Skript: An extendable language flexible avatar control framework for VR applications

by

Alan Stanley Fischer

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Carolina Cruz-Neira, Major Professor
Julie Dickerson
Whitney Sanford

Iowa State University

Ames, Iowa

2005

Graduate College
Iowa State University

This is to certify that the master's thesis of

Alan Stanley Fischer

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

Table of Contents

List of Figures

Abstract

This thesis covers the design and implementation of a framework for the integration of avatars into Virtual Reality (VR) applications. Specifically, the work focuses on a language flexible method of scripting avatars to perform a series of actions. This framework can be integrated with other packages that allow the avatars to be controlled by influences beyond just the script. These other packages can be crowd simulation algorithms, or some other behavior modeling algorithms that simulate certain situations. More importantly, these packages can also enable the avatars to be aware of the VR application user, and therefore can enhance the user experience. The use of this framework simplifies the development of a user aware avatar application. Finally, this framework was integrated with an existing VR application; a reconstruction of a Hindu temple. The results of this integration and the insight gained from it will be examined.

Chapter 1. Introduction

In the past few years, VR has evolved to become its own mature field. Research efforts have broadened from the initial technical challenges for the basic technologies to a much wider range of application domains. Whatever the application domain is, a key point of VR is to immerse the user in the virtual space and provide the illusion of being present in that space. The illusion of being present is when the application user feels like they are "in" the application [Bierbaum00a]. This illusion can be provided at many levels, from the detail of graphics, such as higher triangle counts and faster rendering times for a realistic look of the virtual world, to the interactive capabilities to enable the user explore and manipulate the space and its elements.

An interesting aspect of creating the illusion of presence in VR is the inclusion of virtual humans, also called avatars. In the real world, we are used to see people walking on the streets, standing up in the cashier line at a supermarket, and so on. Until recently, many virtual environments were very “lonely”, with the user being the only inhabitant of that space. Adding avatars to the virtual work has the capability to enhance the VR experience, by letting the user feel like there are other individuals in the virtual world with them. To maximize the effectiveness of these avatars, they should interact with the user in a genuine manner. This involves, among many aspects, natural looking animations for the avatars to perform, and that these animations be strung together in a believable fashion.

There are different methods of instructing these avatars to string together their animations depending upon different factors, such as user input or the actions of other avatars. These instruction methods can vary anywhere from an avatar scripting language [Yang03] to a multiagent interaction crowd simulation engine [Szarowicz01]. There are

benefits to both, but typically the more complicated methods are much more daunting to implement and use effectively; however they do give the application developer a broader range of possibilities.

A developer who is instructing the avatars should be able to instruct a group of avatars without have to learn a specific language. The learning of a specialized scripting language simply for the sake of instructing the avatars can add unnecessary time to the development of the application. This is why a developer should have the flexability of writing commands in whatever scripting language they are most comfortable with.

The implementer of the scripting solution should then have the flexibility to add in more complex influences into the system, such as a crowd behavior system, and have these influence the already written script, without any modifications to these scripts.

The goal of this research is to develop a method of scripting avatars that addresses these shortcomings. It lets a script writer instruct avatars to perform some sequence of actions, and have the script be in a scripting language of the writer's choice. It also is expandable, so that external influences may be taken into account by the avatars running the script. This allows a developer to write an application that uses avatars with the ease of use of a scripting language, but with the added interaction of a more complex artificial intelligence system, if the developer desires.

The end result of this thesis is to give the reader the understanding necessary to construct a robust, but easy to use scripting solution to aid the illusion of reality in a virtual world.

Scope of Thesis

To achieve these goals for an easy to use and extendable scripting system, this thesis targets the development of an extendable solution to control the animation of avatars and create a plugin for this scripting system to integrate it with a crowd simulation model that includes the user as part of the crowd. This thesis focuses on the integration of external influences into the scripting model, and also targets to simulate a situation in which the movement of the avatars is primarily on a two dimentional plane.

Another major goal of this work is that the resulting library, as well as all of its dependencies, be only open source software. We believe that there is quite a lot of work to follow the research presented in this thesis, therefore open source is an excellent strategy to enable future developers and users to continue expanding the framework developed here, and to customize it to their needs. Open source software also provides many advantages to developers, beyond the origional developer using it. If there is some aspect of the software that the developer does not like, the developer is free and able to modify the software to suit his or her needs. Closed source software does not allow this flexability. Because of these reasons, we decided that several of the avatar animation packages that are examined in this document do not suit the needs of this thesis; they are commercial, closed source software focusing on specific aspects of controlling avatars.

Also, the work developed should have the capability of running on all of the primary computing platforms, such as Microsoft Windows, SGI's Irix, Apple's OS-X, and the open source Linux.

Outside the scope of work of this thesis is handling facial animations, such as having avatars express moods, and dealing with avatar emotional behaviors. This does not

mean the capability could not be added into the framework, just that it was not a target of this thesis.

Finally, to provide the information necessary to the reader to develop this scripting system, this theis has been structured as follows:

Chapter one: An introduction to the thesis and a brief description of the goals, followed by a description of the contents of the thesis.

Chapter two: A more in depth background to the thesis, including a short description of avatars in general, and how they are created. Then a description of scripting languages, and how they apply to avatar scripting follows. This is followed by a description of what we are looking for in an avatar integration package that will suit our needs.

Chapter three: An in depth look at several of the current avatar integration packages, their advantages, and their disadvantages.

Chapter four: First, a general discussion about the design of the scripting solution, and an outline of its components. This is followed by a detailed description of the implementation details to the scripting solution. This also includes the development of a simple crowd simulation module that can be used with the scripting solution.

Chapter five: The scripting solution is implemented with an application, the Virtual Hindu Temple, and the results are examined.

Chapter six: A summary of the goals and the conclusion.

Chapter seven: Several possible areas of this thesis that can be expanded upon for future work.

Chapter eight: References.

Chapter 2. Background

Avatars

The word *avatar* comes from the Hindu term for the incarnation of some diety in human or animal form. However, in today's internet world of three dimentional chat rooms and multiplayer games, an avatar has come to refer to the representation of some human character in a virtual world. [Merriam05]

One of the earlier individuals to use the term avatar to refer to the virtual representation of a human character in some virtual world was Neal Stephenson, in his cyberpunk novel Snow Crash [Stephenson93]. In this novel there exists a *metaverse*. This metaverse is a completely virtual world, where human individuals can enter it through terminals, or different connecting points that exist throughout our real world. Once inside the metaverse, they are represented by some avatar, and can interact with other avatars. The other avatars in the metaverse can be representing other humans, or can be controlled completely by a computer, meaning there is no human attached to the avatar. For the human avatars, the visual quality of the avatar varies depending on the quality of connection that the human user has to the metaverse. [Allbeck98]

Avatars with a human counterpart, and *completely virtual avatars* are both used very often in many areas today, especially online gaming. Massivly Multiplayer Online Games (MMOG), where hundreds of users exist in the same online world, are a popular example of avatar use similar to Stephenson's metaverse in many ways. The human users of such MMOGs typically group together to confront different monsters, and these monsters are usually represented by completely virtual avatars.

