XJL – an XML schema for the rapid development of advanced synthetic environments

by

Timothy Paul Griepp

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

Major: Interdisciplinary Graduate Studies

Program of Study Committee:
Carolina Cruz-Neira, Major Professor
Steven M. Herrnstadt
Ann D. Thompson

Iowa State University
Ames, Iowa
2001

Graduate College
Iowa State University

This is to certify that the Master's thesis of

Timothy Paul Griepp

has met the thesis requirements of Iowa State University

Major Professor

For the Major Program

Dedication

this work is dedicated to the One whom

has given me the gift of life in all its abundance

Table of Contents

List of Figures

List of Tables

Acknowledgments

For the labors performed ahead of me, I would like to the VR Juggler development team. They have produced a work of undeniable quality, without which, this research would not have been. A special thank you goes out to Allen, for enduring my many questions.

For her pursuit of excellence, her guidance, and her appreciation for both the arts and the sciences, I recognize Carolina Cruz-Neira.

For the fun, creative, hard-working, atmosphere it maintains, for the staff that make it all possible, and for all the cool “toys” it brought into my world, I acknowledge VRAC.

And for their enduring love, faith, and hope, I lift up my parents, whose unwavering encouragement and unceasing prayers have brought me to where I am today.

Abstract

Virtual reality is a tremendous tool and a powerful catalyst of modern scientific and design achievements. These achievements, however, require users to exhibit a highly technical background beyond their area of expertise. With the consideration that large-scale immersive visualization systems require detailed knowledge of the hardware and software that render and maintain their imagery, we recognize that these systems are not easily usable by non-computer experts.

The research presented in this document discusses the design and implementation of an interpreted language for the rapid development of immersive applications. The design is based on a specialized XML schema and 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. This interface reduces the programming requirements of advanced virtual worlds development, such that a significant number of digital artists, previously held at bay, will have the means to produce powerful and creative virtual environments.

Chapter 1: Introduction

Motivation

Virtual reality (VR) is a tremendous tool and a powerful catalyst for pushing the envelope of modern scientific and design achievements. The realizations of these advancements 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 [Keep93]. 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. In light of this computer scientists need to "hide" the complexity of graphics technology. As such, artists can use the technology and exploit it to its full potential.

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.

Scope of Work

The research presented in this document focuses on the effective use of XML-based control interfaces for rapid development of synthetic environment (SE) applications. We design and implement an XML Schema as a layer over VR Juggler [Bierbaum00] to provide a framework for a group of VR applications.

Our goal is the encapsulation of navigation, interaction, and behavior methods and to identify the level of complexity of the applications that can be built with our XML schema. To achieve this goal, the research will be conducted in the following stages:

Review related work

Define development components of a VR application

Design the XML schema

Design testbed application

Results and discussion

Recommendations for future work in this area.

Definition of Terms

Navigation

Throughout this paper *navigation* will be defined as the process of moving about a virtual world independent of one's position or orientation within the physical space of the real world. That is, it will be used in within the context of *locomotion* [Chance98] or *travel* [Bowman98] through an environment verses *piloting* [Loomis93] [Péruch97] or *wayfinding* [Darken96], which reference the process of user orientation.

Interaction

The present working definition of interaction according to ACM SIGCHI [Hewett96] is as follows: Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them.

Immersion

As defined by [Stuart01], *immersion* is "the presentation of sensory cues that convey, perceptually to users that they're surrounded by a computer-generated environment."

Virtual Environment

From [Merriam93] we learn that an *environment* relates to the circumstances, objects, or conditions by which one is surrounded. It, therefore, follows that a *virtual environment* is

an interactive computer-generated set of conditions and objects you can perceive and with which you can interact [Stuart01]. This definition may also be extended to the term *synthetic environment* and on occasion to *virtual reality*. The use of the term virtual reality, however, is often extended to a broader scope. According to [Pimentel95], *virtual reality* is the immersive, interactive experience based on real-time 3D graphic images generated by a computer. The term is further explained by [Cruz95b] as "immersive, interactive, multi-sensory, viewer-centered, three-dimensional, computer-generated environments and the combination of technologies required to build these environments".

Schema

Using its traditional definition a schema is the structure of a database system, described in a formal language supported by the database management system. When related to XML, schemas express shared vocabularies and allow machines to carry out rules made by people [W3C01]. They provide a means for defining the structure, content and semantics of XML documents.

Scene Graph

"A *scene graph* (scenegraph) holds the data that defines a virtual world. The scene graph includes low-level descriptions of object geometry as their appearance, as well as higher-level, spatial information, such as specifying positions, animations, and transformations of objects, as well as additional application-specific data." [Eckel97]

Chapter 2: Background

"Early in the eighteenth century, [Gotfried Wilhelm] Leibniz envisioned what might fairly be called the first reality engine. Central to the argument of the Theodicy (1710), is the claim that the mind of God comprehends an infinity of possible worlds, each of which exists *in potentia*. Of these, only one was brought into being, because only one – the actual world in which we live – fulfils the divine plan for creation. [Keep93]"

And so it is that we transition to the 21st century; wherein, we stand at the edge of a realm equipped to empower each to envision worlds of their own. Make no mistake, these "made worlds", and our vision do not compare with the grander of our physical existence. They do, nevertheless, provide us a vehicle of exploration: Exploration of physical and emotional experience. Even the prospect of spiritual experience is given light in this burgeoning area of new technology. In the words of Howard Rheingold: "The computer will change the way we create and express ourselves, whether the form of expression is writing, music, painting, sculpture, or virtual reality. The computer will change the way we think about writing, music, painting, sculpture, and reality. With computers as partners, new worlds of expression will unfold. [Rheingold92] " Although we may now observe the foothills of this future, it is not yet within our reach. Nor can we fully comprehend what this future will bring with it. Much of our present challenge is linked not only to the technologies of artificial sensation, but also to the clear absence of a language that transcends both human and computer communication.

A structuralist view of languages provides the theoretical foundation for designing all of these systems. Such a view will be at the foundation of the virtual

> worlds of tomorrow. Virtual realities will consist of worlds of sounds, visual worlds, perhaps worlds of touch, smell, and taste. Each of these worlds will be created by computer engines driven by rule-governed languages, each with its own grammar. [Holtzman94]

Jaron Lanier, who claimed, “Information is alienated experience,” has provided us a great deal to think about towards this end. Revered as virtual reality’s father figure, Lanier recognized; if such a structural language were to exist it must be highly palatable to the human thought process. In his early work, he explored methods of replacing alpha-numeric strings with pictograms as part of a new computer language. In many ways, virtual world’s are the pinnacle of such pursuits. “When you make a program and send it to somebody else, especially if that program is an interactive simulation, it as if you are making a new world, a fusion of the symbolic and natural elements. Instead of communicating symbols like letters, numbers, pictures, or musical notes, you are creating miniature universes that have their own internal mysteries to be discovered. [Rheingold92]"

Thomas A. Furness III illustrated this point as follows:

> We were still requiring people to be “computer-like”… that is, we had to interact with the computer on the computer’s terms, using the computer’s arcane language, etc. Furthermore, we were constrained by an archaic keyboard and a computer screen that compressed the mind of the computer into two-dimensional symbols projected on a plate of glass. I felt that we needed the same paradigm shift that I had experienced in VCASS that first day. We needed to break the glass and enter into the mind of the machine and have the computer create a three-dimensional world with which we could interact using our natural abilities. I realized, instead of

> making the human computer-like, we had to do the opposite – to make the computer "human-like." This is the only way that we could get bandwidth to the brain and allow the machine to become a friendly, transparent, and seamless tool and extension of our intellect. I felt that this paradigm shift is the key to the future of computing and of the information highway. I felt that better use of information technology could ultimately help civilization in solving many of the pervasive problems that are confronting us. [Moody99]

We do stand at the edge of a realm equipped to empower each to develop virtual worlds of their own. As of this writing, however, most would be designers are not yet equipped for such development tasks. The required knowledge-base is highly advanced on both aesthetic and technical fronts; and a qualified rule-governed language does not exist which transcends the communication gaps between disciplines. Presently there are two approaches to software tools for the construction of VR environments. One approach is for highly technical people, requiring in-depth knowledge of programming such as C++, operating systems, multithreading and many other complex topics from computer science. This approach puts a lot of power in the hands of the developer by giving access to the core of the system being programmed, however, the number of people with skills required to use such tools limits their applicability and usefulness. The other approach uses scripting languages "on top" of the previously described tools. This approach builds upon an effort to decrease the programming complexity and increase the user aptitude and zeal for the development of virtual environments. These scripted languages, however, tend to be focused more towards 3D desktop applications than to immersive VR. Furthermore, they lack the flexibility of a user interface that transcends the underling technology and development tools.

VR Toolkits

Software tools solutions such as Avocado [Dai97], CAVELib [Cruz95a], Lightning [Landauer97], MR Toolkit [Shaw93], WorldToolKit [Sense8a], Vega [Vega01], and VR Juggler [Bierbaum00] have made significant ground 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. The CAVE Library and VR Juggler, for example, isolate developers from the complexities of hardware specific programming requirements. They also address issues of cross-platform development. VR Juggler has taken specific steps in overcoming known limitations of the CAVE Libraries, with regards to extensibility and distributed environments [Bierbaum98]. These tools, however, do not have an interface capable of masking the complexity of graphics programming from the non-technical developer. As such, developing software with these tools requires advanced knowledge of topics such as scene graph management, matrix transformations, coordinate systems, and collision detection. MR Toolkit holds similar technical knowledge requirements, however, extensions have been written to enhance its native capabilities. A tool called the *Environment Manager* has been programmed which allows a developer to create a virtual environment using a text-based description file [Wang95]. This system, however, lacks the extensibility required for use across diverse hardware installations. Taking a different direction, Avocado and Lightning strive to simplify the process of real-time graphics development through a combination of traditional programming and scripting languages. Although these approaches 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.

Interpreted Language Interfaces

Other solutions, such as Alice [Conway00], Obliq-3D [Najork94], TBAG [Elliot94], and WorldUp [Sense8b] have made advances using scripting languages and visual interfaces to develop interactive 3D environments.

> Alice and Obliq-3D use an object-oriented interpreted language to allow for rapid prototyping. Both separate the application from the rendering. (Alice uses a process to perform the rendering; Obliq-3D uses a separate animation server thread.) Both systems allow for hierarchical geometry, that is, graphical objects that contain other graphical objects. However, the two systems differ in their basic animation model: Alice treats a scene as a hierarchy of graphical objects, each of which has a list of *action routines* which are called each frame of the simulation, and which are responsible for changing the internal state of the object. New animation effects are created by defining new action routines and adding them to the graphical object. Obliq-3D, on the other hand, uses *properties* to specify the appearance of objects; these properties are inherently time-variant. In other words, Alice performs frame-based animation, whereas Obliq-3D has an explicit notion of time. [Najork94]

TBAG, a research project of Sun Microsystems, took additional strides by including a compiled language interface in addition to its interpreted language system. With such a

structure, TBAG provided both an entry-level mechanism for users and a high level conduit for extending the solutions base capabilities [Elliot94]. The grammar of these systems, however, is cumbersome to many non-technical users. The process of managing numerous objects, actions, and their associated constraints can quickly become overbearing for some users.

Perhaps going the farthest in providing a solution with the non-programmer in mind the Alice project exists to "make it easy for novices to develop interesting 3D environments and to explore the new medium of interactive 3D graphics [Alice01]." The software in essence provides a rapid prototyping system for creating interactive computer graphics applications. With a focus on 3D object behavior, the system uses scripting to control object behavior and appearance as directed by user input via mouse and keyboard. Unlike many of the interpreted language interfaces designed for interactive 3D graphics, Alice has found great success in developing a grammar structure free from cryptic or specialized language. With many similarities to LOGO – the Turtle Graphics programming language – Alice maintains a friendly interface and an intuitive command structure that reads naturally to even the most casual and uninformed observer. Like Obliq-3D and TBAG, nevertheless, the command structure is highly specific to the constraints of the underlying tool. These resources fail to provide a generalized solution that may be used in conjunction with the tools that surface as being best equipped to support varied virtual environment system solutions.

Although the aforementioned tools and interfaces have taken great strides in bridging the gap between the novice and advanced users of interactive 3D graphics, there remain many steps to be taken. In simplifying the process of application development, the main challenge is to provide a development layer on top of powerful tools such as the CAVELib

and VR Juggler. A layer that simplifies the programming process but also allows for full access to the capabilities of the underling tools.

