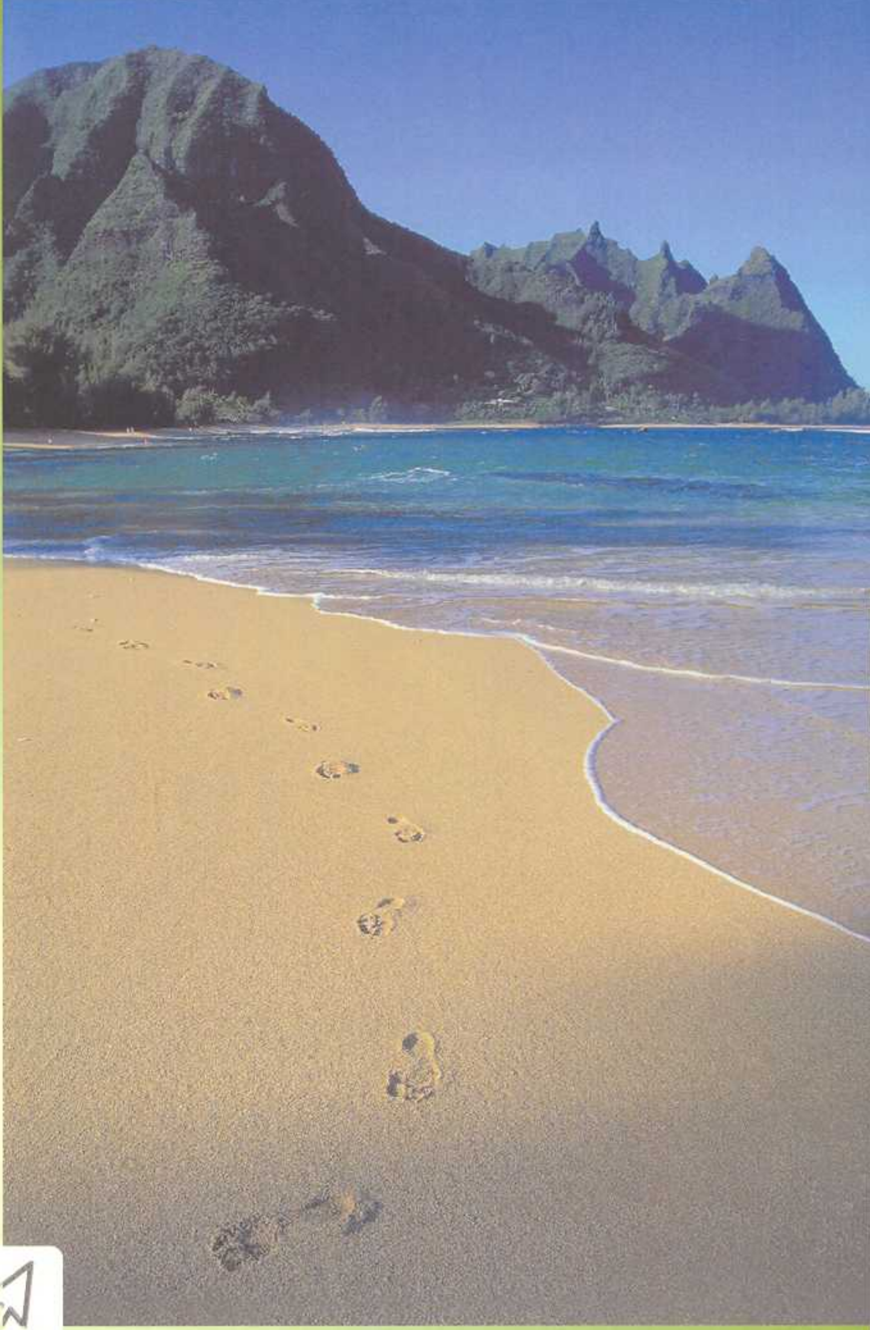

Proceedings of the Eighth IASTED International Conference on

INTERNET AND MULTIMEDIA SYSTEMS AND APPLICATIONS

Editor: M.H. Hamza

August 16 – 18, 2004
Kauai, Hawaii, USA

A Publication of The International Association of Science and Technology for Development - IASTED

ISBN: 0-88986-420-9
ISSN: 1482-7905

ACTA Press

Anaheim • Calgary • Zurich

Proceedings of the Eighth IASTED International Conference
INTERNET & MULTIMEDIA SYSTEMS & APPLICATIONS
August 16-18, 2004, Kauai, Hawaii, USA
© ACTA Press, Calgary, Canada

VOICE BASED INTERACTIVE MODIFICATION OF WEB BASED MOVIES

Bonita Simoes and Arvind K. Bansal
Distributed Multimedia and Cognition Laboratory
Department of Computer Science
Kent State University, Kent, OH 44242, USA
Email: bsimoes@cs.kent.edu and arvind@cs.kent.edu
Phone: +1.330.672.9035, FAX: +1.330.672.7824

ABSTRACT
The Internet is becoming a massive knowledge resource for 3D multimedia information such as movies, news clips, and children's stories. Multimedia information can be interactively modified using multimodal interfaces for entertainment to incorporate subjective perception in a scenario and to analyze situations using many what-if scenario modifications. In this paper we describe an XML voice interaction model for 3D XML movies retrieved over the Internet. This voice model can be used to dynamically and interactively modify and edit 3D movies and archive or transmit them to others over the Internet. Performance analysis has been presented.

