist to have an advanced understanding of visualization programming.

1.1 Background

The number of skilled 3D computer artists has increased dramatically in recent years. The combination of

increased availability of computers and simpler to use modeling tools has allowed for younger artists to explore technology much earlier in their careers. These younger artists integrate technology into their creative process, developing modeling and rendering skills artistically comparable to, for example, painting or sculpting skills. However, these new generations of talented artists are still limited on how they can utilize virtual reality as a medium for their art.

Consider the following scenario: A highly talented digital 3D artist whom has the vision to conceptualize interactive virtual worlds in all their detail. The logical understanding of what such an environment requires is near at hand and the creative skill required to produce the environments content is readily available. He has produced a full-featured backdrop to the application in its entire geometric and textured splendor. He has charted out the relationship between the end user and the dynamic attributes of the scene graph. And he has defined the methodology of how that user will navigate their creation. What this artist lacks, however, are programming skills: The ability to transform their digital resources into a fully functional interactive application. The artist is not prepared to program these resources into a real-time application. The geometry statically resides in its native data file and the relationship between geometric nodes, switches, level-of-detail, lighting, navigation, and interaction remains etched in paper. The challenge is that this artist has no programming experience. As we consider the task of empowering this user, we recognize the tools are not readily available to assist in transforming their ideas into an immersive application.

It seems apparent; whether the focus is on a software engineer attempting to develop an aesthetically enriched simulation, or an artist striving to produce a real-time interactive exhibit, that a strong balance of skills in the realm of the analytical and holistic design is critical. Considering the principles of game development, Moody [2] states: "*Games also must appeal to consumers not simply on technical merit but on aesthetic merit – a point that is lost on the vast majority of programmers, who tend to focus on the underlying algorithms… rather then on such refinements as narrative, level of social commentary, and quality of dialog.*" We aim to provide a solution wherein the artist may reach farther into the technical side of their compositions. Furthermore, we aim to reduce the entry barriers that stand before novice users of large-scale visualization technologies.

As we explore the components of a typical immersive application, or more precisely the common features included in applications inspired by the artistic community, we note requirements in the following areas: Geometry Placement, Geometry Dynamics, Geometry Visibility, Scene Navigation, Scene Interaction, Lighting, Audio, and User Feedback. Exploring these requirements in greater detail we have isolated a series of application features and user driven events. These are events and processes that are routinely integrated into immersive applications and during development undergo frequent modification as the designer fine-tunes their approach.

1.2 Objectives

Our goal for this research is to investigate a venue through which the traditional digital artist can create complex virtual reality applications. Although in its ideal state this solution would relieve the artist of all coded development, using, for example visual programming techniques, our present focus is limited to a solution that eliminates the need for using complex programming languages like C or C++. Our solution focuses on the use of a more intuitive specification approach through XML. By removing the complexity of writing and compiling traditional code, we hope to provide the artist a more tangible method of developing a full-featured application. This work discusses the effectiveness of an interpreted language interface used for the rapid development of synthetic environment applications; in particular we look into a specialized XML schema. This schema is the formal structure of a database system used to express shared vocabularies and allow immersive applications to carry out rules defined by the application designer. We have named this schema XJL, an acronym for eXtensible Juggler Language. The schema is broken into the eight application features identified in the previous section. Through the values and attributes contained therein, a user is able to control a broad range of an application's functionality. Within these elements the user may configure feedback and environmental settings such as heads up displays, statistical feedback, clipping planes, and lighting. Furthermore, they are given the ability to define tracked objects. These objects may be used throughout an application in the same way as physical motion trackers. The ideal implementation will support any number of controlled objects. Furthermore, its navigators and interaction routines shall only be limited by the extents of the control devices used to access such methods. Although our primary emphasis will be the schema driving this solution, we aim to implement most of the schema's features as a means of exercising its limits.