XML Specifications

On a converging front, solutions such as XGL [XGL01] and X3D [X3D01] are providing new opportunities in the way of open standard geometry formats. 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 [XGL01]." In other words, it is a file format wherein a user may export 3D geometry and its associated rendering treatments from a supporting application and view the geometry, with the same rendering characteristics, in another application. X3D also advances in the creation and deployment of 3D graphics. Revisiting the traditional VRML format, X3D is the 'New Generation' specification for Web3D [X3D01]. Capitalizing on these advances, CONTIGRA [Dachselt01] has been developed as an XML-based approach to constructing interactive 3D graphics applications for the web. The acronym CONTIGRA stands for: Component OrieNted Three-Dimensional Interactive Graphical Applications. The solution is built upon three multi-layered markup languages labeled *SceneGraph*, *SceneComponent*, and *Scene*. This trilogy of XML schemas is intended to allow easy, declarative, and interdisciplinary authoring of 3D applications [Dachselt01]. The *SceneGraph* schema is used as an extension of X3D to implement a 3D web component through the description of its geometry and behavior. Similarly, *SceneComponent* defines the component interfaces. Finally, *Scene* is used as a high-level configuration language for component integration. Although powerful in its arena, the scope

of CONTIGRA is presently limited to web based applications. It does not maintain nor indicate future efforts designed to bring its strengths to large-scale immersive visualization systems.

A solution known as 3dml 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 [Figueroa01]. 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. Additional limitations exist in its lack of support for lighting, level-of-detail, animation, and user feedback configuration.

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.

Chapter 3: Components of VR Application Development

User Persona

The number of skilled 3D artists has increased dramatically in resent years. Significant reductions in the entry barriers, previously hindering artists, have paved the way for students to begin honing their modeling skills much earlier in their careers. The accessibility to both hardware and software tools have become far more tangible as a result of the smaller margins and wider marketing campaigns.

Consider the following scenario: A highly talented digital 3D artist who 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 environment's 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.

Whether focusing on a software engineer attempting to develop an aesthetically enriched simulation or an artist striving to produce a real-time interactive exhibit; a strong balance of skills in the realm of the analytical and holistic design is critical. Considering the principles of game development, Moody states: [Moody99] 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

User Feedback

Exploring these requirements in greater detail we have isolated a series of application features and user driven events (Table 1). 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.

	Application Features	Runtime User Manipulation
Geometry Placement	Load geometry Place geometry	Reposition
Geometry Dynamics	Scene driven object animation Algorithmically driven animation	User driven object animation Activate and disable animation
Geometry Visibility	Level-of-detail masking	Switch based masking
Scene Navigation	Gravity Collision	Toggle Constraints Acceleration Deceleration Rotation
Scene Interaction		Reposition objects – including geometry, lights, and audio sources
Lighting	Global Spot	Reposition Activate and disable lighting
Audio	Ambient Dynamic	Reposition Activate and disable audio
User Feedback	Generation of content	Activate and disable feedback

Table 1 Typical Application Features

Software Goals

Our goal for this thesis 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++ and using instead 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 [Rohlf00] and VR Juggler. 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 lie within the following categories: Geometry placement, geometry dynamics, geometry visibility, scene navigation, scene interaction, lighting, audio, and user feedback. Through 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.

Navigational Tools and Interaction Devices

In an exploration of locomotion taxonomies, [Arns01] has provided us an overview of guidelines for constructing navigational methods. One of the main points of such systems is the relationship between navigational paradigms and the physical devices used to implement those paradigms. In her work, Arns discusses methods of VR locomotion and how various interaction and display devices relate to concepts of rotation and translation. Limiting our discussion to interaction devices and considering them the primary tools of both navigation and interaction, we recognize wands, gloves, and mock-ups as the leading interfaces to large-scale visualization environments. The capabilities of these tools are often augmented by additional resources such keyboard and mouse interaction, handheld computers, and specialized input devices such as the CubicMouse™ [Fakespace01]. Looking at the broad application of these technologies, however, we recognize they are, for the most part, all designed to serve two basic requirements. These systems provide an application positional data and act as a command interfaces between the user and the software.

For issues of simplicity, we have chosen to focus our attention on the wand as the interaction device used by XJ Nav. We have elected to work with the wand as it is amongst the most common and most accessible interaction devices presently being used with large-scale visualization systems. We are aware that in order to accommodate other devices, there is a need to investigate other layers of abstraction that are beyond the scope of the research presented here. At the present time, however, there is need for more research on the abstraction layers capable of isolating the application programmer from the complexities of diverse input systems.

The specific wand device we have used in our research is a product known as RemotePoint RF (Figure 1) [Interlink01]. This device, manufactured by Interlink Electronics, acts as a wireless input device and is based upon the protocols of the PS2 mouse standard. We have extended the capabilities of this device by attaching to it a sensor that communicates with the MotionStar Wireless tracking system (Figure 2) [Ascension01] developed by Ascension Technology Corporation.

Figure 1 RemotePoint RF™

Figure 2 MotionStar Wireless® tracking system

We have configured our installation such that the application may respond to five user buttons on the interaction device. These five buttons allow the application designer thirty-five unique button combinations that may be referenced within the XJL configuration. Although there are clear reasons to desire the flexibility a large array of input commands provides, Boyd reminds us that the added flexibility often brings undesired complexities. “People in general are quick to learn simpler interfaces and what slows them down with the others is the complexity of the command language and the indirectness of the mapping from user input to system response [Boyd95].”

Chapter 4: 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 develop advanced virtual environment applications without facing the riggers 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 3.

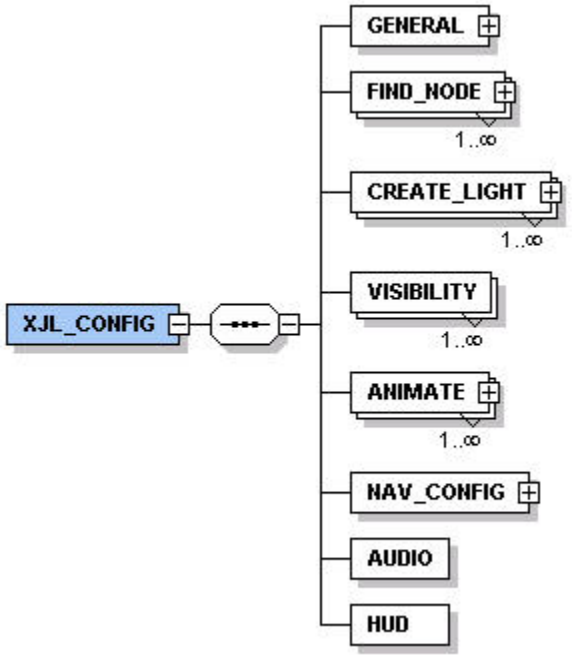

Figure 3 XJL_CONFIG Elements

A hierarchical overview of the XJL elements and attributes is provided in Figure 4. These elements may be used with great flexibility to devise a wide array of navigation and interaction techniques. They may also be used to control object and application behavior.

By cross-linking multiple methods of navigation, interaction, and behavior, highly advanced systems of interaction may be explored.

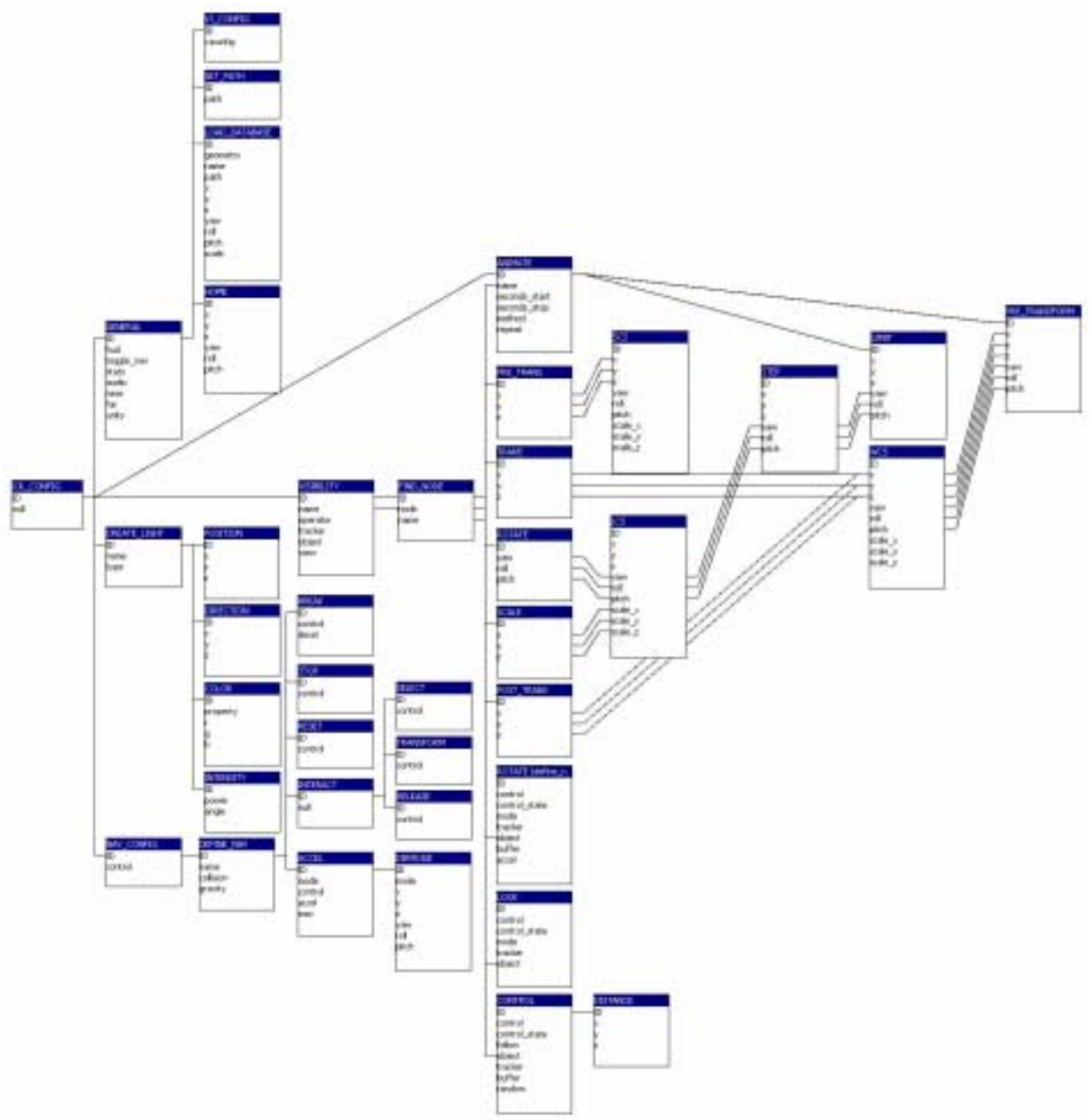

Figure 4 XJL Relationship Diagram

A detailed look at each of the root elements follows. As we explore each of these areas we will show how XJL addresses the development issues surrounding geometry placement, geometry dynamics, geometry visibility, scene navigation and interaction, lighting, audio, and user feedback.

General Configuration

The GENERAL element is where basic application configuration takes place. It is here that flags are set that influence the low level configuration of an application. Table 2 defines the element's maintained at the base level of the GENERAL element.

Table 2 XJL_CONFIG Attributes

Name	Type	Default	Enumerations
hud	string	false	true false
toggle_nav	string	true	true false
stats	string	false	true false
audio	string	false	true false
near	float		
far	float		
units	string	feet	inches feet yards mile centimeters meters kilometer

The *hud* attribute informs the application if a heads up display will be available to the user during run-time. The *stats* and *audio* attributes operate in a similar manner, dictating the availability of statistical feedback and auditory objects. *Toggle_nav* allows the application designer to enable or restrict an end users ability to cycle through various navigators that have been configured by the designer in the NAV_CONFIG element. The *hud, toggle_nav,*

and *audio* attributes are redundant, in that these features may also be disabled by removing their associated elements from the configuration file; or in the case of toggle_nav by removing the control attribute from the NAV_CONFIG element. However, for ease of configuration we have included the listed attributes under the GENERAL element. In doing this, a central medium for disabling these features is made available. As such, the application designer does not need to remove detailed elements that otherwise would need to be reinstated for later use.