KEY WORDS
3D movies, human-computer interfaces, Internet, voice interaction, web movies, XML

1. Introduction

The demand for interactive and next-generation 3D characters and 3D movies is growing tremendously. The designing and rendering of realistic 3D interactive characters and movies has become a reality. With distributed object databases spanning across the Internet, it has become easier for users to combine and reuse resources like textures, sounds and 3D movies, from various databases distributed over the Internet.

The integration of a generic Internet based language such as XML, multimedia technologies, Internet, and voice recognition technologies makes many applications possible that were considered science fiction ten years ago. For example, children could download movies across cultures using the Internet, interact with and modify the animated characters to suit their culture to live up to their fantasy for the ultimate entertainment experience; lawyers could modify and recreate in real time a new story of a crime scene based upon a generic story giving multiple possibilities; airline agencies could interactively modify a plane crash scenario to see multiple possibilities; a chemist could model and interact with 3D cell models in real time using voice commands.

There is a need for a scheme to dynamically modify realistic 3D movies over the Internet based upon voice controlled interaction. Speech input is a natural choice of interaction since speech recognition engines of today achieve high accuracy rates.

A lot of research has been done in the areas of voice recognition, avatars – animated computer characters [1], social agents [2], interactive movies [3], story telling, [4] and Internet based modeling [5]. There are also interactive schemes for helping children learn through interaction [6]. We have been influenced by these systems in developing our model.

This dynamic voice interaction model is part of a large 3D XML based interactive model [7] developed and implemented by the authors. Figure 1 shows the overall model of the system. The server end consists of an XML representation of a 3D movie. The client end parses XML movie; interacts with uses using voice module, interactively extends the XML based movies, composes scenes dynamically, and renders the animated characters in real time. A script interpreter parses text commands sent to the voice sub-system via a speech emulator.

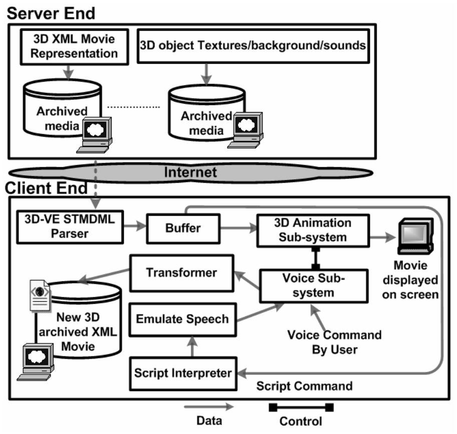

Figure 1. A model for voice modifiable movies

In this paper, we describe the real time voice interaction model. The scene description and animation of 3D objects is described in [7]. The major contributions of this paper are as follows:

1) A voice-based real time control of 3-D animated characters has been developed for XML movies.
2) The XML files are modified using voice-based commands used to interactively generate new variations of an Internet based movie.

The paper is organized as follows. Section 2 briefly describes background concepts and definitions. Section 3 introduces the voice interaction model and describes the voice grammar used to build voice commands. The system implementation is described in Section 4. A performance analysis is presented in Section 5. Section 6 describes some related work in animation and scripting movies. The last section concludes the work and discusses some future work.

2. Background and Definitions

A *hierarchical graph* is a multilayered graph such that a sub-graph might be embedded inside a node or an edge. Each layer is an abstraction of embedded sub-graphs. An edge represents one or more edges between the embedded sub-graphs. The graph at the lowest layer containing media objects is called an *object graph*.

A *3D mesh* is a set of faces. Each face lies on a plane. The faces are made up of a set of vertices. A triangulated mesh is modeled as a set of faces with three vertices represented as (x_i, y_i, z_i). Realism is directly proportional to the number of faces.

The animated characters have been modeled using hierarchical graphs and 3D meshes. The motion of 3D objects is generated by dynamically changing the position and orientation of nodes and edges in an object graph.