Figure 1: Teravon, a multiplayer online game.

When it comes to avatar use in VR, avatars can be representations of actual humans, such as in a networked virtual environment where several users are in the same virtual space, but located in separate physical spaces. In this situation, avatars can be used in each users environment to represent the location and actions of the other networked users. This use of avatars can help a group of users connect by a network meet, collaborate, and work together on some project. [Hagsand96]

Completely virtual avatars can also serve many uses in VR applications. They can be used in combination with collision avoidance and crowd software to simulate large croud situations, and to explore how individuals react and move when exposed to different stimuli. This can be valuable in the studying of how a group of individuals react to a fire or other such emergency in an enclosed space such as a museum. [Muuse97]

The avatars in this thesis concentrate around the completely virtual avatars to aid in populating virtual worlds. Since a goal of this thesis is to control the animations and the movements of the avatars with a written script, it would be conflicting to have the animations and movements of the avatars tied to a human counterpart. Instead, the avatar control software developed has complete control over the actions that the avatars perform.

Avatar Creation

Before an application developer can integrate avatars in to their application, the avatar must first be created. This is true of any application that uses avatars, anywhere from completely virtual avatars like the ones used in this thesis, to the avatars in games, movies, or online chat rooms.

This creation process consists of three parts: [Allen03]

- *Modeling*: This is the process of actually creating the geometry that makes up the avatar, and texturing the avatar with an image that represents with skin or clothes.
- *Skeletal rigging*: When a bone structure is attached to the geometry of an avatar, so bones of the skeleton can be moved and the geometry deforms with it correctly.
- *Animation*: When the avatar creator makes numerous different animations for the avatars to perform, by positioning the skelton made in the previous step in a series of *key frames*, or poses.

This avatar creation can be done in many commercial packages like 3D Studio Max [Discreet05] and Maya [Alias04a], or other freely available packages. If desired, several packages can even be used together for different parts of the avatar development cycle.

Avatar Integration

Once the avatar has been created, a software tool is needed to handle the integration of the avatar within an interactive virtual world. This software tool, or avatar integration package, should be able to take the available avatar models, textures, and animations and render a "displayable" avatar embedded in the VR application. However, there is more to integrating an avatar into VR than just rendering it. The process of avatar integration into a VR application is inherently fairly complex, because there are much stricter requirements for avatar behavior when dealing with a VR application as compared with games or movies. When completely virtual avatars are integrated in games or movies, the movement of the user can be constrained to view the virtual world from specific locations and distances. For example, if the game designer does not want to worry about the user getting too close to an avatar, or trying to walk through the avatar, the designer can simply restrict the user's movement to always stay at a specified distance. It is even more so in a movie, since the director has total control over what the movie viewer sees. However in an immersive VR application, the user has full control over the environment and how they want to move through it. Any avatars in a VR application must be able to account for any movement that the user makes. If the user decides to walk into the avatar, the application developer can not simply keep the user from walking that direction. Instead, the developer

needs to provide some strategy for either the avatar to get out of the user's way, or, at the minimum, make the avatar "solid" so the user does not walk through it.

Furthermore, the tools used to develop the graphical components of VR applications are typically low-level APIs, such as scene graphs like OpenSG [Reiners02], OpenGL Performer[SGI05], or the Open Scene Graph[OSG05]. Most avatar integration packages available are usually not designed to be integrated with such tools.

The avatar control package that will be developed in this thesis needs to handle the rendering of the avatars, but more importantly, it also needs to let the application developer control the movement and actions of the avatars using an easy to utilize, but flexible method. Usually, the avatar integration software has some method of letting the user describe the movements the avatars should take, say perhaps a scripting language of some sort.

Scripting Languages

The book *I-Series Computing Concepts* [Haag02] defines a scripting language as an interpreted programming language that works within another application to perform tasks. Some examples of scripting languages are JavaScript, VBScript, or Python. Though some languages like Python lay on the border between being a scripting language and a regular programming language.

Like any programming language, it takes time for a developer to learn a scripting language that they are not familiar with. Typically, the time it takes to learn a scripting language is less than that of a regular programming language, however it still takes valuable time.

Ideally, a developer should be able to use whatever scripting language they wanted to control the movement and actions of the avatars; whether it be JavaScript, Python, Ruby, or some other in house language. If a developer can use the scripting language of their choice, scripting the avatars, as well as the integration of the scripting language with their application, will become easier and the application will have a shorter development time.

Avatar Scripting

As was stated earlier, controlling the movement of an avatar in VR is much more complicated that it is with avatars for mediums such as games or movies. If a script writer instructs an avatar to move in a straight line from point A to point B, and the application user happens to be in between these points, the avatar will walk right through the user. This will negativly affect the illusion of reality for the VR environment. Instead, it would be more ideal if the avatar accomidates the position of the user, and instead of walking through the user, the avatar would alter its path to avoid the user, if at all possible. This problem can also manifest itself when the avatar is stationary. If the user decides to walk into the avatar, the avatar must then try to avoid colliding with the user and should make every effort to move, so that the avatar is not in the same space as the user.

All of these problems become even worse when you have multiple users in the same VR environment. There are then multiple users for the avatars to attempt to avoid, and the avatars must also attempt to avoid each other, since their movement paths may get modified depending upon the position of the users.

Additionally, a scene may have more objects or events influencing the motions of the avatars than just the existance of one or more application users. Some applications allow

external input from other devices or the network to influence the geometry of the scene on the fly. Events of this nature will affect the result of the processing of a simple script to control avatar movements and actions.

Finally, simple scripting can be insufficient even when there is no unpredictable objects, users, or events at all. If a developer is creating a scene that will be completely autonomous, they will still encouter issues with the complexity of having multiple avatars moving about the scene and not colliding with each other, without painstakingly tweaking the script to ensure no avatars attempt to occupy the same space. In this sort of application, it would be quite benificial if the underlying avatar control system could sort out these conflicts internally.