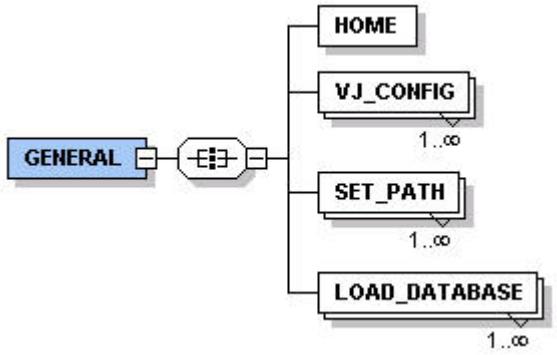

Figure 5 GENERAL Elements

GENERAL contains the sub-elements listed shown in Figure 5. The element HOME is where the user sets the starting position and orientation of an application. By default these values are set to zero.

Table 3 HOME Attributes

Name	Type	Default	Enumerations
x	float	0.0	
y	float	0.0	
z	float	0.0	
yaw	float	0.0	
roll	float	0.0	
pitch	float	0.0	

It is in the VJ_CONFIG element that the application designer references any required VR Juggler configuration files. This is an unbounded element such that the user may load as many configuration files as are required by their application.

Table 4 VJ_CONFIG Attributes

Name	Type	Default	Enumerations
vjconfig	string		

SET_PATH, an unbounded element uses the attribute *path* to provide an application any search paths required for geometry or texture data used by the applications dataset.

Table 5 SET_PATH Attributes

Name	Type	Default	Enumerations
path	string		

LOAD_DATABASE is the final element contained within GENERAL. This element is also unbounded. It is through this element that the user loads and names their database. Presently all datasets supported by OpenGL Performer's loader are supported by XJ Nav.

Table 6 LOAD_DATABASE Attributes

Name	Type	Default	Enumerations
geometry	string		
name	string		
path	string		
x	float	0.0	
y	float	0.0	
z	float	0.0	
yaw	float	0.0	
roll	float	0.0	
pitch	float	0.0	

Find Node

The element FIND_NODE serves two purposes. The first, as might be expected, is to search through the geometric database, isolate, and name specific objects for later use. These objects are set apart as DCS nodes. DCS nodes, otherwise known as "Dynamic Coordinate System" nodes are used to transform objects over time. Unlike static coordinate system (SCS) nodes, objects that may be initialized according to some arbitrary transformation, but which remain fixed to that transformation for the duration of its existence, the transformations of a DCS node may be modified throughout its life. The DCS nodes initialized by FIND_NODE are referenced for their dynamic functionality as part of the ANIMATE and VISIBILITY elements. Furthermore, they may be used as tracked objects as part of the navigation and interaction routines. The second function of FIND_NODE allows the user to pre-transform specific geometry of their dataset. Through FIND_NODE's sub-elements, a set of transformation matrixes can be produced that are statically applied to the node and its surrounding scene graph hierarchy. As such, the transformations will remain intact through the duration of the applications run cycle.

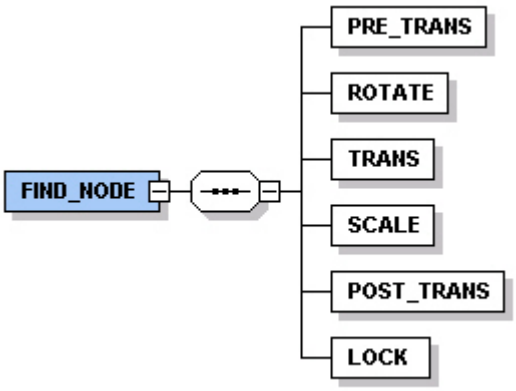

Figure 6 FIND_NODE Elements

Table 7 FIND_NODE Attributes

Name	Type	Default	Enumerations
node	string		
center_node	string		
name	string		
PRE_TRANS /x	float	0.0	
PRE_TRANS /y	float	0.0	
PRE_TRANS /z	float	0.0	
ROTATE /x	float	0.0	
ROTATE /y	float	0.0	
ROTATE /z	float	0.0	
TRANS /x	float	1.0	
TRANS /y	float	1.0	
TRANS /z	float	1.0	
SCALE /x	float	1.0	
SCALE /y	float	1.0	
SCALE /z	float	1.0	
POST _TRANS /x	float	0.0	
POST _TRANS /y	float	0.0	
POST _TRANS /z	float	0.0	
LOCK/x	bool	false	true false
LOCK/y	bool	false	true false
LOCK/z	bool	false	true false
LOCK/yaw	bool	false	true false
LOCK/roll	bool	false	true false
LOCK/pitch	bool	false	true false

As indicated by Table 7, FIND_NODE is used to isolate and name specific geometries as well as to apply a series of transformations to those objects. Figure 7 illustrates the application of these transformations to the scene graph in greater detail. In order to facilitate transformations about an objects local coordinate system (LCS) the objects are first transformed to the scene origin. This transformation is applied to an OpenGL

Performer™ static coordinate system (SCS) node. Once brought to the origin, the object may be further manipulated about its local axis; which at this time is located inline with the global or world coordinate system (WCS); that is, the coordinate system under which the scene exists and navigation tasks are executed. Following the transformation of the objects local identity, it is repositioned according to the inverse of the previously defined static transformation. This is the operation that brought the LCS inline with the WCS. Returning the object in this way brings it to a resting position wherein all transformations are rendered relative to the original placement of the object.

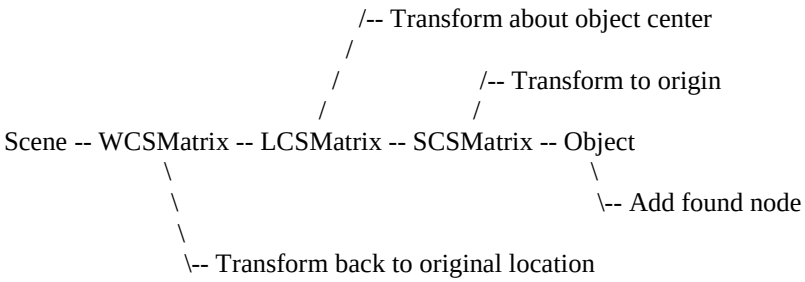

Figure 7 FIND_NODE Matrix Transformations

In addition to being a resource for the placement or repositioning of geometry, FIND_NODE is a mechanism for isolating and naming specific elements of a scene's geometry. After an object has been named it may be accessed by other XJL components. These objects are subsequently considered "tracked" and may be used both as agents of interaction and objects upon which the user, or other tracked objects, may interact. The relationship between FIND_NODE and other elements is further illustrated in Figure 8.

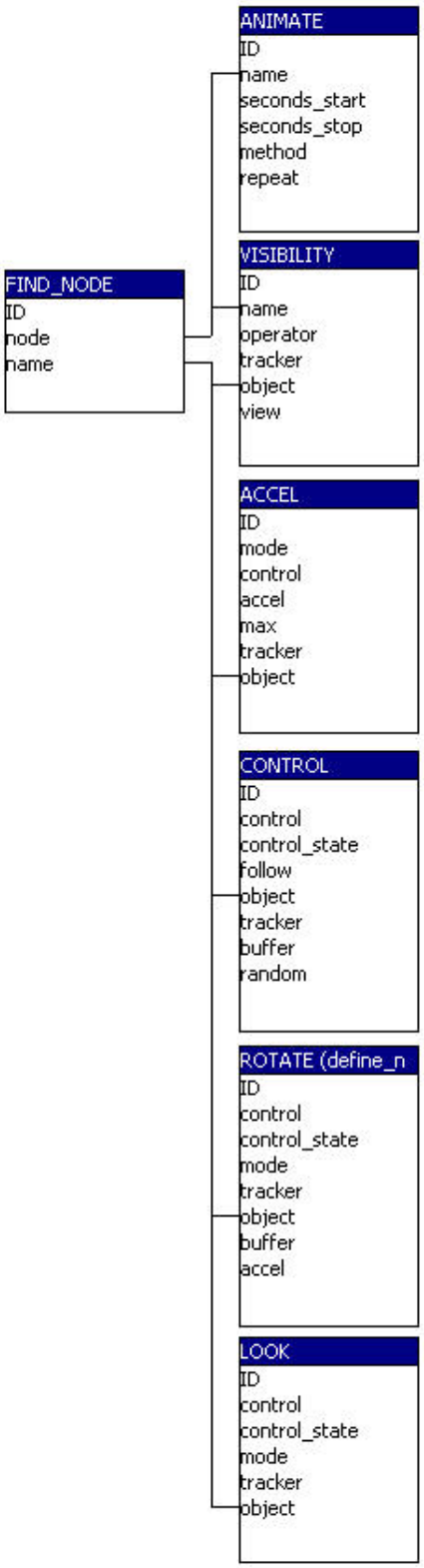

Figure 8 FIND_NODE Relationship Diagram

Lighting

Lights may be defined and added to a scene graph through the use of the lighting element. Configuration attributes provide a means of defining a lights position, color, and intensity. At the root level of the CREATE_LIGHT element the attributes *name* and *type* may be set. The value set to the *name* attribute is used by other XJL elements to dynamically redefine a lights position and visibility. The *type* attribute may be used to configure a light as a global or spotlight source of illumination. When set as global, several of the position and intensity attributes will be ignored. CREATE_LIGHT is unbounded; as such any number of lights may be defined though the element. As illustrated in Figure 9, the COLOR element is referenced three times within a given light's configuration. It is in this way that unique color attributes are assigned to the diffuse, ambient, and specular components of the lighting source. The INTENSITY element provides access to attributes describing the intensity and diffusion of a light source. The *focus* attribute addresses the concentration of a light source. A light's intensity is the highest at the center of its focal point and fades or is "attenuated" towards its edge. The level of attenuation is determined by raising the cosine of the angle, defined by the difference between the direction of the light and the position of a lit vertex, to the value of the *focus* attribute. The *angle* attribute controls a spotlight's cutoff. The value given by *angle* represents the angler distance between the focal point of a light and a lights edge. The *constant, linear,* and *quadratic* attenuation attributes are closely related to *focus.* These attributes effect how the energy of a light fades towards the perimeter of its influence.

The attenuation is calculated through the following formula:

$$attenuation\ \ factor = \frac{1}{k_c + k_c d + k_q d^2}$$

where

d = distance between the light's position and the vertex

k_c = INTENSITY/ATTEN/constant

k_l = INTENSITY/ATTEN/linear

k_q = INTENSITY/ATTEN/quadratic

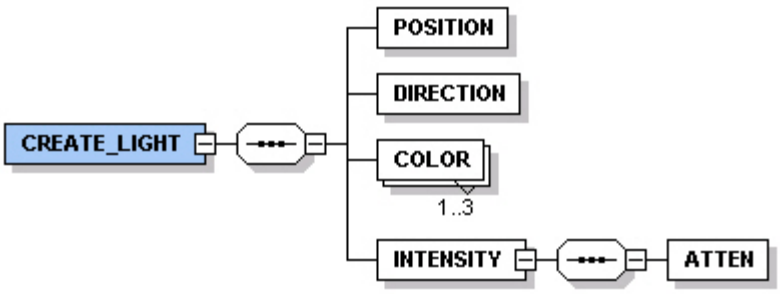

Figure 9 CREATE_LIGHT Elements

Table 8 CREATE_LIGHT Attributes

Name	Type	Default	Enumerations
name	string		
type	string		SPOTLIGHT GLOBAL
POSITION /x	float	0.0	
POSITION /y	float	0.0	
POSITION /z	float	0.0	
DIRECTION /x	float	0.0	
DIRECTION /y	float	0.0	
DIRECTION /z	float	0.0	
COLOR/property	string		DIFFUSE AMBIENT SPECULAR
COLOR /r	float	1.0	
COLOR /g	float	1.0	
COLOR /b	float	1.0	
INTENSITY/focus	int	100	
INTENSITY/angle	int	180	
ATTEN/constant	float	1.0	
ATTEN/linear	float	0.0	
ATTEN/quadratic	float	0.0	

Visibility

Visibility and level-of-detail (LOD) behavior may be defined though the use of the VISIBILITY element. Level-of-detail pertains to virtual world datasets wherein multiple instances of objects are stored at varying levels of complexity. Each of these related objects are considered a LOD node. Using level-of detail logic, an application designer may elect which LOD node is displayed based on the users proximity to object of interest. Using the VISIBILITY element an application designer may manipulate a node previously declared in the FIND_NODE element. By using operators such as less-then or greater-then, the designer is able to toggle the viewing state of an object. The *view* attributed determines whether an

object is visible or hidden in its default state. If *operator* is set to "less" and view is set to "show"; an object will only be visible if it the selected tracker or tracked object exists within the distance range defined by the *x*, *y*, and *z* distance attributes. Tracked objects, DCS nodes that have been declared in FIND_NODE, may be used as keys to unlock features of a world. By selecting and "grabbing" such an object through the interaction routine a user may arbitrarily relocate the object bringing it within the control range defined by the distance attributes. Building upon this idea, the *operator* "selected" or "unselected" may be used. In this way, the *view* attribute will be enforced according the tracked object's state of selection.