Due to the connectivity of nodes in animated skeletal objects, *skeletal* animation [8] has been used. Skeletal animation provides a sequence of different positions of the nodes in the object graph at different time instances. Movement of one node also affects the movement of neighboring nodes. This movement effect on neighboring nodes is modeled by associating a unique weight w_{ij} with each edge. The matrix of these weights is used to model effect of animation on connected nodes. The animation effect is transitive. By multiplying the weights w_{ij} (between nodes v_i and v_j) and w_{jk} (between nodes v_j and v_k), the effect of movement of a node v_i on v_k is computed.

Any complex animation of 3D objects is decomposed into a combination of three basic operations: translation, rotation and scaling. These three basic operations are modeled using *transformation matrices.* Transformations are combined by multiplying the corresponding transformation matrices in a specific order.

3. The Voice Interaction Model

The voice interaction model consists of a speech recognition module, voice command processor and an object motion controller (see Figure 2). Every object in a scene is uniquely identified by a name. Each object in a scene has a unique animation set and a local object controller associated with it. Each *animation set* is an ordered set of *time instances*.. There are multiple attributes such as *animation speed*, *movement speed*, and *looping* that are associated with an animation set. The *animation speed* determines rendering speed of animation set. The *movement speed* determines how fast an object moves during the rendering of an animation set. *Looping* determines if the animation repeats itself. Animation speed is directly proportional to the rendering speed.

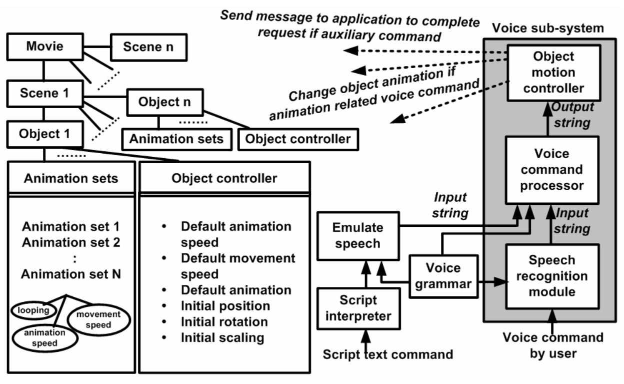

Figure 2. The voice interaction system

The speech recognition module takes a voice command as input, denoted as an *input string*, and passes it to the voice command processor. The voice command processor determines the type of the command rule, and transforms the *input string* to a set of phrases, denoted as an *output string*.

There are two types of *voicecommand* rules: *auxiliary* or *animation* rule. An *auxiliary* rule helps a user navigate through menus. An example of an *auxiliary* command would be "Help me", "Show actions" or "Close the player". An *animation* command instructs the local object controller to change the current animation of the corresponding object to perform an action. All phrases to be recognized are specified in a *list* construct. The field *property-name* of a *phrase* contains semantic information about that phrase for the global object motion controller to understand. The *recognized_text* is a sequence of words that is mapped to the semantic value stored in the field *property-name*. The grammar for parsing voice commands is given in Figure 3.

```
<voicecommands>::= '<voicecommands>' {<rule>}*
                          '</voicecommands>'
<rule>::= '<rule' <rule_attr>'>'{<list>}* {<rule-reference>}*{<o>}*
            '</rule>'
<rule_attr> ::= [<identifier>] <name> [<rule_state>]
<name> ::= "animation" | "auxiliary" | "person" | "action" |
              "direction" | "amount"
< rule_state> ::= "active" | "inactive"
<list> ::= '<list>' {<phrase>}+ '</list>'
<phrase> ::= '<phrase' <property-name> '>' <recognized_text>
               '</phrase>'
<property-name> ::= <object-name> | <set-name> |
                        <direction> | {<animation_amount>}+
<object-name> ::= <characters>
<set-name> ::= <characters>
<direction> ::= "left" | "right" | "up" | "down"
<recognized-text> ::= <characters>
<rule-reference> ::= '<rule-reference' <name> '/>'
<o> ::= '<o>' <rule-reference> '</o>'
```

Figure 3. Voice grammar for the system

Each *output string* is a quadruple of the form (*object-name, action, direction, amount*). An *object-name* uniquely refers to an animated object. To enhance user friendliness multiple *input strings* might refer to the same object using aliases. All aliases of the same object map to the same *object-name* in the *output string*. An *action* refers to one or more animation sets. An action could be *simple* or *composite*. A *simple action* refers to an animation set. A *composite action* is a partially ordered set of simple actions. Both simple and composite actions can have aliases that map to the same name in the *output string*. For example, the simple action "walk" or "go" are mapped to the same animation set. The *direction* can be either left, right, up or down. The optional field *amount* specifies the time an object's animation set is rendered.

Besides receiving a voice command, the voice sub-system also takes text input from a script in an XML movie file. The script interpreter passes the text command to the "Emulate speech" module, whose job is to treat the text as if it were speech. The "Emulate speech" module passes the text voice command denoted as an *input string* to the voice command processor where it is transformed into an *output string*.