Our test application, which we have named XJ Nav, is a C++ software tool built upon OpenGL Performer [3] and VR Juggler [4]. Upon this foundation we have created an interface through which the user may define object behavior, navigation algorithms, object transformations, and environmental settings. XJ Nav has been designed

such that the user only need modify the XJL configuration file to make advanced additions and modifications to the behavior of their immersive application. In this way, its features parallel the eight core elements of the XJL schema and provide an answer to the typical requirements of prototypical applications, inspired by the artistic community. In review, these requirements lay within the following categories: Geometry placement, geometry dynamics, geometry visibility, scene navigation, scene interaction, lighting, audio, and user feedback. Though the use of an XJL configuration file, a novice, and non-technical user, may quickly and easily develop an advanced application. The component driven nature of the XJL schema further simplifies the process of development by freeing the user to archive, share, and rapidly replace specific components of their configuration as they refine their application.

2 Related Work

2.1 VR Toolkits

Software tools solutions such as Avocado [5], CAVELib [6], Lightning [7], MR Toolkit [8], WorldToolKit [9], Vega [10], and VR Juggler [4] have made significant advances in bringing visualization within the grasp of the typical software engineer. These solutions, however, fail to provide a working environment friendly to the non-programmer. These tools require advanced knowledge of topics such as scene graph management, matrix transformations, coordinate systems, and collision detection.

Using a hybrid combination of compiled and interpreted programming paradigms, some of these solutions have taken steps to simplify the tasks of real-time graphics development. Although these systems bring with them great advantages for rapid development of advanced applications, they remain outside the grasp of non-technical users. The interpreted language interfaces do not fully address the skill deficiencies many artists face when exploring 3D graphics programming. The learning curve of these scripting languages, although more manageable then compiled languages, are often non trivial to the novice user.

2.2 Interpreted Language Interfaces

Other solutions, such as Alice [11], Obliq-3D [12], TBAG [13], and WorldUp [14] have made advances using scripting languages and visual interfaces to develop interactive 3D environments.

Although these tools have taken great strides in bridging the gap between the novice and advanced users, there remain many steps to be taken. In simplifying the process of application development, we aim to explore an interface that operates on top of other powerful tools. A layer that simplifies the programming process but also allows full access to the capabilities of advanced visualization technologies.

2.3 XML Specifications

On a converging front, solutions such as XGL [15] and X3D [16] are providing new opportunities in the realm of open standard geometry. XGL is an extensible markup language specification designed to represent 3D information for the purpose of visualization. As such “it attempts to capture all of the 3D information that can be rendered by SGI’s OpenGL rendering library.” Revisiting the traditional VRML format, X3D is the ‘New Generation’ specification for Web3D. Capitalizing on these advances, CONTIGRA [17] has been developed as an XML-based approach to constructing interactive 3D graphics applications for the web. Although powerful in its arena, the scope of CONTIGRA is presently limited to web based applications. It does not maintain nor indicate future efforts that would bring its strengths to large-scale immersive visualization systems.

A solution known as 3dml[18] has perhaps made the most significant advances, of late, to the end of allowing non-programmers to create large-scale simulation applications. 3dml is a markup language that describes applications such as desktop-based 3D presentations, virtual reality applications, or augmented reality applications; in other words, applications with different types of input devices, output devices, and 3D interaction techniques. 3dml places a strong emphasis on what its authors describe as ITs or interaction techniques. By abstracting the underling complexities of these interaction techniques and providing the application designer a modular, component driven, markup language, 3dml addresses issues of readability and rapid development as they relate to the production of immersive applications. In its pursuit of these objectives, 3dml defines details such as VR objects, interaction techniques, and interaction devices. The solution does not, however, address details such as device configuration or level-of-detail. It also refrains from describing visual, haptic, or auditory capabilities of VR objects. 3dml provides an extensive coverage of interaction techniques. The specification, however, tends to yield data files that can often be difficult for the non-technical user to read, interpret, and consequently develop with. The work described herein focuses on a solution that steps beyond the previously referenced paradigms. Our work strives to address issues of scene configuration, navigation, interaction, and behavior methods, which include processes such as

lighting, level-of-detail, and animation. To this end, we aim to root our solution in a XML schema that is both simplistic and extendable. Our objective is to capitalize upon the strengths of tools such as VR Juggler while freeing the user from its technical knowledge requirements.