Table 9 VISIBILITY Attributes

Name	Type	Default	Enumerations
name	string		
operator	string	less	less greater selected unselected
tracker	string	head	head wand object
object	string		
view	string	show	hide show
x	float	0.0	
y	float	0.0	
z	float	0.0	

Animation Configuration

Triggered and unrestricted animation sequences may be defined within the ANIMATE element. The element acts upon tracked objects. The *method* attribute is where the user configures how coordinate based transformations cycle between iterations. When set to swing the sequence will animate to the outer constraints of its transformation and return to its original position using the inverse of its previous motion path. The loop method

operates by snapping the object back to its original position directly after it reaches the limit of its transformation. The *repeat* attribute sets the number of times a given animation runs. The sequence counter iterates each time an object returns to its starting position. With this constraint, a swing based animation sequence will take twice as long to iterate as a similar animation operating under the loop method. If *repeat* is set to negative one, the sequence will operate indefinitely or according to the constraints of the CONTROL element. Operating in a similar way, *seconds_start* and *seconds_stop* may be used to control when an animation sequence is to start and the length of time it operates. The clock, dedicated to a specific animation, which drives the operation of *seconds_start* and *seconds_stop*, is returned to zero upon encountering an animation sequence trigger from the CONTROL element. In this way the user may arbitrarily trigger an object's animation sequence or sequences any number of times during execution of an application.

PRE_TRANSFORM allows the designer to manipulate an objects center of rotation. The inverse of all pre-animation transformations are applied to the object's matrix following the addition of any animation movements.

The attributes of the LIMIT element set the limits of a coordinates based animation sequence. The *x, y,* and *z* values are measured in linear units according to the units attribute set in the GENERAL element. The attributes of the STEP element operate on the same unit system. An angular measurement is used for *yaw, roll*, and *pitch* elements. These values of these attributes are measured in degrees.

Table 10 ANIMATE Attributes

Name	Type	Default	Enumerations
name	string		
method	string	swing	swing loop static
repeat	int	-1	
seconds_start	int	0	
seconds_stop	int	-1	
PRE_TRANSFORM/x	float	0.0	
PRE_TRANSFORM/y	float	0.0	
PRE_TRANSFORM/z	float	0.0	
PRE_TRANSFORM/yaw	float	0.0	
PRE_TRANSFORM/roll	float	0.0	
PRE_TRANSFORM/pitch	float	0.0	
LIMIT/x	float	0.0	
LIMIT/y	float	0.0	
LIMIT/z	float	0.0	
LIMIT/yaw	float	0.0	
LIMIT/roll	float	0.0	
LIMIT/pitch	float	0.0	
STEP/x	float	0.0	
STEP/y	float	0.0	
STEP/z	float	0.0	
STEP/yaw	float	0.0	
STEP/roll	float	0.0	
STEP/pitch	float	0.0	

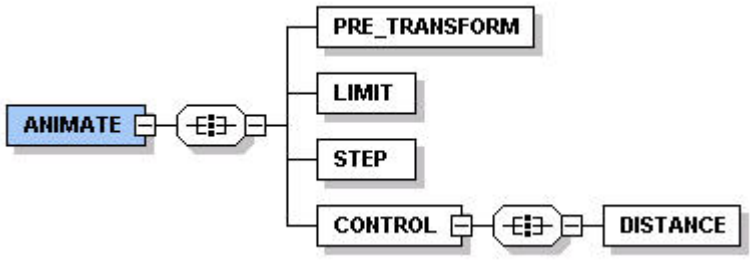

Figure 10 ANIMATE Elements

Related to Figure 7, which addresses the matrix transformations of FIND_NODE, Figure 11 pertains to the transformations made by the ANIMATE element. In FIND_NODE a static transformation is applied to the object that repositions the local coordinate system of

the object. The effect of this transformation remains intact for the duration of the objects existence. The final placement of an object is augmented by the dynamic coordinate system nodes maintaining the WCS and LCS transformations. Translations are applied to the object with respect to the world coordinate system. This is done to maintain an orderly relationship between the axis of the system and the movement of the object. Alternatively, rotations are applied to the local coordinate system. In this way such operations are performed with respect to the center of the object rather then its position relative to the global coordinate system. If a user desires to apply these transformations from a position other then the center of an object, he may use *pre_transform* to relocate the object with respect to its local axis.

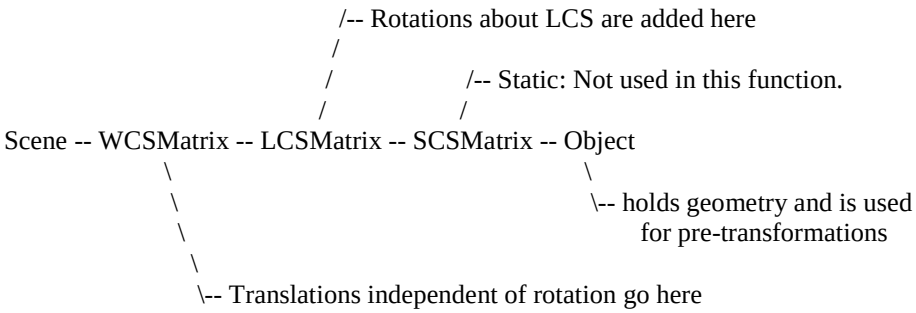

Figure 11 ANIMATE Matrix Transformations

An animation may be initiated at the start of an application or may be triggered by a user or proximity event. The CONTROL element is where these settings are configured. If no trigger event is desired, that is, if the designer does not need to provide the sequence a specific start or stop a null value may be used for the *control* attribute. An interaction device will trigger the event when the device button indicated by the *control* attribute is selected. If the keyword “tracker” is used as the value of this attribute the sequence will be triggered by

the tracker or tracked object indicated by the *tracker* attribute. When the tracking device comes within the spatial range set in the DISTANCE element the sequence will be initiated.

Further influence over a controlled animation is made possible through the *follow* element. This element determines whether an object will shadow the movements of a tracker or tracked object. The *buffer* element is where the application designer sets the distance a tracked object will follow behind the tracked object. Coordinates based transformations may be layered with tracked movements for more complex behaviors. To further diversify an objects movement, the random element may be used. With this element, a noise function is added to the animated objects movement as it exercises its coordinate's transformation or follows a tracked object.

Table 11 ANIMATE/CONTROL Attributes

Name	Type	Default	Enumerations
CONTROL/control	int	00000	bbbbb tracker
CONTROL/control_state	string	toggle	toggle hold
CONTROL/tracker	string	head	head wand object
CONTROL/object	string		
CONTROL/follow	bool	false	true false
CONTROL/buffer_x	float	0.0	
CONTROL/buffer_y	float	0.0	
CONTROL/buffer_z	float	0.0	
CONTROL/random_x	bool	false	true false
CONTROL/random_y	bool	false	true false
CONTROL/random_z	bool	false	true false
CONTROL/DISTANCE/x	float	1.0	
CONTROL/DISTANCE/y	float	1.0	
CONTROL/DISTANCE/z	float	1.0	
CONTROL/DISTANCE/operator	string	less	less greater selected unselected

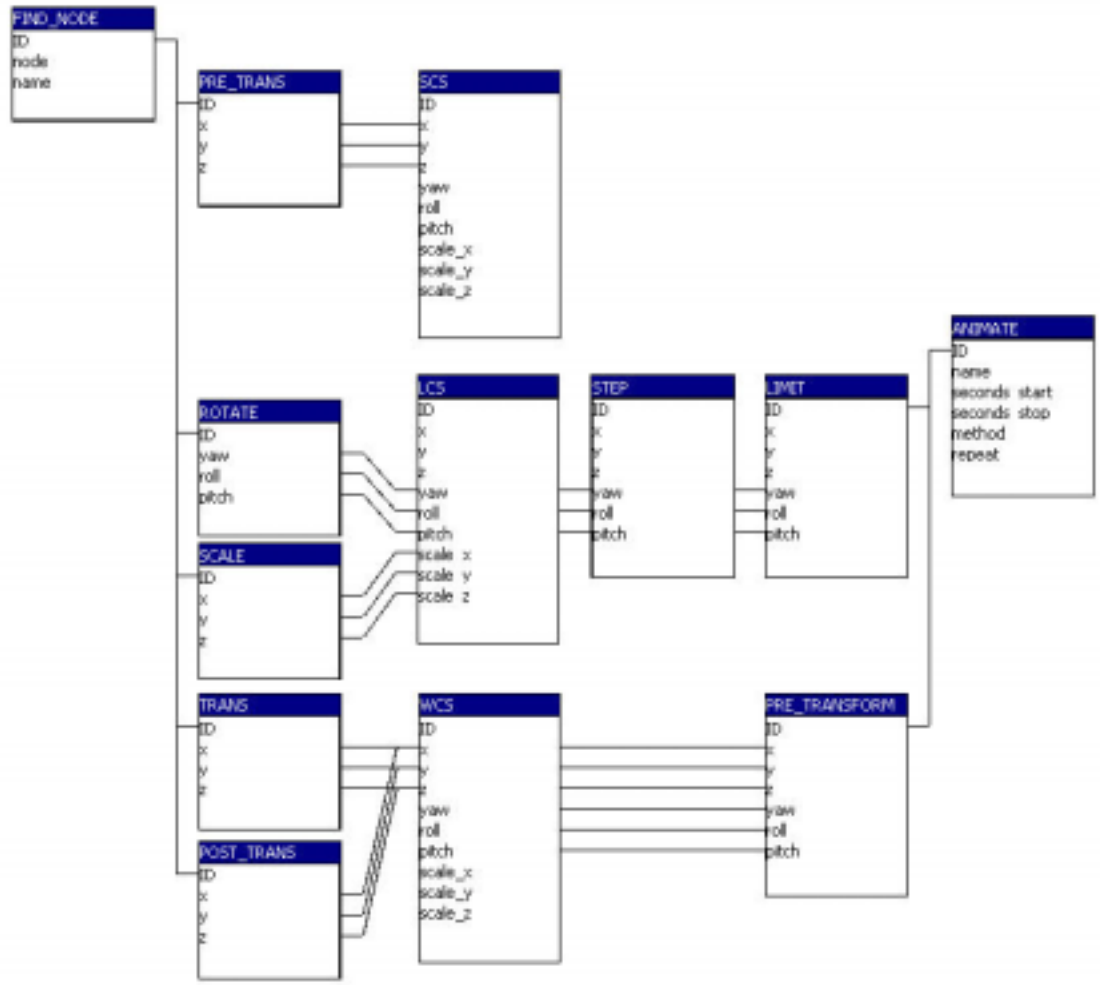

Figure 12 Object Positioning

Figure 12 provides an overview of how various elements are related to the local and world coordinate systems. Each set of transformations build upon its predecessors. In this way an object may follow an algorithmic animation sequence that rests on top of user driven transformations. That is to say a user may interact with, or otherwise manually transform, an object that is actively processing pre-defined animation commands. In this way the pre-defined animation will act relative to the modified position of the object relative to the global coordinate system.

Navigation Configuration

Each navigation method is built on a core set of sub-attributes. A given configuration file may be composed of any number of navigators which may be toggled within the application either by use of the heads up display control or by use of a button controller defined by the user. Within each navigator the user defines the name, collision attributes, momentum and directional controls, as well as viewing perspective, and the enabling of heads up display and geometry selection.