It quickly becomes apparent that for even a modest VR application with multiple moving avatars, a simple scripting language will no longer suffice. There are too many unknowns in the scene to attempt to create a script that will fit all the situations. It is, however, still desirable for an application developer to have the ability to instruct the avatars in the scene with the simplicity of a scripting language. Ideally, a developer should have the ease of a scripting language, combined with the automation of a more complex system behind the scripting language that will ensure correct scenes.

What is Needed

What we are looking for is an avatar control package that is first of all, open source. The freedom given by open source software allows a developer to modify the software in whatever manner is required for their application.

Secondly, the control package should have the freedom of letting a developer

choose the scripting language of his choice to script the avatars. This lets a developer be as productive as possible when developing an application, since they do not need to learn a specific language just for scripting the avatars.

The avatar package should also be more than just a scripting package. We are looking for it to handle the advanced controling of the avatars necessary to avoid collision between each other and the environment. The package should also handle the adjusting of the animations that the avatars are playing, so the movements generated by changes in the scene look natural to a user.

Finally, the package should not be limited to providing avoidance of other avatars. It should also allow a developer to replace or expand on the events that influence the avatars in an extendable manner. If a developer needs another source of influence integrated into the application, they should be able to write a minimal amount of code and easily extend the package to suit their needs.

The following chapter will examine both open source and commercial avatar integration libraries and packages that are available to help develop an interactive VR application with avatars, and compare them against these needs that we have defined.

Chapter 3. Previous Work

There are numerous avatar packages available to help a developer integrate avatars into an application. In this section several of these packages will be examined, along with how closely they suit the needs that were outlined above.

Jack

The first package examined is Jack, which was originally developed by Cary Phillips and Norm Badler for use in virtual human prototyping. [Phillips88] It is now developed and marketed by UGS for manufacturing engineering management solutions. [UGS05a]

Jack provides a developer with a method to put Jack avatars and imported geometry into the Jack application, and then put these avatars into multiple different positions and see if they can realistically fit into the situation, and how comfortable the avatars are. The avatars used in Jack are some of the industry's most biomechanically correct avatars available, since they are based on data from the 1988 Anthropometric Survey of U.S. Army Personnel. [UGS05b]

The avatars can also be instructed to perform different tasks based upon scripts written by the developer. Jack will then interface these scripts with it's own computational models of human motion, such as realistic joint limits, to determine if the avatar can complete the desired task.

Jack does not, however, focus on general avatar control, such as having avatars react to other avatars, or other stimuli from within and outside of the application. Since this is the main focus of what we are looking for in an avatar control package, Jack does not quite

suit our needs.

Also, Jack is not an external package that can be integrated with another VR package, such as VR Juggler. It is a closed source package developed to be an all in one solution. It does include some limited support for several VR devices, however we are looking for an open source package that can be easily integrated with VR Juggler.

Alias Motionbuilder

Another popular package is Alias Motionbuilder, previously having gone by the name Kaydara Motionbuilder. However Kaydara has been acquired by Alias, and in turn the software is now called Alias Motionbuilder.

Alias Motionbuilder is a commercial package aimed specifically at the task of animating avatars, or any three dimensional character. [EST03] It is in popular use in the film industry, having previously gone by the name Filmbox. [Gamasutra04] This package lets a developer import an avatar model created in any of many different modeling packages, perform the skeletal rigging on the model if necessary, and then animate the skeleton using its wide variety of animation utilities.

When animating a skeleton, Motionbuilder lets a developer use key framed animation, inverse kinematics, and motion capture data, along with many other tools. It also contains the ability to generate realistic facial movements for an avatar, using its powerful facial animation technology. An avatar can have its lip movement synchronized with recorded audio tracks or live microphone input. [Alias04b] These sort of utilities can be very useful when a developer is creating an interactive avatar. The ability to have the avatar's mouth mimic what a person is saying in real-time adds significantly to the illusion of reality.

Another benefit of Motionbuilder is its recent addition of the Story Timeline mode. The Story Timeline feature lets a developer create a complex animated scene, involving many characters, and advanced effects, such as camera fades and character interaction. However this feature is useful primarily for pre-rendered scenes, providing little use when it comes to attempting to create a fully interactive real-time world, since it's ability to interface with external sources is limited.

While Motionbuilder is very apt at animating an avatar for the film industry or a virtual world, it is not well suited to controlling a virtual avatar in real time; there is not a flexible API to dynamically influence the behavior of avatars based on events in the virtual world. It also was not designed to be interfaced with a VR package such as VRJuggler, which makes Motionbuilder difficult to use for interactive rendering in VR. Instead, a typical developer would use it to define a set of animations that could then be switched and blended between to provide a smooth lifelike character in VR using another package specifically for the avatar rendering and control.

While Motionbuilder is supported on some of the most popular platforms, specifically Windows XP, and Mac OS X 10.3, it is not supported on SGI's Irix, or Linux systems, which is a requirement for our needs. [Alias04c]

Finally, Motionbuilder is a closed source, commercial package. This is typically not an issue for movie or game development use, but considering we are looking for a package that is open source, this presents a problem.

Boston Dynamics' DI – Guy

Like Alias Motionbuilder, Boston Dynamics' DI – Guy is also a commercial package. However, DI – Guy is aimed more at aiding a developer in implementing avatars into an application, unlike Motionbuilder, which concentrates on helping an artist animate the avatars.

DI – Guy specializes in providing not only an avatar animation framework, but also a powerful API to interface with this framework, as well as many pre-made human avatars and animations for these avatars. These animations can be strung together to create lifelike avatars, since DI – Guy will provide smooth transitions between different animations. It also, like Motionbuilder, has the ability to perform advanced facial animations, to simulate emotion for the avatars. [BostonD04a]

However, DI – Guy alone does not provide the capabilities that we are looking for. It doesn't have the ability to have characters react to events, and interface with the user very well. To remedy this, there is an additional package by the name of DI – Guy Scenario. This is also produced by Boston Dynamics, and builds on the DI – Guy framework to provide the ability to develop complete scenarios, including a decision and event framework to let avatars respond to other avatars, and use input such as a joystick. The event framework can constructed through either a graphics interface, or using the scripting language Perl. [BostonD04b]

DI – Guy is even supported on most of the major platforms that we are looking for, mainly Microsoft Windows, SGI's Irix, and Linux. [USArmy04] Apple's Mac OS X, however, is not supported, and since we are aiming to use this avatar control package with

every platform that VR Juggler supports, this poses one problem.

Since DI – Guy is not open source, this means that it will not satisfy the requirements that we are looking for. This also means there is no hope for an outside developer to port Boston Dynamics' DI – Guy to Mac OS X, due to it being closed source.