3 The XJL Schema

Given a user whom understands the high-level concepts of 3D modeling and animation – based on experience with advanced 3D content production tools – we have developed an XML schema that equips the user to produce advanced virtual environment applications without facing the rigors of technical programming and scripting languages. Our solution, the eXtensible Juggler Language (XJL) is designed to provide a simple, clean, and extendable interface for the development of such environments. The foundational elements of XJL are illustrated in Figure 1.

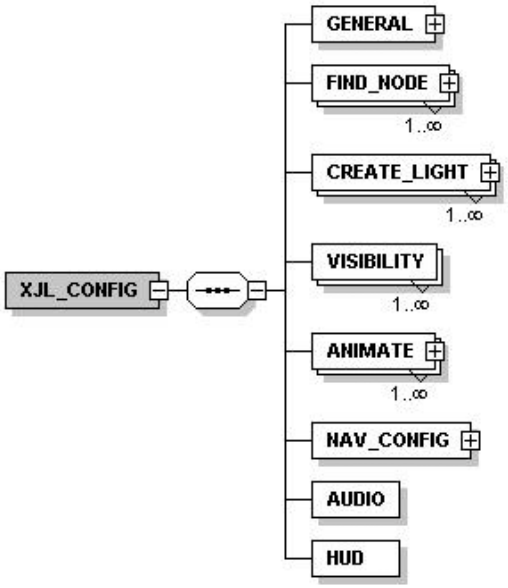

Figure 1 XJL Configuration Elements

XJL allows users to explore highly advanced systems of interaction and automation through cross-linking of navigation, interaction, and behavior methods.

Named objects exist at the foundation of such systems. These objects may either act on or be acted upon by other named objects. Much like physical motion trackers, also used in this paradigm, all named objects are considered tracked. In this way, an object referenced and named within the FIND_NODE element may be used as part of a level-of-detail, animation, or navigation algorithm.

As illustrated in Figure 2, FIND_NODE contains a series of sub-elements that specify the placement and transformational constraints of a geometry system. *Node*, *center_node*, and *name* are among the attributes used in this element. These attributes identify the object name used in the source dataset, a optionally used object to define the center of transformation, and a friendly name that is used as a reference by other interrelated methods.

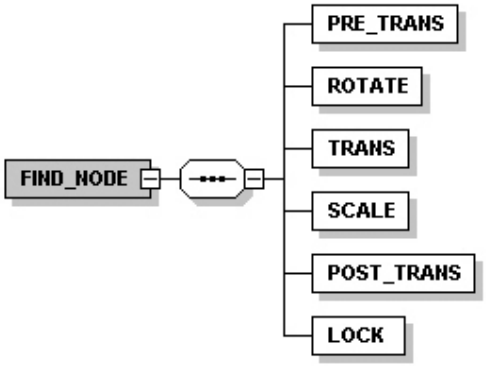

Figure 2 FIND_NODE Elements

As a child of NAV_CONFIG, the DEFINE_NAV (figure 3) element is used in the development of an applications navigation and interaction algorithms. A given application may hold any number of navigators, each of which may maintain any number of accelerators. The accelerators, based upon the ACCEL element, are used in part to define how an environment is transformed about its user. A users movement and viewing perspective as they travel through a virtual space may be further be controlled by the TURN and LOOK elements. In this way an application designer may separate the viewing direction for the direction of traversal. Furthermore, they may determine the methods of directional control. That is, the designer can specify whether the direction of motion is controlled by a tracked object (a named object or physical motion tracker) or if the direction of motion is determined by some predefined vector within the xyz coordinate system.