Table 12 NAV_CONFIG Attributes

Name	Type	Default	Enumerations
control	int		bbbbb

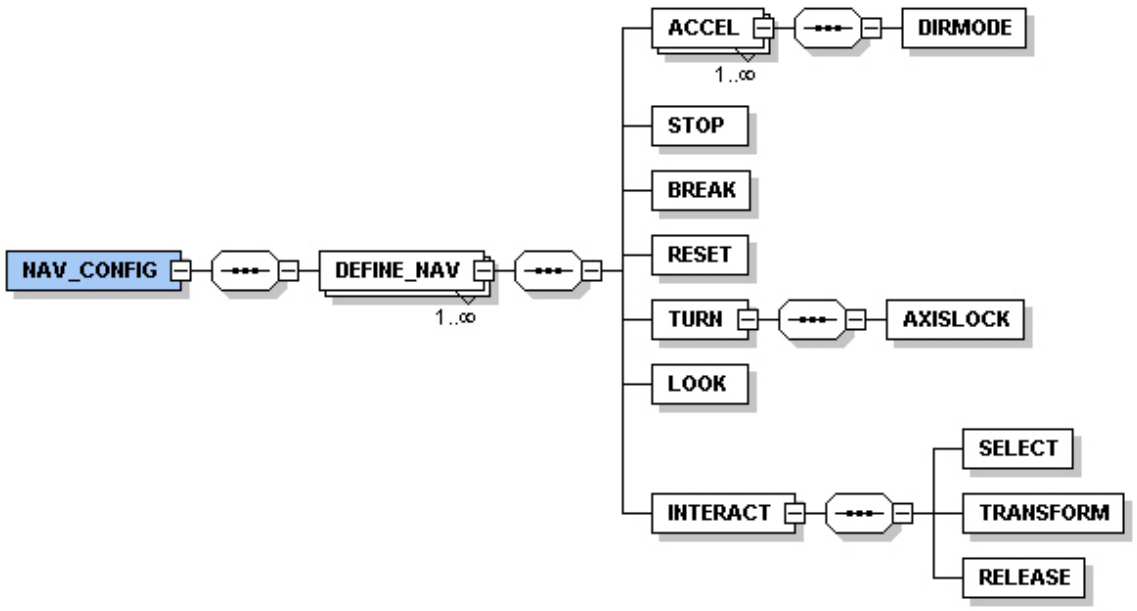

Figure 13 NAV_CONFIG Elements

Name

The navigation method name is used primarily for user feedback. By referencing the user-defined name in the heads up display, the user may quickly identify their navigation mode and view how the navigators cycle through the stack.

Collision

The collision attribute is composed of a true or false state declaration. This flag corresponds to all horizontal collision.

Gravity

Similar to the collision attribute, gravity is also composed of a true or false state declaration. Modifying this flag controls weather or not ground collision is enabled. With gravity set to true the navigator will operate in what is traditionally referred to as drive or walk mode. With gravity set to false the navigator will operate in a fly mode. The effect of gravity is further defined by the use of two additional variables. These values are gravitational acceleration and terminal velocity.

Table 13 DEFINE_NAV Attributes

Name	Type	Default	Enumerations
name	string		
collision	string	true	true false
gravity	string	true	true false

Accel

Each navigator may contain as many accelerators as desired up to the number of control buttons available. If a control button is referenced multiple times within a

navigator's set of accelerators, the accelerator referenced last will take precedence. An accelerator is composed of the attributes listed in Table 14.

Table 14 ACCEL Attributes

Name	Type	Default	Enumerations
mode	string	ramp	ramp hold
control	int		bbbbb
accel	float		
max	float		
DIRMODE/mode	string	tracker	coord tracker
DIRMODE/x	float	0.0	
DIRMODE/y	float	0.0	
DIRMODE/z	float	0.0	
DIRMODE/yaw	float	0.0	
DIRMODE/roll	float	0.0	
DIRMODE/pitch	float	0.0	

A binary control variable is used to reference the button combination linked to a given accelerator. The bits of this control variable read left to right such that the combination 01100 indicates buttons two and three being simultaneously selected on a five-button interface device. Figure 14 illustrates the relationship between an application's interaction manager and the various elements that access the interaction manager through the use of a control attribute.

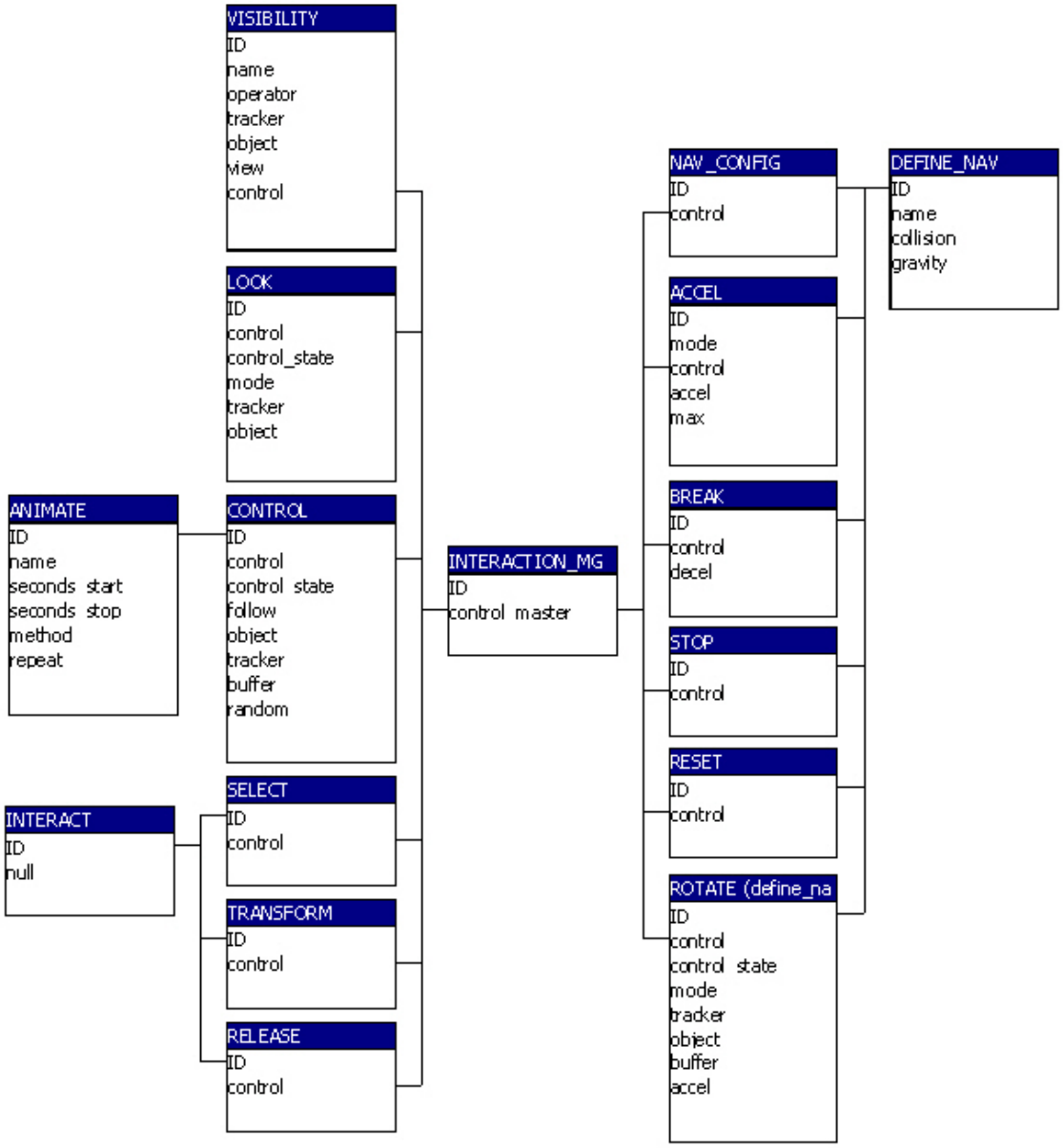

Figure 14 Control Instances

The method by which acceleration takes place is controlled by the *mode* attribute. There are two states to this attribute: ramp and hold. When the relevant control button is selected, the ramp accelerator gradually increases the speed of transversal at a rate of *accel* units per cycle. The speed will accelerate up to the limit set by the variable *max*;

representing the maximum units to be traversed per cycle through its associated accelerator. When the control button is released the speed of transversal will continue at its present rate until influenced by another accelerator, a stop command, or a reset. In comparison, the hold accelerator influences the binary state of motion. So long as the accelerator is selected traversal will be influenced at a rate of *max* units per cycle. When the accelerator is released its influence on motion will subside and return to zero.

The method of directional acceleration is determined by the *mode* attribute of the DIRMODE element. This attribute contains two states: *tracker* and *coord*. When *mode* is set to tracker the directional component of acceleration is obtained by the orientation of the tracker as indicated in the TURN element. The rotation methods will be expanded upon later. In the tracker mode the directional components, if given, are ignored. If *mode* is set to *coord*, however, the direction of acceleration is determined by the value given to each of the six directional components. These components are *x*, *y*, *z*, *yaw*, *roll*, and *pitch*. The *x*, *y*, and *z* components provide for acceleration along each of the give axis. *Yaw*, *roll*, and *pitch* allow the user to build upon these linear movements with angular acceleration. With the accelerators set to relative mode (more on this later) a value give to one of the angular components will rotate the user accordingly. The influence of this rotation will propagate to the linear accelerators where the direction of acceleration is offset from the previous axis. For example: Given an acceleration value of 10 in the x-axis and a roll acceleration of 5. When the acceleration button is held for a single iteration of the program, the user will be rotating at a rate of 5 degrees per second traveling forward, relative to the direction of rotation, at a speed of 10 units per second. Looking closer at this example we can determine

the radius of the arc traversed. Given a 360˚ arc we can determine the number of seconds required to traverse the entire arc.

$$\frac{360°}{\left(\frac{5°}{1\sec ond}\right)} = 72\sec onds$$

We can also calculate the circumference by multiplying the recently discovered time by the distance traveled per second.

$$circumference = 72\sec onds * 10units = 720units$$

Knowing

$$circumference = 2 * \pi * \mathrm{r}$$

We can calculate

$$\frac{circumference}{2 * \pi} = \frac{720}{2 * \pi} = 114.6\mathrm{units} = \mathrm{r}$$

This same operation performed based upon the accelerators set to absolute mode yields a different result. In this second scenario, the rotational and directional components work independently from one another. The direction of acceleration is constant based on the input of the x, y, and z components relative to the global coordinate system. That is, given the example above, after 10 seconds the user will have moved forward 72 units along the x-axis of the system and the user's lookup vector would have rotated 50 degrees. Additional examples are provided in Chapter 6.

The coordinates based acceleration mode provides the user an added level of control by separating the viewing direction from the vector of the motion path. The implications of this functionality are discussed in more detail with the LOOK and TURN elements.

Stop

The *stop* method simply halts all accelerators. The single method of the elements is *control*. Control is used in the same way as the accelerators. It is a binary number related to the buttons of the interaction device. When the associated button on the device is selected the stop routine is activated and all navigational motion for the given navigator is suspended.

Table 15 STOP Attributes

Name	Type	Default	Enumerations
control	int		bbbbb

Break

The *break* method is closely related to stop. The key difference is in the method of deceleration. The stop method is used to bring all accelerators to a full halt. Alternatively, the break method gradually reduces all momentum at a rate of *decel* units per second. Although this method may be used to bring all accelerators to a zero value, its primary focus is to reduce the momentum to an arbitrary value between the current rate of traversal and zero.

Table 16 BREAK Attributes

Name	Type	Default	Enumerations
control	int		bbbbb
decel	int	0	

Reset

Reset positions the user to the applications home position and halts all accelerators. The element used in this method is control. Control relates to the button used on the navigation device to initiate the reset routine.

Table 17 RESET Attributes

Name	Type	Default	Enumerations
control	int	00000	bbbbb

Turn

Turn is the key method used to manipulate a navigational orientation. The method, composed of *control, control_state, mode, tracker, object, buffer,* and *accel* elements, allows a user to define what techniques will be embraced for the control of a world's rotation about the user. Activation of the *turn* method is initiated through the use of a control element. By using a null value for turn's control element, the default state of the method will be realized and the rotation state shall be considered always on. *Control_state* is comprised of two states: *toggle* and *hold*. This element determines whether the influence of the *turn* method is toggled on or off or if it is active only when the control buttons are depressed. (The control state is ignored when the control element is set to a null value.) When set to toggle the method becomes active as soon as the control button is selected. The method will remain active until the control button is selected again. When set to hold, however, the method is active only during the time in which the control button is held.