Another issue with the DI – Guy framework is that its framework, while fairly open ended, does not have the specific features that we are looking for. When combined with DI – Guy Scenario, it lets a developer script the avatars either graphically, or in Perl, as was stated earlier. However, it does not let a developer script the avatars in a language of their choice, and this is one of our requirements.

From here we will be looking at more research oriented frameworks. Usually these will either be open source or at least have in depth explanations behind the mechanics that control the avatars.

Improv

The Improv system, developed by Ken Perlin, [Perlin95] has been used in some part for different purposes such as games. [Perlin02] It has even gone so far as to be spun off into a company, *Improv Technologies*, that specializes in creating animation tools for VR and games. [Improv04]

The idea of the Improv system is to separate the actual animation of the avatar from the *behavior*, or how animations are blended together to create lifelike avatars. Performing avatar control using this method allows for several distinct advantages, such as the ability to easily transmit the motion of avatar across a network to a duplicate virtual

environment, all with a low latency. Improv also lets a developer do advanced facial animation of an avatar, to further the illusion of reality for a user.

Improv also contains a method of letting external programs interface to the Improv system, and allow them to instruct avatar as to which behaviors and motions they should perform. This can be useful if the functionality built into Improv is not enough for the users needs. Perlin called this the *blackboard*. The blackboard lets a developer write programs in C, then link to the Improv libraries, and then externally instruct Improv.

While Improv does have a lot of features, it is still lacking specifically in the area of having avatars respond to unpredictable external influences. Improv is good for scripting a scene with user interaction, where an avatar will choose different behaviors to perform based upon some user input. However, if a user in the VR world moves in to an avatar, or some other event happens that influences the avatar in the middle of its sequence of motions, the avatar may be unable to respond to this event in the manner that we are looking for.

VHD

The VHD system, or Virtual Human Director, was developed at the Swiss Federal Institute of Technology in Switzerland. It aims to provide a setup that lets a director, or developer, create virtual stories by instructing all the avatars as to how they should act. [Sannier99] VHD focuses on providing a basic interface to control the movement of avatars, and which animations the avatars are performing, while at the same time providing a powerful method of integrating the avatars with the facial animations present during displays of emotion or speech. The VHD system has recently been expanded to be a complete VR

framework, titled VHD++. [Ponder03]

One of VHD's strong points is the facial animation system. It includes the ability to perform a text to speech system for the avatars, so their facial expressions closely follow the facial expressions of a real human when speaking a certain phrase.

The avatar motion control in VHD consists mostly of predefined motions, such as motion capture or key framed animation, and what the VHD team refers to as *motion motors*. Motion motors are a way of instructing an avatar to move with a certain velocity in a direction.

While some of VHD's features are very powerful, we are looking for a package with more focus on the general control of avatar movement, and having the avatars respond to events. VHD focuses more on pre-scripted scenes, so it does not suit the needs we have.

vjAvatar

The final avatar integration software package examined is vjAvatar [Hare03], which is an open source package developed at Iowa State University's Virtual Reality Application Center. vjAvatar aims to handle the main complexities of avatar integration into a VR application, by building on another open source package called Cal3D. [Cal3d05]

Cal3D is an avatar animation and rendering package, which when combined with vjAvatar lets a developer import avatars made in 3DStudio Max, Maya, or other modeling packages, directly into the developer's VRJuggler application. vjAvatar then lets the developer give the avatars simple instructions as to which animation to play, how much influence the animation should have on the avatar, which animations to stop, and to move in

a certain direction for a distance.

While vjAvatar is very strong when it comes to rapidly inserting controllable avatars into a VR application, it is lacking in the area of advanced methods of controlling the avatars. Beyond the simple commands listed above, developers are left with implementing the advanced avatar control schemes themselves, which is one of the things we are looking for the avatar integration package to handle, so vjAvatar by itself will not satisfy our needs.

While the avatar integration packages examined above each offer various features that we are looking for, no one package offers everything that is needed. To have a truely poweful and extendable solution, this thesis presents to the reader the design of the Skript avatar control library.

Chapter 4. Design and Implementation of Skript

The Skript library is designed to be an avatar scripting solution that addresses all of the issues stated in the previous chapter. It provides developers with a framework that lets them use the language of their choice to script avatar movements and animation,. The Skript system then handles the necessary modifications to the avatar movements and actions to let the supplied script be influenced by both avatars and users inside the application, as well as external forces. The external forces come in the form of modules that can be loaded into the Skript system to change the forces that affect the avatar movements and actions. This gives the developer flexibility. If a new type of influence is needed for an application, instead of rewriting the Skript library, the developer can simply write a new module.

Skript Library Structure

The Skript library has been designed with an object oriented structure in mind, and hence it's components have been separated up into individual classes to represent the components of the scripting system. These classes interact with each other through the concept of an *interface*. According to the World Wide Web Consortium, an interface is "a set of methods with no information about their implementation." [WWWC04a] The interface of an object in the scripting system can then be exposed to the scripting language for the external control. This is covered in more detail later in the thesis.

The idea of an interface is also used in the same manner to implement the pluggable influence modules that are desired. A *plugin* is a hardware or software module that adds a specific feature or service to a larger system. [Askit04] The modules that can be

plugged into the scripting system all have the same interface, so the scripting system will not care what kind of influence module it is connected to, just that it conforms to a certain interface.

Figure 2 below gives a visual representation of the different Skript components, or objects, and how they are related.

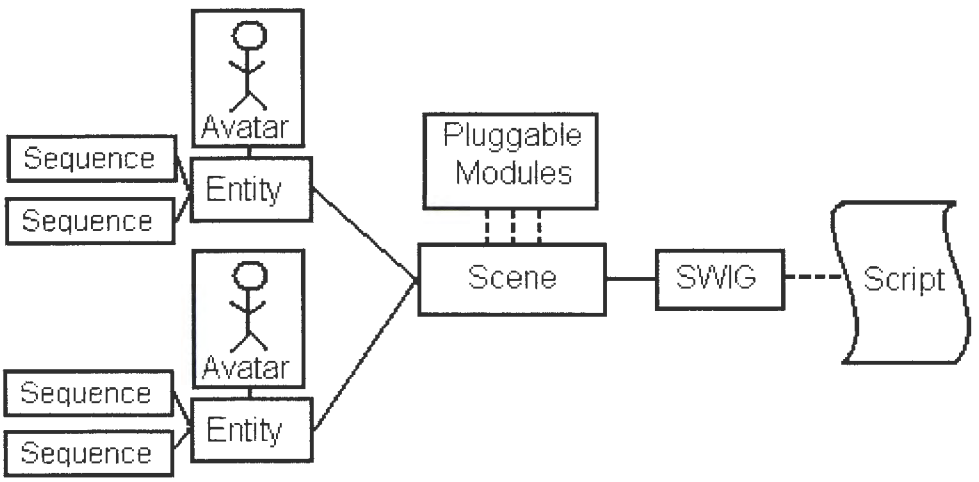

Figure 2: Skript components

A *scene* is the core Skript component. It contains objects known as entities, which for now the reader can assume are avatars with extra Skript specific data associated with them. The scene also contains information about the virtual world in which the entities operate, such as the geometry that makes up the scene, as well as any lights or sounds in the world. The scene object also contains enough information about script file used in the scene to execute different portions of the script.