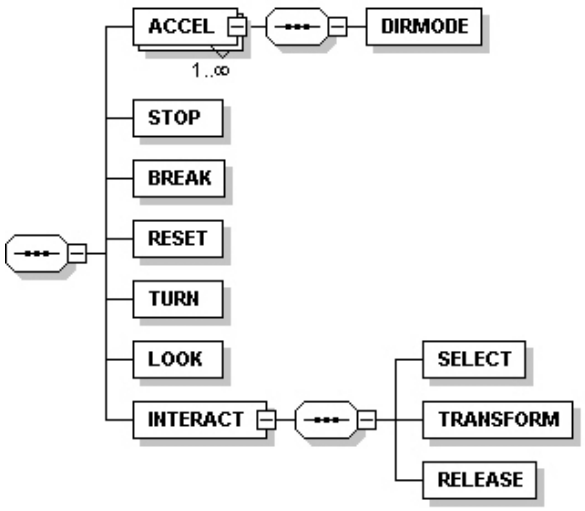

Figure 3 DEFINE_NAV Elements

4 Design

VR Juggler [4], a project developed at Iowa State University's Virtual Reality Application Center, provides an object-oriented, component based approach to application development. The system exhibits itself as a well-rounded solution for cross-platform development and demonstrates its strength through its run-time configurable hardware abstraction layer, support for distributed environments, and a graphical manager for extension of application interfaces. Capitalizing upon these functionalities, we have developed an object-oriented application, which allows the end-user to take full advantage of XJL's extendibility. The Xerces C++ Parser [19], a project of the Apache Software Foundation, is used to support the XML data processing.

Being built upon VR Juggler, XJ Nav, is initiated by adding an application object to the VR Juggler kernel. As part of the VR Juggler design, the kernel, or VR Juggler system, is started independently from the actual application. The main loop consequently takes the following form:

```
// Start the kernel
   1  vjKernel* kernel = vjKernel::instance();
   2  kernel->start();
// Instantiate XJL parse engine
   3  XParse* parseXP = new XParse();
// Instantiate application
   4  xjNavApp * my_xjApp = new xjNavApp ();
// Parse XJL configuration for GENERAL element
   5  parseXP->generalParse(XParse(*my_xjApp,
      XJ_filename));
// Configure application VR Juggler config chunks
   6  for ( int i = 1; i <= XParse::configFileCount; ++i )
   7  kernel->loadConfigFile(XParse::XJconfig[i]);
// Set application
   7  kernel->setApplication(my_xjApp);
```

In lines one and two we initiate and start the VR Juggler system kernel. The XJL parse engine and XJ Nav application are instantiated in lines three and four. At line five we begin actively processing the XJL configuration file. For issues of readability and data management we have partitioned the parsing of configuration data into a number of subroutines specific to the various XJL elements. In this particular segment of processing we extract data relevant to the initialization of the application. This data is homed in the GENERAL element. Amongst this data is a list of VR Juggler "config chunks" [20]. Juggler specific configuration data, obtained from XJL, is loaded to the kernel through lines six and seven. In line eight we activate our application by sending it to the kernel.

As mentioned in the previous section, there are several parsing functions accessed by the application. Unlike generalParse, which acts on the data it obtains prior to the end of its execution, some of the data processing functions store the information they obtain for use later in the programs execution. One such method, named dcsParse, process the data contained within the FIND_NODE element. In this function data is stored in a series of hash_map's. A hash_map is a data container that associates "key" objects with related "data" objects. By using hash_maps, geometric transformations may be stored according to the name given to a node by an application designer. The name becomes the map key. In this way, other XJL defined processes may reference the previously named node and the XJ Nav software can retrieve all required data objects by way of that name.