There are two modes of this *turn* method: These are absolute and relative. When set to *absolute* mode, the coordinate system of the tracker, relative to the coordinate system of

the visualization environment will remain fixed. In this scenario, the relationship between the orientation of the tracker and the direction of traversal will maintain a one to one correlation. (This assumes the accelerator being used for traversal is set to tracker mode.) If the user rotates the wand to some arbitrary position $\{\theta z, \theta x, \theta y\}$ relative to the wands native position, wherein its rotational position is inline with the rotational coordinates of the system, the users position will be translated across the scene by some distance in the direction of $\{\theta z, \theta x, \theta y\}$. If the mode is set to *relative*, however, the scene will rotate about the user in a direction that is opposite the wands rotation relative to the coordinates of the visualization system. The relative rotation mode is further defined by AXISLOCK and the elements *buffer* and *accel*. AXISLOCK, a sub-element of TURN is used to constrain the degrees of freedom upon which the method operates. The nature of relative rotation is such that when the tracker is positioned, inline with the coordinates of the visualization system, the environment is free from rotational movement about the user. To provide an increased level of navigational stability the *buffer* element is used. This element holds a value that represents the number of degrees off axis the tracker may be rotated before its rotation influences the rotation of the environment. If element =“10” the user may rotate the tracker freely within a range plus or minus ten degrees of the systems global axis. As the user rotates the tracker outside this range, the environment will rotate about the user according to the constraints of the *accel* element. The value of the accel, measured by the number of degrees the world will rotate per second for each degree of wand rotation beyond the rotation buffer.

Let us know explore an example. If buffer =”10” and accel =”2”, a rotation of 12 degrees about the y-axis will yield a negative 4 degree counter rotation of the world about the

user for every second the tracker held at the previously defined rotation. In most applications the user would use a much smaller accel value such that the speed of rotation is more easily managed and the acceleration of such a rotation is less extreme. Given the same buffer value and an accel value equal to 0.25 for example would yield a counter rotation of only 0.5 degrees per second. As we increase the tracker rotation to a larger number, 30 for instance, the resultant rotation would be 7.5 degrees per second; verses the 60 degrees per second that would be realized with an accel of 2.

Table 18 TURN Attributes

Name	Type	Default	Enumerations
control	int		bbbbb
control_state	int		toggle hold
mode	string		absolute relative
tracker	string		wand head
buffer	float		
accel	float		
AXISLOCK/freeze_yaw	bool	false	true false
AXISLOCK/freeze_roll	bool	false	true false
AXISLOCK/freeze_pitch	bool	false	true false

Look

Look is used to manipulate the lookup vector of the viewing perspective independently from the vector of traversal when using a tracker-based accelerator. That is to say, when traversing a scene through the use of an accelerator, with *DIRMODE* set to *tracker*, the user will move in the direction indicated by the tracker element chosen in the *turn* method. For example: If <TURN tracker =“wand”> and <DIRMODE mode =

“tracker”>, the accelerator will influence the scene’s movement about the user according to the orientation of the wand. If the user desires to continue moving in the direction of the wand, yet rotate the scene about the user independently of the scene traversal the *look* method may be used. This method is most useful for applications displayed on partially enclosed visualization systems. A common example of its use is to fly above a scene while continually adjusting the view such that an object located below remains in the active viewpoint.

There are four elements that define the *look* method. These are *control, control_state, mode,* and *tracker*. Control, as might be expected defines the button used to activate the routine. *Control_state* is comprised of two states: *toggle* and *hold*. This element determines whether the influence of the look method is toggled on or if it is active only when the control buttons are depressed. When set to toggle the method becomes active as soon as the control button is selected. The method will remain active until the control button is selected again. When set to hold, however, the method is only active when the control button is held. *Mode* allows the user to define whether the tracker influence is based upon absolute or relative positioning. Given *mode* is set to *absolute;* the scene will be orientated such that its coordinate system aligns with the coordinate system of the relevant tracker. When *mode* is set to *relative,* the scene will be rotated about the user relative to the change in orientation of the tracker from the time the control button is first selected. The final element of the method is *Tracker*. The *tracker* element gives the user an option of influencing the look method through the *head* or *wand* tracker. When head is selected, the users head position will control the position of the database about the user. If wand is selected, the wand tracker will be used to modify the orientation of the database.

Table 19 LOOK Attributes

Name	Type	Default	Enumerations
control	int		bbbbb
control_state	int		toggle hold
mode	string		absolute relative
tracker	string		wand head

Interaction

In an effort to extend the flexibility of applications designed with XJL, while at the same time maintaining a clean and precise development environment, the deployment of interaction techniques has been included in the XJL schema as a sub-element of DEFINE_NAV. In this way designers may readily associate specific navigation methods to their associated interaction techniques. The paradigm is that as an end user toggles through various navigation systems that may be designed into an application, he or she will also be selecting the interaction systems linked to a given navigator.

The actual configuration of the INTERACT element is quite simple at all levels. The system contains three sub-elements, namely: SLECT, TRANSFORM, and RELEASE. Each element addresses a single *control* attribute used to define the control button linked to the given element. If a designer leaves an interaction element undefined or does not include its control attribute the system will process the parent navigation system without enabling interaction.

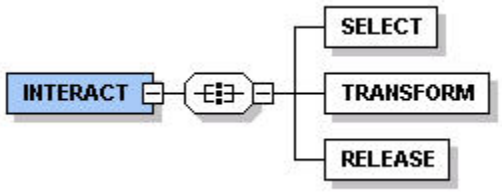

Figure 15 INTERACT Element

Table 20 INTERACT Attributes

Name	Type	Default	Enumerations
SELECT/control	int	0.0	bbbbb
TRANSFORM/control	int	0.0	bbbbb
RELEASE/control	int	0.0	bbbbb

Audio

Through the AUDIO element an application designer may add auditory objects to the scene graph of an application. Such objects may be positioned in a scene according to the "x, y, z" coordinates defined as part of the elements attributes. The designer may also set the auditory intensity or relative volume of these objects. Audio objects are named entities and thus may be manipulated in a similar way to other objects. Using an ANIMATE element, for example, an object may be assigned to traverse the scene according to predefined constraints. A wider array of manipulations may be explored by linking interactions with geometric objects to the programmatic control of an audio object.

Table 21 AUDIO Attributes

Name	Type	Default	Enumerations
audio_data	string		
name	string		
x	float	0.0	
y	float	0.0	
z	float	0.0	
intensity	float	1.0	

Heads Up Display

The HUD element is used to indicate the characteristics that are rendered by the heads-up-display manager. Although this feature is enabled in the GENERAL element, visibility of the heads-up-display may be toggled on and off by its associated control attribute. The display attributes managed by this element are: display*_nav, display_active_obj, display_position,* and *display_orientation* (Table 22). The *display_nav* object provides user feedback as to the navigation device actively being used. In doing this it references the name given to the active navigator through the DEFINE_NAV element. Providing additional information on the navigator, *display_motion* enables the viewing of movement about the scene. *Display_active_obj* displays the name of all active or selected objects, *display_position* provides the users position relative to the global coordinate system, and *display_orientation* provides the users orientation.

Table 22 HUD Attributes

Name	Type	Default	Enumerations
control	int		bbbbb
display_nav	bool	true	true false
display_motion	bool	true	true false
display_active_obj	bool	true	true false
display_position	bool	true	true false
display_orientation	bool	true	true false

Chapter 5: Implementation of XJL

In the previous chapter we explored the XJL specification. Now, we turn our attention to XJ Nav, an implementation of the XJL specification built upon VR Juggler and OpenGL Performer.

The Design of XJ Nav

VR Juggler [Bierbaum00], 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 [Apache01], 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
8       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” [Just01]. 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.

Running an XJ Nav Application

In order to run an XJ Nav application, the following resources must be available:

- Precompiled XJ Nav Binary
- VR Juggler ConfigChunks
- XJL Configuration File

As indicated earlier in this paper the XJL configuration includes an element where the application designer may define the VR Juggler configuration chunks that are to be used. In this way the execution of a VR Juggler application is simplified to a single argument. In the case of XJ Nav the command argument would take the following form:

```
%>  XJNav myxjlconfig.xml
```

Chapter 6: Results and Discussion

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.

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™ [MultiGen01] 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="/home/users/timmyg/.vjconfig/sim.base.config"/>
                <VJ_CONFIG vjconfig="/home/users/timmyg/.vjconfig/sim.wand.mixin.config"/>
                <VJ_CONFIG vjconfig="/home/users/timmyg/.vjconfig/sim.displays.config"/>
                <SET_PATH path="/home/users/timmyg/XJL/data"/>
```

```
        <LOAD_DATABASE
            geometry="/home/users/timmyg/XJL/data/test.flt"
            name="test_db"
            path="/home/users/timmyg/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>
```

Advanced Application Development

Building upon the features explored in the previous example, we will now explore several of the advanced functionalities XJL presents. As discussed in Chapter 4, there are a number of ways a user may customize an applications navigation system. 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>
                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="0.0"
                        yaw="0.0" roll="0.0" pitch="0.0"/>
        </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">
```

Chapter 7: Conclusions and Directions

In designing the XJL specification, and its associated test application, we have developed a solution equipped to empower non-technical digital artists for the task of designing immersive visualization applications. With a focus on component driven extendibility, XJL is 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 (Appendix B: XJL DTD) a graphical interface may be developed that supports the visual – drag and drop / point and click – design of advanced applications. Extending the INTERACT element would also be of value. Through the innovative combination of diverse

elements an application designer is presently able to implement a wide variety of interaction and navigation techniques. There are, however, significant methods that remain inaccessible. Among these are “Go-Go” [Poupyrev96] and “Ray Casting” [Bowman99] locomotion techniques. By adding support for ray cast object selection and extending the DEFINE_NAV/DIRMODE/mode attribute, for the support of object-tracked motion, such methods may be realized. Unlike coordinates, tracker, or tracked-object transformation methods, an object-tracked paradigm would support transformations wherein the directional vector is positioned at the location of the user its interest point is directed at some tracked object. Upon initiating motion, the scene would be traversed along that vector, until the distance between the user and tracked object is nullified. Related techniques could be explored to support “hot spot” lighting, wherein the users wand or a tracked object acts as a spotlight, highlighting or providing otherwise unique rendering treatments to a specific part of the visible scene. Solutions supporting partial transparency and line rendering are examples such rendering treatment.

Appendix A: Sample XJL Document

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XJL file-->
<!DOCTYPE XJL_CONFIG SYSTEM "E:\home\Data\Thesis\XML_SPY\XJL7.dtd">
<XJL_CONFIG xjlconfig="">
        <GENERAL
                hud="false"
                toggle_nav="true"
                stats="false"
                audio="false"
                near="0.2"
                far="20000"
                units="feet">
                <HOME
                        x="0.0" y="0.0" z="0.0"
                        yaw="0.0" roll="0.0" pitch="0.0"/>
                <VJ_CONFIG vjconfig="String"/>
                <SET_PATH path="String"/>
                <LOAD_DATABASE
                        geometry="String"
                        name="String"
                        path="String"
                        x="0.0" y="0.0" z="0.0"
                        yaw="0.0" roll="0.0" pitch="0.0"
                        scale="1.0"/>
        </GENERAL>
        <FIND_NODE
                node="String"
                center_node=”String”
                name="String"
                selectable=”true”>
                <PRE_TRANS x="0.0" y="0.0" z="0.0"/>
                <ROTATE yaw="0.0" pitch="0.0" roll="0.0"/>
                <TRANS x="0.0" y="0.0" z="0.0"/>
                <SCALE x="1.0" y="1.0" z="1.0"/>
                <POST_TRANS x="0.0" y="0.0" z="0.0"/>
                <LOCK
                        x="false" y=" false " z=" false "
                        yaw=" false " pitch=" false " roll=" false "/>
        </FIND_NODE>
        <CREATE_LIGHT
                name="String"
                type="GLOBAL">
                <POSITION x="0.0" y="0.0" z="0.0"/>
                <DIRECTION x="0.0" y="0.0" z="0.0"/>
                <COLOR
                        property="DIFFUSE"
                        r="1.0" g="1.0" b="1.0"/>
                <INTENSITY
                        power="1.0"
```

```
                angle="180">
                <ATTEN
                        constant=”1.0”
                        linear=”0.0”
                        quadratic=”0.0”/>
        </INTENSITY>
</CREATE_LIGHT>
<VISIBILITY
        name="String"
        group=”String”
        operator="less"
        tracker="head"
        object="String"
        view="hide"
        x="1.0" y="1.0" z="1.0"/>
<ANIMATE
        name="String"
        seconds_start="-1"
        seconds_stop="-1"
        method="swing"
        repeat="-1">
        <PRE_TRANSFORM
        x="0.0" y="0.0" z="0.0"
        yaw="0.0" roll="0.0" pitch="0.0"/>
        <LIMIT
                x="0.0" y="0.0" z="0.0"
                yaw="0.0" roll="0.0" pitch="0.0"/>
        <STEP
                x="0.0" y="0.0" z="0.0"
                yaw="0.0" roll="0.0" pitch="0.0"/>
        <CONTROL
                control="10000"
                control_state="toggle"
                follow="false"
                object="String"
                tracker="wand"
                buffer_x="0.0" buffer_y="0.0" buffer_z="0.0"
                random_x="false" random_y=" false " random_z=" false ">
                <DISTANCE x="0.0" y="0.0" z="0.0"/>
        </CONTROL>
</ANIMATE>
<NAV_CONFIG control="10000">
        <DEFINE_NAV
                name="String"
                collision="true"
                gravity="true">
                <ACCEL
                        mode="ramp"
                        control="10000"
                        accel="10"
                        max="50">
                        <DIRMODE
                                mode="tracker"
                                x="0.0" y="0.0" z="0.0"
```