The e*ntities* mentioned above are used to represent an avatar in the virtual

environment. The actual data contained in an entity will vary depending upon the information necessary to animate the avatars skeleton, and to render the avatar. If an external package is used, the avatar portion of an entity may simply be a handle or reference to the avatar object in another package. The Skript entity also contains information about its position in the virtual world, as well as what actions it is currently performing, such as if it is moving, and if so, which direction and at what velocity it is moving.

As can be seen in the diagram, entities also contain references to multiple s*equences*. A sequence is another object that will typically contain information about one or more animations that the entity's avatar can play, combined with any necessary information to smoothly transition between the animations. For example, a developer may wish to include a fade in and fade out time, for which an avatar will fade in and our of a certain animation within that specified amount of time. This could be useful to ensure smooth transitions between different animations in a sequence. An example of a sequence could be one that would instruct an avatar to perform a waving animation, and then follow that with a nodding animation in another direction. These sequences are typically referenced in the script by a unique, user readable name, such as "Wave", to simplify the process of writing a scene.

The other component that a Skript scene has are the p*luggable modules*. These are the method by which the Skript system is influenced by external sources. A plugable module consists of a class in C++ that implements the influence module interface. This interface consists of simply a callback function. Each pluggable module then has its callback function called each frame by the Skript scene, and it is in this function that a module can take into account avatar positions and scene states, and from there add the corresponding forces to the avatars. The Skript scene object will then adjust the movement and animations

of the avatars accordingly. The process by which these modules are plugged in to the scene, and how their influence affects the script to produce the resulting scene will be covered in more detail in the implementation section of the thesis.

The *SWIG* (Simple Wrapper Interface Generator) [Beazley96a] component is what allows the language flexibility that is desired. SWIG is a pre-compilation step of the application where the SWIG processor generates bindings to the Skript library by a language chosen by the developer. Again, this will be explained in more detail in the implementation section.

Implementation of the Skript Library

This section covers the actual implementation of the Skript library, and how its components work together to result in a final interactive, avatar populated scene.

To save in the development time of Skript, it was decided to base it off of one of the already existing packages that were examined in the background section. The vjAvatar package was chosen as the base for the Skript library. Considering that one of the goals of the Skript library is to be compatible with VRJuggler, using an avatar animation package built off of VRJuggler made a lot of sense. This decision was for several reasons. Not only will this mean less dependencies for the final Skript library, but programmers familiar with VRJuggler will also likely be familiar with the interface to vjAvatar.

Like VRJuggler's configuration system, Skript also has a configuration process by which the initial parameters of the scene are setup. [Bierbaum00b]

Configuration Process

For any scene that a developer wishes to create, they need a method of setting up the initial states of all the scene components. This typically includes which avatars are in the scene, as well as what world geometry is to be rendered and at what location. The scene also contains lights and sounds that need to be set up. To accomplish this task, Skript makes use of its configuration system.

Looking back at the Skript components, it can be seen that most of the objects in the Skript library are *hierarchal* in nature. In other words, a scene object contains one or more entity objects, which in turn contain one or more sequences, and so forth. It would make the most sense to base the Skript configuration system on a hierarchal data storage method.

Coincidentally, VRJuggler also bases its configuration system off of a hierarchal method by the name of XML, or the Extensible Markup Language.

According to the World Wide Web Consortium [WWWC04b], XML is defined as, "... an extremely simple dialect of SGML (Standard Generalized Markup Language). The goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML. "

XML's extreme flexibility comes from its ability to let the developer write his or her own Document Type Definition (DTD). The DTD lets applications that use the XML document know what the different valid tags are that can exist in the document, and how these tags can relate. While most applications that parse and create XML documents will have a corresponding DTD, a DTD was not created for the XML documents used by Skript. Since most of the Skript XML documents are small and not likely to be used by an

application outside of Skript, it was not deemed necessary.

To handle this parsing of the XML configuration files, this implementation of the Skript library used the open source software package XML-CppDom. According to the XML-CppDom website, it is a C++ based XML loader and writer with an internal DOM representation. [CppDom04] It is also very lightweight and high-performance. Since XML-CppDom is also the XML parser that VRJuggler's configuration system is based upon, this is another software dependency that is basically free for us to use.

There are many different elements of the scene that a developer can configure using XML. Figure 3 shows an example of an XML configuration for Skript, that shows what is necessary for a simple scene.

```
<Scene>
  <Script> simplescript.py </Script>
  <Entity name="PersonA"
          avatar="TallPerson.xml"
          personalSpace="10">
      <Sequence name="Jump">
         <Animation file="Jump.xml"
                    fadein="0.5"
                    fadeout="0.5"
                    loops="false" />
      </Sequence>
   </Entity>
</Scene>
```

Figure 3: A simple XML configuration for Skript

Skript creates a "blank" scene for the application that is then populated based on the XML configuration file set by the developer. Once the XML tree is passed to the scene, the scene can extract the necessary information, such as the number of entities in the scene, any lights or sounds, and the filename of the script to load. The XML tree must also include an XML node for each entity in the scene. Once the scene has made an entity, it passes this

entity it's respective XML node, so that the entity can then configure its self, in a proper object oriented fashion.

Once each entity gets its configuration node from the scene, it then reads several properties of the entity, such as a unique name for this entity, it's corresponding avatar file, a list of sequences, and the entity's personal space radius (This will be explained later). The entity then makes new sequences for each sequence in the XML file, and then passes the sequence it's corresponding XML node.

The sequence gets its configuration node from it's entity, and then reads the sequence's name, it's animation file, or files, as well as a possible fade in and a fade out time for the sequence, and if the sequence loops or not.

Once all of the scene elements are configured, the Skript package is ready to start executing its scene.

Running an Application Using Skript

When an application is run that makes use of the Skript library, there are two primary functions of Skript that must be called each frame. These two functions are *update* and *render*.