Fulfilling another unique need is dcsAnimParse. This is used to extract animation commands from the configuration file and process those commands during each pass of VR Jugglers pre-frame routine. The data retrieved by dcsAnimParse is stored in a struct that is passed back to the XJ Nav application as a hash_map. During each instance of the applications preframe routine a function named dcsAnimate is called, wherein the position of tracked objects are updated according to the perimeters obtained by dcsAnimParse. The operations of dcsInteract are accessed in a similar way. In the parsing of the INTERACT element, however, we take a different approach. INTERACT exists as a sub-element to the DEFINE_NAV element. As such, we process its data as part of the navigation system. The data stored in the INTERACT element is limited to the interaction controls used to initiate processes of interaction. Upon being parsed this data is converted to a vector. (Vectors are

variable-sized containers whose elements are arranged in a strict linear order. Vector data types allow for the random access of elements and for the insertion and removal of elements that exist in the data set.) In this way the application may process the system of interaction buttons used to identify unique operations. Each element of the vector data set represents one of the five buttons accessible by the interaction device. During each pass of the application we check to see if the buttons of interest are active. If all buttons are active we enable the operations associated with that button combination. As mentioned earlier, the data of the INTERACT element is processed with the navigation data. Considering the requirement of allowing any number of navigation configurations to be accessible by the application designer and subsequently the end user we have used vectors again for the storage of navigation objects. In this way an undetermined number of navigation routines may be parsed and used by the application. The components of each navigation system are stored to a temporary class object. Upon completing the configuration of this temporary object, it is pushed onto a vector of similar objects accessible by the navigation engine.

5 Results

Using XJL as the conduit, a wide variety of applications may be developed. With digital artists as our target audience we have devised a specification that is simple and clean in its layout, yet includes many advanced features, which come to light through the interlinking of elements. In the following text we discuss a series of XJL configurations that demonstrate avenues of simple and advanced application development.

5.1 Elementary Application Development

In the configuration below we have defined an application based on a simple “drive” navigator. That is a locomotion system with collision and gravity that has forward and reverse accelerators whose directional vectors are determined by the wand. Using the GENERAL element we have disabled heads-up-display access, statistical feedback, and audio. The geometry set is an OpenFlight™ [21] database, which we have loaded to its default position and orientation. Our scene has one global light and there are no tracked objects, animated sequences, or visibility behaviors.

```
<XJL_CONFIG>
  <GENERAL
    hud="false" toggle_nav="true"
    stats="false" audio="false"
    near="0.4" far="200000" units="FEET”>
    <HOME x="0.0" y="0.0" z="0.0"
      yaw="0.0" roll="0.0" pitch="0.0"/>
    <VJ_CONFIG
      vjconfig="/homedir/.vjconfig/sim.config"/>
    <SET_PATH path="/homedir/XJL/data"/>
    <LOAD_DATABASE
      geometry="/homedir/XJL/data/test.flt"
      name="test_db"
      path="/homedir/XJL/data/textures"
      x="0.0" y="0.0" z="0.0"
      yaw="0.0" roll="0.0" pitch="0.0" scale="1"/>
  </GENERAL>
  <CREATE_LIGHT name="sun" type="GLOBAL">
    <POSITION x="1.0" y="1.0" z="0.0"/>
    <COLOR property="DIFFUSE"
      r="0.7" g="0.7" b="0.7"/>
    <COLOR property="AMBIENT"
      r="0.3" g="0.3" b="0.3"/>
    <COLOR property="SPECULAR"
      r="1.0" g="1.0" b="1.0"/>
    <INTENSITY power="200" angle="55"/>
  </CREATE_LIGHT>
  <NAV_CONFIG>
    <DEFINE_NAV
      name="drive" collision="true" gravity="true">
      <ACCEL
        control="10000" mode="ramp"
        accel="10.0" max="50.0">
      <DIRMODE mode=" tracker "/></ACCEL>
      <ACCEL
        control="00100" mode="ramp"
        accel="-10.0" max="-50.0">
      <DIRMODE mode=" tracker "/></ACCEL>
      <STOP control="11000"/>
      <BRAKE control="010000" decel="0.15"/>
      <RESET control="11100"/>
      <TURN
        mode="absolute" tracker="wand"
        buffer="3.5" accel="1.25"/>
    </DEFINE_NAV>
  </NAV_CONFIG>
</XJL_CONFIG>
```

5.2 Advanced Application Development

Building upon the features explored in the previous example, we will now explore several of the advanced functionalities XJL presents. In this next example we look at a coordinates based navigator that holds similarities to the ‘retro-rocket’ system of a space satellite.