```
                                yaw="0.0" roll="0.0" pitch="0.0"/>
                        </ACCEL>
                        <STOP control="10000"/>
                        <BREAK control="10000" decel="0.15"/>
                        <RESET control="10000"/>
                        <TURN
                                control="10000"
                                control_state="toggle"
                                mode="absolute"
                                tracker="wand"
                                object="String"
                                buffer="5"
                                accel="1.25">
                                <AXISLOCK
                                        freeze_yaw="false"
                                        freeze_roll="false"
                                        freeze_pitch="false"/>
                        </TURN>
                        <LOOK
                                control="10000"
                                control_state="toggle"
                                mode="absolute"
                                tracker="wand"
                                object="String"/>
                        <INTERACT>
                                <SELECT control="10000"/>
                                <TRANSFORM control="10000"/>
                                <RELEASE control="10000"/>
                        </INTERACT>
                </DEFINE_NAV>
        </NAV_CONFIG>
        <AUDIO
                audio_Data="String"
                name="String"
                x="0.0"
                y="0.0"
                z="0.0"
                intensity="1.0"/>
        <HUD
                control="10000"
                display_nav="true"
                display_motion="true"
                display_active_obj="true"
                display_position="true"
                display_orientation="true"/>
</XJL_CONFIG>
```

Appendix B: XJL DTD

In this section we provide the XJL DTD. A DTD is the "Document Type Definition" or formal specification of a markup language. Similar to XML, DTD content is written using SGML, the "Standard Generalized Markup Language" – a standard for describing markup languages. Using a DTD, the formal document structure of a specification may be maintained. In this way related XML content may be validated against the DTD for conformance.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- XJL DTD-->
<!ELEMENT XJL_CONFIG (GENERAL, FIND_NODE*, CREATE_LIGHT*, VISIBILITY*, ANIMATE*,
NAV_CONFIG?, AUDIO?, HUD?)>
<!ATTLIST XJL_CONFIG
        xjlconfig CDATA #IMPLIED
>
<!ELEMENT GENERAL (HOME, VJ_CONFIG+, SET_PATH+, LOAD_DATABASE+)>
<!ATTLIST GENERAL
        hud (true | false) #IMPLIED
        toggle_nav (true | false) #IMPLIED
        stats (true | false) #IMPLIED
        audio (true | false) #IMPLIED
        near CDATA #IMPLIED
        far CDATA #IMPLIED
        units (feet | meters | inches) #IMPLIED
>
<!ELEMENT FIND_NODE (PRE_TRANS?, ROTATE?, TRANS?, SCALE?, POST_TRANS?, LOCK?)>
<!ATTLIST FIND_NODE
        node CDATA #REQUIRED
        center_node CDATA #IMPLIED
        name CDATA #REQUIRED
        selectable (true) #IMPLIED
>
<!ELEMENT CREATE_LIGHT (POSITION?, DIRECTION?, COLOR*, INTENSITY?)>
<!ATTLIST CREATE_LIGHT
        name CDATA #IMPLIED
        type (SPOTLIGHT | GLOBAL) #IMPLIED
>
<!ELEMENT VISIBILITY EMPTY>
<!ATTLIST VISIBILITY
        name CDATA #REQUIRED
        group CDATA #IMPLIED
        operator (unselected | selected | less | greater) #REQUIRED
        tracker (head | wand | object) #IMPLIED
        object CDATA #IMPLIED
```

```
        view (hide | show) #REQUIRED
        x CDATA #IMPLIED
        y CDATA #IMPLIED
        z CDATA #IMPLIED
>
<!ELEMENT ANIMATE (PRE_TRANSFORM?, LIMIT?, STEP?, CONTROL)>
<!ATTLIST ANIMATE
        name CDATA #REQUIRED
        seconds_start CDATA #IMPLIED
        seconds_stop CDATA #IMPLIED
        method (swing | loop | static) #IMPLIED
        repeat CDATA #IMPLIED
>
<!ELEMENT NAV_CONFIG (DEFINE_NAV*)>
<!ATTLIST NAV_CONFIG
        control (10000 | 01000 | 00100 | 00010 | 00001 | 11000 | 10100 | 10010 | 10001 | 01100 | 01010 | 01001
| 00110 | 00101 | 00011 | 11100 | 11010 | 11001 | 10110 | 10101 | 10011 | 01110 | 01010 | 00111 | 11110 | 11101 |
11011 | 10111 | 01111 | 11111) #REQUIRED
>
<!ELEMENT AUDIO EMPTY>
<!ATTLIST AUDIO
        audio_Data CDATA #REQUIRED
        name CDATA #REQUIRED
        x CDATA #IMPLIED
        y CDATA #REQUIRED
        z CDATA #IMPLIED
        intensity CDATA #IMPLIED
>
<!ELEMENT HUD EMPTY>
<!ATTLIST HUD
        control (10000 | 01000 | 00100 | 00010 | 00001 | 11000 | 10100 | 10010 | 10001 | 01100 | 01010 | 01001
| 00110 | 00101 | 00011 | 11100 | 11010 | 11001 | 10110 | 10101 | 10011 | 01110 | 01010 | 00111 | 11110 | 11101 |
11011 | 10111 | 01111 | 11111) #REQUIRED
        display_nav (true | false) #IMPLIED
        display_motion (true | false) #IMPLIED
        display_active_obj (true | false) #IMPLIED
        display_position (true | false) #IMPLIED
        display_orientation (true | false) #IMPLIED
>
<!ELEMENT HOME EMPTY>
<!ATTLIST HOME
        x CDATA #IMPLIED
        y CDATA #IMPLIED
        z CDATA #IMPLIED
        yaw CDATA #IMPLIED
        roll CDATA #IMPLIED
        pitch CDATA #IMPLIED
>
<!ELEMENT VJ_CONFIG EMPTY>
<!ATTLIST VJ_CONFIG
        vjconfig CDATA #REQUIRED
>
<!ELEMENT SET_PATH EMPTY>
<!ATTLIST SET_PATH
```

```
        path CDATA #IMPLIED
>
<!ELEMENT LOAD_DATABASE EMPTY>
<!ATTLIST LOAD_DATABASE
        geometry CDATA #REQUIRED
        name CDATA #REQUIRED
        path CDATA #IMPLIED
        x CDATA #IMPLIED
        y CDATA #IMPLIED
        z CDATA #IMPLIED
        yaw CDATA #IMPLIED
        roll CDATA #IMPLIED
        pitch CDATA #IMPLIED
        scale CDATA #IMPLIED
>
<!ELEMENT PRE_TRANS EMPTY>
<!ATTLIST PRE_TRANS
        x CDATA #IMPLIED
        y CDATA #IMPLIED
        z CDATA #IMPLIED
>
<!ELEMENT ROTATE EMPTY>
<!ATTLIST ROTATE
        yaw CDATA #IMPLIED
        pitch CDATA #IMPLIED
        roll CDATA #IMPLIED
>
<!ELEMENT TRANS EMPTY>
<!ATTLIST TRANS
        x CDATA #IMPLIED
        y CDATA #IMPLIED
        z CDATA #IMPLIED
>
<!ELEMENT SCALE EMPTY>
<!ATTLIST SCALE
        x CDATA #IMPLIED
        y CDATA #IMPLIED
        z CDATA #IMPLIED
>
<!ELEMENT POST_TRANS EMPTY>
<!ATTLIST POST_TRANS
        x CDATA #IMPLIED
        y CDATA #IMPLIED
        z CDATA #IMPLIED
>
<!ELEMENT LOCK EMPTY>
<!ATTLIST LOCK
        x (true | false) #IMPLIED
        y (true | false) #IMPLIED
        z (true | false) #IMPLIED
        yaw (true | false) #IMPLIED
        roll (true | false) #IMPLIED
        pitch (true | false) #IMPLIED
>
```

```
<!ELEMENT POSITION EMPTY>
<!ATTLIST POSITION
        x CDATA #IMPLIED
        y CDATA #IMPLIED
        z CDATA #IMPLIED
>
<!ELEMENT DIRECTION EMPTY>
<!ATTLIST DIRECTION
        x CDATA #IMPLIED
        y CDATA #IMPLIED
        z CDATA #IMPLIED
>
<!ELEMENT COLOR EMPTY>
<!ATTLIST COLOR
        property (DIFFUSE | AMBIENT | SPECULAR) #REQUIRED
        r CDATA #IMPLIED
        g CDATA #IMPLIED
        b CDATA #IMPLIED
>
<!ELEMENT INTENSITY (ATTEN?)>
<!ATTLIST INTENSITY
        focus CDATA #IMPLIED
        angle CDATA #IMPLIED
>
<!ELEMENT PRE_TRANSFORM EMPTY>
<!ATTLIST PRE_TRANSFORM
        x CDATA #IMPLIED
        y CDATA #IMPLIED
        z CDATA #IMPLIED
        yaw CDATA #IMPLIED
        roll CDATA #IMPLIED
        pitch CDATA #IMPLIED
>
<!ELEMENT LIMIT EMPTY>
<!ATTLIST LIMIT
        x CDATA #IMPLIED
        y CDATA #IMPLIED
        z CDATA #IMPLIED
        yaw CDATA #IMPLIED
        roll CDATA #IMPLIED
        pitch CDATA #IMPLIED
>
<!ELEMENT STEP EMPTY>
<!ATTLIST STEP
        x CDATA #IMPLIED
        y CDATA #IMPLIED
        z CDATA #IMPLIED
        yaw CDATA #IMPLIED
        roll CDATA #IMPLIED
        pitch CDATA #IMPLIED
>
<!ELEMENT CONTROL (DISTANCE?)>
<!ATTLIST CONTROL
```

```
        control (10000 | 01000 | 00100 | 00010 | 00001 | 11000 | 10100 | 10010 | 10001 | 01100 | 01010 | 01001
| 00110 | 00101 | 00011 | 11100 | 11010 | 11001 | 10110 | 10101 | 10011 | 01110 | 01010 | 00111 | 11110 | 11101 |
11011 | 10111 | 01111 | 11111 | tracker) #IMPLIED
        control_state (toggle | hold) #IMPLIED
        follow (true | false) #IMPLIED
        object CDATA #IMPLIED
        tracker (wand | head | object) #IMPLIED
        buffer_x CDATA #IMPLIED
        buffer_y CDATA #IMPLIED
        buffer_z CDATA #IMPLIED
        random_x (true | false) #IMPLIED
        random_y (true | false) #IMPLIED
        random_z (true | false) #IMPLIED
        operator (unselected | selected | less | greater) #IMPLIED
>
<!ELEMENT DEFINE_NAV (ACCEL*, STOP?, BREAK?, RESET?, TURN?, LOOK?, INTERACT?)>
<!ATTLIST DEFINE_NAV
        name CDATA #REQUIRED
        collision (true | false) #IMPLIED
        gravity (true | false) #IMPLIED
>
<!ELEMENT ATTEN EMPTY>
<!ATTLIST ATTEN
        constant CDATA #IMPLIED
        linear CDATA #IMPLIED
        quadratic CDATA #IMPLIED
>
<!ELEMENT DISTANCE EMPTY>
<!ATTLIST DISTANCE
        x CDATA #IMPLIED
        y CDATA #IMPLIED
        z CDATA #IMPLIED
>
<!ELEMENT ACCEL (DIRMODE)>
<!ATTLIST ACCEL
        mode (ramp | hold) #IMPLIED
        control (10000 | 01000 | 00100 | 00010 | 00001 | 11000 | 10100 | 10010 | 10001 | 01100 | 01010 | 01001
| 00110 | 00101 | 00011 | 11100 | 11010 | 11001 | 10110 | 10101 | 10011 | 01110 | 01010 | 00111 | 11110 | 11101 |
11011 | 10111 | 01111 | 11111) #REQUIRED
        accel CDATA #IMPLIED
        max CDATA #IMPLIED
>
<!ELEMENT STOP EMPTY>
<!ATTLIST STOP
        control (10000 | 01000 | 00100 | 00010 | 00001 | 11000 | 10100 | 10010 | 10001 | 01100 | 01010 | 01001
| 00110 | 00101 | 00011 | 11100 | 11010 | 11001 | 10110 | 10101 | 10011 | 01110 | 01010 | 00111 | 11110 | 11101 |
11011 | 10111 | 01111 | 11111) #REQUIRED
>
<!ELEMENT BREAK EMPTY>
<!ATTLIST BREAK
        control (10000 | 01000 | 00100 | 00010 | 00001 | 11000 | 10100 | 10010 | 10001 | 01100 | 01010 | 01001
| 00110 | 00101 | 00011 | 11100 | 11010 | 11001 | 10110 | 10101 | 10011 | 01110 | 01010 | 00111 | 11110 | 11101 |
11011 | 10111 | 01111 | 11111) #REQUIRED
        decel CDATA #IMPLIED
```

```
>
<!ELEMENT RESET EMPTY>
<!ATTLIST RESET
	control (10000 | 01000 | 00100 | 00010 | 00001 | 11000 | 10100 | 10010 | 10001 | 01100 | 01010 | 01001
| 00110 | 00101 | 00011 | 11100 | 11010 | 11001 | 10110 | 10101 | 10011 | 01110 | 01010 | 00111 | 11110 | 11101 |
11011 | 10111 | 01111 | 11111) #REQUIRED
>
<!ELEMENT TURN (AXISLOCK)>
<!ATTLIST TURN
	control (10000 | 01000 | 00100 | 00010 | 00001 | 11000 | 10100 | 10010 | 10001 | 01100 | 01010 | 01001
| 00110 | 00101 | 00011 | 11100 | 11010 | 11001 | 10110 | 10101 | 10011 | 01110 | 01010 | 00111 | 11110 | 11101 |
11011 | 10111 | 01111 | 11111) #REQUIRED
	control_state (toggle | hold) #IMPLIED
	mode (absolute | relative) #IMPLIED
	tracker (wand | head | object) #IMPLIED
	object CDATA #REQUIRED
	buffer CDATA #IMPLIED
	accel CDATA #IMPLIED
>
<!ELEMENT LOOK EMPTY>
<!ATTLIST LOOK
	control (10000 | 01000 | 00100 | 00010 | 00001 | 11000 | 10100 | 10010 | 10001 | 01100 | 01010 | 01001
| 00110 | 00101 | 00011 | 11100 | 11010 | 11001 | 10110 | 10101 | 10011 | 01110 | 01010 | 00111 | 11110 | 11101 |
11011 | 10111 | 01111 | 11111) #REQUIRED
	control_state (toggle | hold) #IMPLIED
	mode (absolute | relative) #IMPLIED
	tracker (wand | head | object) #IMPLIED
	object CDATA #IMPLIED
>
<!ELEMENT INTERACT (SELECT, TRANSFORM, RELEASE)>
<!ELEMENT DIRMODE EMPTY>
<!ATTLIST DIRMODE
	mode (tracker | coord) #IMPLIED
	x CDATA #IMPLIED
	y CDATA #IMPLIED
	z CDATA #IMPLIED
	yaw CDATA #IMPLIED
	roll CDATA #IMPLIED
	pitch CDATA #IMPLIED
>
<!ELEMENT AXISLOCK EMPTY>
<!ATTLIST AXISLOCK
	freeze_yaw (true | false) #IMPLIED
	freeze_roll (true | false) #IMPLIED
	freeze_pitch (true | false) #IMPLIED
>
<!ELEMENT SELECT EMPTY>
<!ATTLIST SELECT
	control (10000 | 01000 | 00100 | 00010 | 00001 | 11000 | 10100 | 10010 | 10001 | 01100 | 01010 | 01001
| 00110 | 00101 | 00011 | 11100 | 11010 | 11001 | 10110 | 10101 | 10011 | 01110 | 01010 | 00111 | 11110 | 11101 |
11011 | 10111 | 01111 | 11111) #REQUIRED
>
<!ELEMENT TRANSFORM EMPTY>
<!ATTLIST TRANSFORM
```

```
        control (10000 | 01000 | 00100 | 00010 | 00001 | 11000 | 10100 | 10010 | 10001 | 01100 | 01010 | 01001
| 00110 | 00101 | 00011 | 11100 | 11010 | 11001 | 10110 | 10101 | 10011 | 01110 | 01010 | 00111 | 11110 | 11101 |
11011 | 10111 | 01111 | 11111) #REQUIRED
>
<!ELEMENT RELEASE EMPTY>
<!ATTLIST RELEASE
        control (10000 | 01000 | 00100 | 00010 | 00001 | 11000 | 10100 | 10010 | 10001 | 01100 | 01010 | 01001
| 00110 | 00101 | 00011 | 11100 | 11010 | 11001 | 10110 | 10101 | 10011 | 01110 | 01010 | 00111 | 11110 | 11101 |
11011 | 10111 | 01111 | 11111) #REQUIRED
>
```