The render function should be called by the application using Skript during the application's rendering method. In Skript, the render function will render all of the world geometry, as well as any avatars in the scene. In this implementation of Skript, all of the rendering takes place using OpenGL [OpenGL05], since that is the graphics language that is used by vjAvatar, as well as any VRJuggler application. This will also work with most any scene graph that could be used in a VRJuggler application, since OpenSG, OpenGL

Performer, or Open Scene Graph all can base their graphics calls off of OpenGL.

The update function is where the majority of Skript's actions take place. This is where the avatars in the scene are updated, as well as the script that is controlling the scene is updated.

Skript Update

The Skript update function has two parts to it. The first part updates the script that is controlling the scene. If the application developer wishes to write the scene scripts in Python, then a Python interpreter would need to be embedded into the application to interpret the scripts. The same applies to if a developer wanted to use another scripting language. Regardless of which scripting language is used, this part of Skript will simply call an update function in this scripting language.

The update function in the script advances the *state of the scene* depending upon some parameters. The state of the scene is typically represented by some variable in the scripting language, that upon advancing, will trigger some events, or instruct avatars to move to some location. Depending upon the application, there can be multiple states in the scene, some that are completely independent, and some that depend upon other states.

The conditions that move the script forward can be anything from the simple passage of time, to the condition that an avatar of a certain name has reached a specified location.

The following example shows a very simple script that will move two avatars, "TallPerson" and "ShortPerson" to two different locations, and upon their arrival at those

locations, tell the TallPerson to play the "Jump" sequence.

```
variable state = 0
function update{
  if state = 0
    then
      move("TallPerson",5,5)
      move("ShortPerson",1,10)
      state = 1
  else if state = 1 and
    isAtDestination("TallPerson") and
    isAtDestination("ShortPerson")
    then
      playSequence("TallPerson","Jump")
      state = 2
  else ......
}
```

Figure 4: A simple script update

One of the keys to Skript's flexibility comes from the fact that the script that controls the scene is not the final authority on where an entity is, or what it is doing. If the script instructs an avatar to move to a location, the avatar is not going to simply move straight to that point. Instead, Skript will handle the actual moving of the avatars, as well as applying the correct animations to the avatars to make it look as if the avatars are walking, running, or standing.

This lets an application script writer not be concerned with the tiny details of how an avatar will get to a spot, or when. The script writer can simply focus on the overall general flow of the scene, such as where groups of avatars move to, and once they have arrived, then instruct some other group of avatars.

The process behind how an avatar actually moves from one position to another, however, is fairly complex.

Avatar Movement

When an avatar or entity is instructed by the script to move from one position to another position, it initially assigns the entity a new *destination*. An entity's destination is a two dimensional point, just like an entity's position, which is used to influence the entity's velocity.

The velocity of an entity is recomputed each frame, and is determined by several factors. The first one being the direction towards the entity's destination, which is expected. However, the final velocity of an entity is computed as a combination of this distance, as well as the combined velocity influence suggested from the pluggable influence modules.

For the purposes of explanation, this thesis will cover the development of an example crowd model influence plugin. While Skript is not limited to this type of plugin, this plugin will illustrate the example well. This plugin will also be used in the project presented in the results section of this thesis.

The crowd model plugin operates by taking into account all other entities besides the current one, and finding those entities which are within the *personal space radius* of the current entity. These entities then generate a repellent force on the current entity, with the strength of the force corresponding to the distance from the current entity. This general repellent force is then the force that the crowd model plugin applies to this entity. To add simulated user awareness to the entities, a developer can simply treat the user's position as another entity. Then as the user approaches another entity, that entity will move out of the users way. The developer must also then modify the user's special entity position according

to the users head position each frame.

The combination of an entity's destination velocity with the crowd plugin velocity will result in something similar to the following:

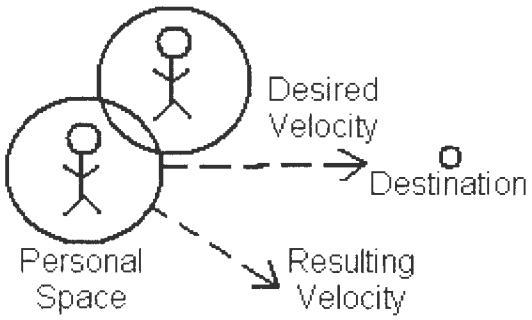

Figure 5: Avatar's new velocity and direction when near other avatar, in the special case of the crowd plugin.

As can be seen in the figure above, the velocity suggested from the destination is to the right, while the velocity that should be applied due to the crowd plugin is to the lower left. The resulting velocity is a combination of both, which is the vector going to the lower right.

Once an entity reaches its destination, it sets a flag so that the script may check to see if it can advance its state of the scene. However, the definition of *reaches its destination* is not as simple as the entity's position being equal to its destination. To prevent artifacts that can arise from this scheme, such as any entity passing over its destination point in one frame due to a sudden drop in frame rate, and in turn promptly reversing direction, we define a *destination radius* for an entity. This is simply a radius of how close the entity must be to the destination point to be considered done with its movement command.

While this scheme results in avatar movements that keep them from intersecting other avatars, and in the case shown above, the user, that is not enough. To create a truly

believable VR scene, the avatars must also transition into a walking animation at the correct times.

Automatic Avatar Animation

The process by which an avatar transitions between standing and walking at the appropriate times is called *automatic avatar animation*. This process of telling an avatar to play a walking or standing animation can not be left up to the script. Since the script is unaware of the path by which an avatar will reach its destination point, and does not know if the avatar will have to stop as another avatar passes, the process by which an avatar transitions between walking and standing animations must be handled by Skript.

To accomplish this, Skript keeps track of the magnitude of the desired velocity of an entity. If this magnitude is greater than some threshold, which is usually determined by trial and error, then Skript instructs vjAvatar to switch the avatar's animation to be a predefined walking animation. Instead of simply activating the walking animation, Skript fades out the current animation, and fades in the desired walking or standing animation, usually over a period of one half second. The threshold mentioned earlier is necessary to insure that avatars do not rapidly oscillates between walking and standing.

While this works well for most avatars and animations, sometimes an avatar may be in an animation which can not easily be interrupted. If this is the case, then the script may simply set a flag for an avatar. In this case the automatic avatar animation will not occur until the current animation has finished. If this is so, the avatar will also ignore the influences from the pluggable modules.

The final piece necessary to have realistic avatar movement is to have Skript handle the direction that the avatar is facing when an avatar is in the middle of a movement command. This way if an avatar must change its movement velocity, it will still be facing the correct direction. The direction that an avatar should face is determined by its velocity, as well as a dampening factor over time. This means that if an avatar is moving a certain velocity and must abruptly change its movement velocity, the avatar will quickly, but not instantaneously, change its facing direction. This results in a smooth, visually pleasing directional transformation.