Setting the acceleration mode to *ramp* – a paradigm that increases the momentum of a navigator, at a rate of *accel* units per second, until the value of *max* is reached – we are able to define a system wherein momentum is maintained until acted upon by another object. In this case until a user collides with another object, applies a thruster in an opposing direction, or activates the break or stop functions. Setting the directional mode to "coord", the system accelerators act in the direction indicated by the "*x, y, z*" coordinates and "*yaw, roll, pitch*" rotational attributes. If the attribute *y* is set to "1" and *x, z,* and the rotational attributes are set to "0", upon pressing the interface button(s) indicated by the control element the user would move upwards in the scene, or more precisely, the scene would descend with respect to the users physical position. Coordinates based motion is linked to the users local coordinate system. In this way, the rotational elements may be used to reorient the users directional perspective, with respect to the scene. Our example illustrates the use of six thrusters; one dedicated to each of the following directions: x, y, z, -x, -y, -z. A user may modify this design by adding three rotational accelerators and leaving only one directional accelerator. In this way system will respond more like a rocket with rotational thrusters.

```
<DEFINE_NAV
  name="thruster" collision="true" gravity="false">
  <ACCEL
    mode="ramp" control="10000"
    accel="20.0" max="60.0">
    <DIRMODE mode="coord"
      x="1.0" y="0.0" z="0.0"
      yaw="0.0" roll="0.0" pitch="0.0"/></ACCEL>
  <ACCEL
    mode="ramp" control="01000"
    accel="20.0" max="60.0">
    <DIRMODE mode="coord"
      x="0.0" y="1.0" z="0.0"
      yaw="0.0" roll="0.0" pitch="0.0"/></ACCEL>
  <ACCEL
    mode="ramp" control="00100"
    accel="20.0" max="60.0">
    <DIRMODE mode="coord"
      x="0.0" y="0.0" z="1.0"
      yaw="0.0" roll="0.0" pitch="0.0"/></ACCEL>
  <ACCEL
    mode="ramp" control="10001"
    accel="20.0" max="60.0">
    <DIRMODE mode="coord"
      x="-1.0" y="0.0" z="1.0"
      yaw="0.0" roll="0.0" pitch="0.0"/></ACCEL>
  <ACCEL
    mode="ramp" control="01001"
    accel="20.0" max="60.0">
    <DIRMODE mode="coord"
      x="0.0" y="-1.0" z="0.0"
      yaw="0.0" roll="0.0" pitch="0.0"/></ACCEL>
  <ACCEL
    mode="ramp" control="00101"
    accel="20.0" max="60.0">
    <DIRMODE mode="coord"
      x="0.0" y="0.0" z="-1.0"
      yaw="0.0" roll="0.0" pitch="0.0"/></ACCEL>
  <STOP control="11000"/>
  <BRAKE control="10100" decel="0.15"/>
  <RESET control="00011"/>
</DEFINE_NAV>
```

Other advanced navigational methods that may be explored involve the tracked objects. The TURN and LOOK elements below illustrate two methods of controlling ones motion about and perspective of a scene. In the TURN element we have set the *control_state* to "toggle". In this way, the turn methods are activated when a user selects the button indicated by the *control* attribute. Selecting this button again would deactivate the method. If a condition is desired, wherein the turn method is always active, the control attributes may be excluded from the configuration. As a user rotates the wand 3 degrees off axis, with respect to the coordinates of the physical display system the direction of rotation will be adjusted at a rate of 2 degrees for each additional degree of wand rotation. In light of the AXISLOCK configuration, however, the rotation will be constrained to the horizontal plain.

```
<TURN
  control="01100"
  control_state="toggle"
  mode="relative"
  tracker="wand"
  buffer="3.0" accel="2.0">
  <AXISLOCK
    freeze_yaw = "false"
    freeze_roll = "true"
    freeze_pitch = "true" />
</TURN>
```