References Cited

[Alice01] "About Alice", http://www.alice.org, Visited 10-01-01.

[Apache01] "Xerces C++ Parser", http://xml.apache.org/xerces-c , Visited 11-06-01.

[Arns01] Arns, L., "A new taxonomy for locomotion in virtual environments", Ph.D. Thesis, Iowa State University, 2001.

[Ascension01] "MotionStar Wireless", http://www.ascension-tech.com/products/mswireless, Visited 11-06-01.

[Barnsley72] Barnsley, M., "Fractals Everywhere", Pan Books: London, 1972.

[Bierbaum98] Bierbaum and Just, "*Software Tools for Virtual Reality Application Development*", ACM SIGGRAPH 98 Course Notes.

[Bierbaum00] Bierbaum, A., "*VR Juggler: A Virtual Platform for Virtual Reality Application Development*", MS Thesis, Iowa State University, 2000.

[Bowman98] Bowman, Koller, and Hodges. "A Methodology for the Evaluation of Travel Techniques for Immersive Virtual Environments", Virtual Reality: Research, Development, and Applications, vol 3, no 2, 1998, pp.120-131.

[Bowman99] Bowman and Hodges. "Formalizing the Design, Evaluation, and Application of Interaction Techniques for Immersive Virtual Environments", Journal of Visual Languages and Computing. 1999, vol 10, no 1, pp. 37-53.

[Boyd95] Boyd, C., "Human and Machine Dimensions of 3D Interfaces for Virtual Environments", ACM CHI '95 Doctoral Consortium, 1995, pp 41-42.

[Brutzman01] Brutzman, D., "X3D Content Examples, Editing, Conformance Suite, and Software Development Kit (SDK) CDs", SIGGRAPH 2001 Conference Abstracts and Applications, p 280.

[Chance98] Chance, Gaunet, Beall, and Loomis, "Locomotion Mode Affects the Updating of Objects Encountered During Travel: The Contribution of Vestibular and Proprioceptive Inputs to Path Integration", Presence, vol 7, no2, 1998, pp 168-178.

[Conway00] Conway, Audia, Burnette, Cosgrove, and others, "Alice: Lessons Learned from Building a 3D System for Novices", ACM CHI 2000 Papers, pp 486-493.

[Cooper00] Cooper, A., “The Inmates are Running the Asylum”, SAMS: Indianapolis, Indianapolis, Indiana, 2000.

[Cruz95a] 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.

[Cruz95b] Cruz-Neira, C. “Projection-based Virtual Reality: The CAVE and its Applications to Computational Science”, Ph.D. Thesis, University of Illinois at Chicago, 1995.

[Dachselt01] Dachselt, R., “CONTIGRA: A High-Level XML-Based Approach to Interactive 3D Components”, SIGGRAPH 2001 Conference Abstracts and Applications, p 163

[Dai97] Dai, Eckel, Göbel, Hasenbrink, and others, “Virtual Spaces: VR Projection System Technologies and Applications”, Tutorial Notes, Eurographics 97, Budapest 1997.

[Darken96] Darken, and Sibert. “Navigating Large Virtual Spaces.” International Journal of Human-Computer Interaction, Jan-Mar 1996, vol 8, no 1, 1996, pp.49-72.

[Eckel97] Eckel, Jones, and Domeier, “OpenGL Performer™ Getting Started Guide”, Silicon Graphics, Inc., 1997.

[Elliot94] Elliott, Schechter, Yeung, and Abi-Ezzi, “TBAG, A High Level Framework for Interactive, Animated 3D Graphics Applications”, SIGGRAPH 94 Conference Proceedings, 1994.

[Fakespace01] “The CubicMouse™”, http://www.fakespacesystems.com, Visited 11-06-01.

[Figueroa01] Figueroa, Green, and Hoover, “3dml: A Language for 3D Interaction Techniques Specification”, EuroGraphics 2001 Short Presentations, 2001.

[Hewett96] Hewett, Baecker, Card, Carey, and others, “ACM SIGCHI Curricula for Human-Computer Interaction”, http://sigchi.org/cdg/cdg2.html, Visited 09-25-01, 1996.

[Holtzman94] Holtzman, S., “Digital mantras: the languages of abstract and virtual worlds”, MIT Press: Cambridge, Massachusetts, 1994.

[Interlink01] ACM SIGCHI Curricula for Human-Computer Interaction", http://www.interlinkelec.com, Visited 11-06-01.

[Just01] 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.

[Keep93] Keep, C., "Knocking of Heaven's Door: Leibniz, Baudrillard and Virtual Reality", EJournal, Volume 3 Number 2., September 1993.

[Landauer97] Landauer, Blach, Bues, Rösch, and Simon, "Toward Next Generation Virtual Reality Systems", Proceedings IEEE International Conference on Multimedia Computing and Systems, Ottawa, 1997.

[Loomis93] Loomis, Klatzky, Golledge, Cicinelli, Pellegrino, and Fry. "Nonvisual Navigation by Blind and Sighted*?: Assessment of Path Integration Ability." Journal of Experimental Psychology: General, vol 122, no 1, 1993, pp. 73-91.

[Merriam93] "Merriam Webster's Collegiate Dictionary", Merriam-Webster, Inc.: Massachusetts, 1993.

[Moody99] 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.

[MultiGen01] Dowgiallo, T., "Introduction to OpenFlight APIs", http://www.multigen.com/products/openflight/intro_api.shtml, Visited 10-06-01, 1997.

[Najork94] Najork, M., "Obliq-3D Tutorial and Reference Manual", SRC Research Report 129, December 1994.

[Rohlf00] 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.

[Péruch97] Péruch, May, and Wartenberg, "Homing in Virtual Environments: Effects of filed of view and path layout", Perception, vol 26, 1997, pp 301-311.

[Pimentel95] Pimentel and Teixeira, "Virtual Reality: Through the New Looking Glass", McGraw-Hill, Inc.: New York, 1995.

[Poupyrev96] Poupyrev, Billinghurst, Weghorst, and Ichikawa, "The Go-Go Glass Interaction Technique: Non-linear Mapping for Direct Manipulation in VR", ACM UIST '96, Seattle Washington, pp. 79-80.

[Rheingold92] Rheingold, H., "Virtual Reality", Simon & Schuster: New York, 1992.

[Sense8a] "WorldToolKit Release9: Technical Overview", http://www.sense8.com, Visited 10-01-01.

[Sense8b] "World Up User's Manual", http://www.sense8.com, Visited 10-01-01.

[Shaw93] Shaw and Green, "The MR Toolkit Peers Package and Experiment", IEEE Virtual Reality Annual International Symposium (VRAIS 1993), pp 463-469.

[Stoakley95] Stoakley, Conway, and Pausch, "Virtual Reality on a WIM: Interactive Worlds in Miniature", Proceedings of ACM CHI 1995, pp. 265-272.

[Stuart01] Stuart, R., "The Design of Virtual Environments", Barricade Books, Inc.: New Jersey, 2001.

[Suhayda00] Suhayda, G., "Computer and Video Games: The Next Generation", ACM SIGGRAPH 2000 Conference Abstracts and Applications, ACM Press: New York, 2000, p 110.

[Vega01] "Vega: The Comprehensive Software Environment for Realtime Application Development", http://www.multigen.com/support/dc_files/Vega_brochure.pdf, Visited 10-01-01.

[W3C01] "XML Schema", http://www.w3.org/XML/Schema, Visited 10-01-01.

[Wang95] Wang and Green, "EM – An Environment Manager for Building Networked Virtual Environments", IEEE Virtual Reality Annual International Symposium, 1995.

[X3D01] "X3D – Extensible 3D", http://www.web3d.org, Visited 10-01-01.

[XGL01] "XGL File Format Specification", http://www.xglspec.org, Visited 10-01-01.