Scripting Language Flexibility

The last area of Skript to be discussed is its language flexibility. This benefits of having a language flexible avatar scripting framework is it facilitates its integration in the available VR application frameworks, as well as to accommodate for developer's preferences.

As stated earlier, the SWIG processor will generate a set of language bindings for the Skript library for the language of the developers choice. Some of the languages that SWIG can generate the bindings for are Perl [Beazley98a], Python [Beazley96b], or Tcl [Beazley98b], among others. An interpreter for the language of choice is then embedded into the application, and the bindings to the scene object are setup. The details of how to embed an interpreter for the language of choice, and how to setup the scene bindings are language dependent, beyond the scope of this paper, and well documented on SWIG's web site. So they are not discussed here.

These language bindings then expose some interface from the C++ side of the

application to the scripting language. As can be seen in Figure 6, the bindings are all to the scene class, while the Skript library is built using C++, the interface to the scene from the script is fairly simple, and therefore doesn't need to be exposed in an object oriented fashion.

`void move(avatarname, xposition, yposition)` - Instructs an Avatar of a specified name to start to move from its current position to the destination position specified.

`bool isAtDestination(avatarname)` - Tells if an Avatar has reached a position specified by the last move command.

`void playSequence(avatarname, sequencename)` - Instructs an Avatar to start to play a specified sequence.

`bool isSequenceFinished(avatarname)` - Tells if an Avatar has finished its last specified sequence.

`void followPath(avatarname, pathname)` - Instructs an Avatar to start to follow a specified path file.

Figure 6: Scene interface as seen by the script

Since the language is chosen by the developer at the time the application is compiled, scripts written for an application using a different scripting language can not be used unless the script is converted to the new scripting language.

Once the scripting system had been developed it was integrated into an existing application framework, both to test the scripting system, as well as to add interactive avatars into the application.

Chapter 5. Results: Virtual Hindu Temple

The application used as a testbed for the Skript library is the Virtual Hindu Temple, developed at Iowa State University's Virtual Reality Application Center. [VRAC02] This application was the result of a collaborative effort between the Department of Religious Studies and the Virtual Reality Application Center. The goal of the project was to reconstruct a Hindu Temple in Virtual Reality, specifically a temple to the Hindu god Krishna in Vrindavan, India. This way, students of Iowa State's Religious Studies department, who were studying the Hindu religion, could further their knowledge of the religion by being immersed in the virtual temple and the service that goes on inside it.

Because the goal of this application, like most VR applications, is to immerse the users completely in the application, the visual and control quality of the avatars was critical. If the avatars did not respond quickly and accurately to the user, the illusion of reality would be compromised. By using Skript to handle the avatar integration and control, the application was up and running quickly. Skript's flexibility also allowed the developers to hold multiple review meetings to refine the avatars and their behaviors during the ceremony.

Figure 7: The Virtual Hindu Temple application

The temple model was constructed in 3D Studio Max™ from photographs and video footage of the real temple. Volunteer Indian students at Iowa State University served as the models for the creation of the temple avatars. The avatars' animations and movement patterns were modeled from the video footage.

There were a total of nine avatars used in the application including the regular people who visited the temple, a priest to run the service, and finally musicians in the back of the temple. All of these avatars were assigned to entities in the scene configuration file, and then the avatar influence module developed earlier in this thesis was plugged into Skript. Python was used as the scripting language for this application, since it was known by the developers, and hence allowed rapid development. Entities then had their basic movements and animations scripted in a Python file. The result was a scene where the temple visitors gradually entered into the temple and walked about until every temple visitor had entered the temple, at which point the priest entity came out and conducted a short service. When he was

finished, the individuals went up to the priest to be blessed, and then gradually left the temple. All the while musicians in the back of the temple provided the worship music.

The user was also added to the application as an entity, however the user's position was obtained from the head tracking in the VR application, so the temple visitors could walk around the user if the user was in the visitors path. This also allowed a user to "push" the visitors about by walking into them. This helped the user feel as if he or she was an active part of the temple crowd, since the avatars were aware of the users presence and adjusted accordingly.

The resulting application was shown at the conferences Supercomputing 2002 (Baltimore), ACM SIGGRAPH 2003 (Los Angeles), and Mundo Internet 2004 (Madrid, Spain). It was also at the Iowa State University Brunnier museum for one month, and is regularly used at Iowa State University for lectures and discussions by faculty of the Religious Studies Department.

Chapter 6. Conclusion

The goal of this research was to create a framework that would aid in the integration of flexible, user-aware avatars in a VR application. This was accomplished by developing a dynamic scripting system that was built off of vjAvatar, to save time, as well as make it easier for a developer familiar with VRJuggler and vjAvatar to use it.

The scripting system also allowed us to plug in influence modules, so developers could expand upon the avatar influence system with minimal effort. A simple module was also developed, a crowd simulation module, which allowed a group of avatars to be aware of each other and move accordingly.

Finally, the developed scripting system was applied to a virtual heritage application, the Virtual Hindu Temple. This application received positive feedback from users who experienced it, and it has been shown at several conferences.

The development time of this application was also significantly shorter than it would have been without the use of the Skript library. The ability to enable our Religious Studies collaborators to write scripts in a language other than C and C++ helped them quickly test avatar movements and animations.

Chapter 7. Future work

There are several areas of the scripting system that could be developed to be more robust and useful to more applications. The current Skript system is designed around avatar movement in a two dimensional plane. For some applications, it would be beneficial to expand it to three dimensions, to allow for more complex avatar movements on multiple levels of a scene.

Another area that could be expanded upon would be the automatic avatar animation. Right now it allows an avatar to transition between walking and standing animations played at a fixed speed. It would be beneficial to allow the amount and types of animations that are automatically triggered to be specified by the developer in the configuration XML file. This way additional automatic animations could be added in, depending upon the needs of the application.

A final area of Skript that could benefit from additional work would be the limitation of one type of scripting language for all the scripts used. For some projects with multiple developers, they may wish to each write a portion of the overall scene in their own preferred language. Skript could be expanded to include the capability for multiple scripting languages executing multiple scripts at different times in the same scene, and allow for these languages to communicate between each other.