Using the LOOK element we explore another method of rotation. In this example we explore the use of tracked objects as a mechanism for rotation. With *mode* set to "absolute" the orientation of the viewing lookup vector will be directly related to the orientation of the selected tracker. We have selected a tracked object for this purpose. As the object "looking_object" is rotated with respect to its local coordinate system the users viewing perspective will also be adjusted. That is to say, the

orientation of the global coordinate system will be modified by the inverse of the change in the objects local coordinate system.

```
<LOOK
  mode="absolute "
  tracker=”object"
  object="looking_object"/>
```

Additional advances may be realized by linking the motion of the tacked object to animation behaviors. We now explore two such behavioral systems. The first example links the position of the tracked object to position of the user. This configuration directs the object to shadow the position of the wand offset by a distance of 1 unit along the z-axis.

```
<ANIMATE
  name="looking_object "
  <CONTROL
    follow="true"
    tracker="wand"
    buffer_x="0" buffer_y="0" buffer_z="1" >
</ANIMATE>
```

Our second example the object has been configured to rotate 90 degrees at a rate of 0.5 degrees per frame each time the first, third, and fifth buttons of the control device are selected.

```
<ANIMATE
  name="looking_object "
  method="static"
  repeat=”1”
  <LIMIT x="0.0" y="0.0" z="0.0"
    yaw="90.0" pitch="0.0" roll="0.0"/>
  <STEP x="0.0" y="0.0" z="0.0"
    yaw="0.5" pitch="0.0" roll="0.0"/>
  <CONTROL control="10101"
    control_state="toggle"  >
</ANIMATE
```

Through the combined use of the three preceding examples a configuration can be established wherein a reference object shadows the position of the wand and is used to gradually rotate the users viewpoint 90 degrees about the y-axis each time a specific button combination is selected. Related methods may be used to implement a full featured, worlds-in-miniature [Stoakley95] control set for the navigation of ones environment.

The VISIBILITY element allows the application designer to configure level-of-detail and switch-node operations. The following two XJL segments show a sample level-of-detail configuration. In the first segment an object is set to be hidden from the rendering pipeline when the user comes within the distance specified by the given x, y, z coordinates. The counterpart to this object is referenced in the next segment. Here, the object is set to be visible when the user navigates within the specified by the x, y, z coordinates. When the selected tracker exits the coordinate range the visibility method inverses the action taken by the *view* attribute of a given configuration.

```
<VISIBILITY
  name="small_object"
  operator="greater”
  tracker="head"
  view="hide"
  x="3000" y="1000" z="3000"/>

<VISIBILITY
  name="big_object"
  operator="less"
  tracker="head"
  view="show"
  x="3010" y="1010" z="3010"/>
```

Closely related to the level-of-detail operations, our next example explores the use of VISIBILITY for the setup of a switch-node operation. In this configuration an object named “locked_door” is hidden when an object known as “key” comes within 5 units of the door. Related features may be designed through the use of an ANIMATE element. Using ANIMATE an object, the door in the case, may be configured to slowly open when a secondary object enters the desired range.

```
<VISIBILITY
  name="locked_door"
  operator="less”
  tracker="object"
  object=”key”
  view="hide"
  x="5" y="5" z="5"/>
```

Another switch-node configuration may be designed by using “selected” as the value that fills the *operator* attribute. In the example below, we use the object “button1” to control the visibility of the object “geom_1”. When button1 is selected geom_1 is made visible.

```
<VISIBILITY
  name="object_1"
  operator="selected”
  tracker="object"
  object=”button1”
  view="show">
```

6 Conclusion

In designing the XJL specification, and its associated test application, we have developed an approach to empower non-technical digital artists for the task of designing immersive visualization applications.

With a focus on component-driven extendibility, XJL proves itself to be a simple yet powerful tool for the implementation of advanced navigation, interaction, and behavioral models. Fundamental to the success of this approach is the ability to use tracked objects as tools for the manipulation of a scene. In this way, designers may exercise their application using predefined mathematical routines, user driven events, or multi-layered combinations encompassing both methodologies. Although the successful application designer must have a working knowledge of 3D space and geometric modeling tools, we are confident we have developed a system through which users may develop immersive applications unencumbered by the technical requirements of related solutions with similar functionality.