Chapter 8. References

[Alias04a] Alias Maya 6.5, The Maya Family, http://www.alias.com/eng/products-services/maya/index.shtml, (retrieved 15 April, 2005)

[Alias04b] Alias Motionbuilder Overview, http://www.alias.com/eng/products-services/motionbuilder/file/motionbuilder6_overview.pdf, (retrieved 15 April, 2005)

[Alias04c] Alias Motionbuilder System Requirements, http://www.alias.com/eng/products-services/motionbuilder/system_requirements.shtml, (retrieved 15 April, 2005)

[Allbeck98] J. Allbeck, N. Badler, "Avatars á la Snow Crash", Center for Human Modeling and Simulation, University of Pennsylvania, 1998

[Allen03] J. Allen, "Creating Cal3D Characters with 3D Studio MAX", http://cal3d.sourceforge.net/modeling/tutorial.html, 2003, (retrieved 12 December, 2004)

[Askit04] The University of Queensland's Ask IT, Definition of Plugin, http://askit.uq.edu.au/glossary/glossaryo-z.html, (retrieved 5 January, 2005)

[Beazley96a] D. Beazley, "SWIG: An Easy to Use Tool for Integrating Scripting Languages with C and C++", Presented at the 4th Annual Tcl/Tk Workshop, 1996

[Beazley96b] D. Beazley, "Using SWIG to Control, Prototype, and Debug C Programs with Python", Submitted to the 4th International Python Conference, 1996

[Beazley98a] D. Beazley, D. Fletcher, D. Dumont, "Perl Extension Building with SWIG", 1998

[Beazley98b] D. Beazley, "Tcl Extension Building With SWIG", Department of Computer Science, University of Chicago, 1998

[Bierbaum00a] A. Bierbaum, "VR Juggler: A Virtual Platform for Virtual Reality Application Development", Masters Thesis, Iowa State University, 2000

[Bierbaum00b] A. Bierbaum, C. Cruz-Neira, "Run-time reconfiguration in VR Juggler", Presented at the 4th Annual Immersive Projection Technology Workshop, 2000

[BostonD04a] Boston Dynamics' DI-Guy, The Industry Standard in Real-Time Human Simulation, http://www.bdi.com/content/sec.php?section=diguy, (retrieved 12 December, 2004)

[BostonD04b] Boston Dynamics' DI-Guy Scenario, http://www.bdi.com/content/sec.php?section=discen, (retrieved 12 December, 2004)

[Cal3d05] Cal3d – Character Animation Library, http://cal3d.sourceforge.net/, (retrieved 15 April, 2005)

[CppDom04] CppDom Website, http://sourceforge.net/projects/xml-cppdom, (retrieved 8 Feburary, 2005)

[Discreet05] Discreet 3ds max overview, http://www4.discreet.com/3dsmax/, 2005

[EST03] Kaydara MOTIONBUILDER™ Overview, http://www.est-kl.com/software/kaydara/motionbuilder.html, (retrieved 12 December, 2004)

[Gamasutra04] Gamasutra Product Review: Motionbuilder 5.5, http://www.gamasutra.com/features/20040623/chambers_01.shtml, (retrieved 15 January, 2005)

[Haag02] S. Haag, "I-Series Computing Concepts", Mc Graw Hill Online Learning Center, 2002

[Hagsand96] O. Hagsand, "Interative Multiuser VEs in the DIVE System", Swedish Institute of Computer Science, 1996

[Hare03] J. Hare, "The vjAvatar Library: A toolkit for development of avatars in virtual reality applications", Masters Thesis, Iowa State University, 2004

[Improv04] Improv Technologies, http://improv-tech.com, (retrieved 5 April, 2004)

[Merriam05] Merriam-Webster Online, Definition of Avatar, http://www.m-w.com (retrieved 12 January, 2005)

[Muuse97] S. Musse, D. Thalmann, "A Model of Human Crowd Behavior: Group Inter-Relationship and Collision Detection Analysis", Computer Graphics Lab, Swiss Federal Institute of Technology, 1997

[OpenGL05] The OpenGL Website, http://www.opengl.org, (retrieved 12 January, 2005)

[OSG05] The Open Scene Graph, http://www.openscenegraph.org, (retrieved 12 January, 2005)

[Perlin02] K. Perlin, "Better acting in computer games: the use of procedural methods", Media Research Laboratory, Department of Computer Science, New York University, 2002

[Perlin95] K. Perlin, A. Goldbery, "Improv: A System for Scripting Interactive Avatars in Virtual Worlds", Media Research Laboratory, Department of Computer Science, New York University, 1995

[Phillips88] C. Phillips, N. Badler, "JACK: a toolkit for manipulating articulated figures", Symposium on User Interface Software and Technology, 1988

[Ponder03] M. Ponder, G. Papagiannakis, T. Molet, N. Thalmann, D. Thalmann, "VHD++ Development Framework: Towards Extendible, Component Based VR/AR Simulation Engine Featuring Advanced Virtual Character Technologies", The Swiss Federal Institute of Technology, and the University of Geneva, 2003

[Reiners02] D. Reiners, G. Voss, J. Behr. OpenSG: Basic Concepts, Feburary 2002

[SGI05] SGI's OpenGL Performer, http://www.sgi.com/products/software/performer, (retrieved 12 January, 2005)

[Sannier99] G. Sannier, S. Balcisoy, N. Thalmann, D. Thalmann, "VHD: a system for directing real-time virtual actors", The University of Geneva and the Swiss Federal Institute of Technology, 1999

[Stephenson93] N. Stephenson, "Snow Crash", Bantam Spectra, 1992

[Szarowicz01] A. Szarowicz, J. Amiguet-Vercher, P. Forte, "Multiagent interaction for crowd scene simulation", Kingston University, 2001

[UGS05a] UGS: Product Lyfecycle Management Solutions, http://www.usg.com/, (retrieved 18 January, 2005)

[UGS05b] UGS: E-Factory–Jack, http://www.ugs.com/products/efactory/jack/, (retrieved 18 January, 2005)

[USArmy04] Commercial Terrain Visualization Software Product Information: DI-Guy http://www.tec.army.mil/research/products/TD/tvd/survey/DIGuy.html, (retrieved 12 December, 2005)

[VRAC02] Iowa State's Virtual Hindu Temple Website, http://www.vrac.iastate.edu/research/sites/htemple, (retrieved 15 April, 2005)

[WWWC04a] The World Wide Web Consortium, Definition of Interface, http://www.w3.org, (retrieved 22 January, 2005)

[WWWC04b] The World Wide Web Consortium, Definition of XML, http://www.w3.org, (retrieved 22 January, 2005)

[Yang03] X. Yang, D. Petriu, T. Whalen, E. Petriu, "Script Language for Avatar Animation in 3D Virtual Environments", VECIMS 2003