A limitation of our present solution includes a lack of support for a wide variety of interaction devices. As our work continues we hope to generalize our solution such that the application designers need not limit themselves to a specific interaction device. At this time, however, more work is required exploring abstraction layers that isolate developers from the wide variety of interface protocols. Another area where we see room for improvement includes the development of a visual design interface. Using the XJL document type definition, a graphical interface may be developed that supports the visual – drag and drop / point and click – design of advanced applications.

7 References

[1] Keep, C., "Knocking of Heaven's Door: Leibniz, Baudrillard and Virtual Reality", EJournal, Volume 3 Number 2., September 1993.

[2] Moody, F., "The Visionary Position: the inside story of the digital dreamers who are making virtual reality a reality", Random House, Inc.: New York, 1999.

[3] Rohlf and Helman, "IRIS Performer: A High-Performance Multiprocessing Toolkit for Real-Time 3D Graphics", Proceedings SIGGRAPH 94, ACM Press, New York, 1994, pp 381-394.

[4] Bierbaum, A., "*VR Juggler: A Virtual Platform for Virtual Reality Application Development*", MS Thesis, Iowa State University, 2000.

[5] Dai, Eckel, Göbel, Hasenbrink, and others, "Virtual Spaces: VR Projection System Technologies and Applications", Tutorial Notes, Eurographics 97, Budapest 1997.

[6] Cruz-Neira, C. "Virtual Reality Based on Multiple Projection Screens: The CAVE and Its Applications to Computational Science and Engineering", PhD. Dissertation, University of Illinois at Chicago, 1995.

[7] Landauer, Blach, Bues, Rösch, and Simon, "Toward Next Generation Virtual Reality Systems", Proceedings IEEE International Conference on Multimedia Computing and Systems, Ottawa, 1997.

[8] Shaw and Green, "The MR Toolkit Peers Package and Experiment", IEEE Virtual Reality Annual International Symposium (VRAIS 1993), pp 463-469.

[9] "WorldToolKit Release9: Technical Overview", http://www.sense8.com, Visited 10-01-01.

[10] "Vega: The Comprehensive Software Environment for Realtime Application Development", http://www.multigen.com /support/dc_files/Vega_brochure.pdf, Visited 10-01-01.

[11] Conway, Audia, Burnette, Cosgrove, and others, "Alice: Lessons Learned from Building a 3D System for Novices", ACM CHI 2000 Papers, pp 486-493.

[12] Najork, M., "Obliq-3D Tutorial and Reference Manual", SRC Research Report 129, December 1994.

[13] Elliott, Schechter, Yeung, and Abi-Ezzi, "TBAG, A High Level Framework for Interactive, Animated 3D Graphics Applications", SIGGRAPH 94 Conference Proceedings, 1994.

[14] "World Up User's Manual", http://www.sense8.com, Visited 10-01-01.

[15] "XGL File Format Specification", http://www.xglspec.org, Visited 10-01-01.

[16] "X3D – Extensible 3D", http://www.web3d.org,

[17] Dachselt, R., "CONTIGRA: A High-Level XML-Based Approach to Interactive 3D Components", SIGGRAPH 2001 Conference Abstracts and Applications, p 163

[18] Figueroa, Green, and Hoover, "3dml: A Language for 3D Interaction Techniques Specification", EuroGraphics 2001 Short Presentations, 2001.

[19] "Xerces C++ Parser", http://xml.apache.org/xerces-c , Visited 11-06-01.

[20] Just, Bierbaum, Hartling, Meinert, Cruz-Neira, and Baker, "VjControl: An Advanced Configuration Management Tool for VR Juggler Applications", Published at IEEE VR 2001, Yokohmoa, March 2001.

[21] Dowgiallo, T., "Introduction to OpenFlight APIs", http://www.multigen.com/products/openflight/intro_api.shtml, Visited 10-06-01, 1997.