An *input string* can have a varying number of words since multiple words may be joined to form an action. An *input string* is terminated by a pause above a time-threshold. There should not be a significant pause between the spoken words in a sentence to avoid discontinuity in an *input string*.

Example 1: This example illustrates an *input string*: "*Zombie walk limping left five*". In this example, *object-name* is "Zombie", *action* is "WalkLimping", *direction* is "left", and *amount* is 5. Note that there are five words in the *input string*, and two words "walk limping" map to the animation set "WalkLimping". The animation set "WalkLimping" contains time instances at which the object graph of the Zombie changes.

The *output string* is sent to a global object motion controller that instructs the local object controller of the corresponding object to update its current animation to perform the specified action. The renderer carries out animations. The application handles the request for auxillary commands. An abstract description of the voice recognition process is shown in Figure 4.

```
Input: Voice grammar for the scene;
Output: Output strings for the corresponding voice commands;
while true do {
 wait for a voice or text script command;
 if a voice command or text script command is found {
  convert voice command to text input string $I = \{w_1, w_2, \ldots, w_n\}$;
  parse input string to get an output string O = (object-name,
      action, direction, amount);
  frame_instance ß amount * frames/sec;
  validate the quadruple (object-name, action, direction,
     frame_instance) for proper mapping to the object-name's
     animation set;
  if (validated) update the animation-possible state variable for
     the corresponding object having the name object-name;
} //end if } // end while
```

Figure 4. Processing voice commands

During validation, the quadruple *O* is first checked to see if an object exists. If an object exists, the next step checks if the *action* in the *output string* exists for the *object-name*. After validation, the output string is processed by the object motion controller.

After a user is done with animating the objects, user interaction can be saved and used later by another user. During a save, the movie is modified thereby creating a new dynamically modified XML movie file.

3.1 The Object Motion Controller

This system consists of a global object motion controller and multiple local object controllers. Each object in the scene has its own local state of its current animation. The global object motion controller instructs an object's local controller to change the corresponding object's animation state based upon an *output string*. The state transition of the object motion controller is depicted in Figure 5. In the absence of an *output string*, local controllers for provide *default animation* to the corresponding objects to achieve realism.

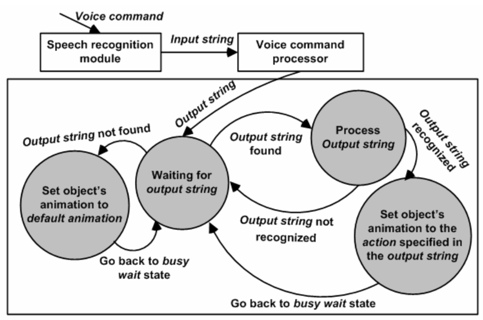
Voice command
Speech recognition module
Input string
Voice command processor
Output string
Output string not found
Output string found
Process Output string
Output string recognized
Set object's animation to default animation
Waiting for output string
Output string not recognized
Set object's animation to the action specified in the output string
Go back to busy wait state
Go back to busy wait state

Figure 5. State diagram of the motion controller

The global object motion controller is always in a *busy wait* state until an *output string* is found. After receiving an *output string*, the global object motion controller checks for the object-name. If the pair (*action*, *object-name*) exists, the global object motion controller instructs the individual local controller to identify the corresponding animation matrix, and change the state of the object. The *animation-possible* state is set to true and the object's *action*, *direction* and *amount* parameters are updated based on the *output string*. The object's local controller performs the animation for the desired amount of time based upon new parameter values. After this action, the object's local controller puts the object back in *default animation* state.

After instructing an object's local controller, the global object motion controller resumes in the busy-wait loop, waiting for another output string. If the *input string* is not understood by the speech recognition system, the global object motion controller remains in a busy-wait loop.

Collision detection has been implemented to avoid collision by detecting if objects try to occupy the same area in a scene. The objects are not allowed to walk through the floor, walls, or any other objects in a scene.

3.2 Concurrent Animation of Objects

This model supports concurrent animation of two or more objects in a scene since each object has a local controller for animation. After a voice command is issued, the user need not wait for the command to be completed before issuing another command.

Besides issuing a command to multiple local object controllers and have them perform their animation concurrently, the global object motion controller can also interrupt local controllers, and overwrite the animation state parameters of the corresponding objects thereby changing their animations.

4. System Implementation

This system has been implemented in C++, which provides faster rendering than Java. The parser is implemented in the .NET XML framework and the renderer uses the DirectX graphics API (http://www.microsoft.com/directx) along with other C++ user defined libraries built on DirectX. Microsoft Speech API (*http://www.microsoft.com/speech*) is used for speech recognition.

Figure 6 shows the client implementation model. The client parses the transmitted XML movie file and builds the scene, object and animation databases. The client also downloads other resources from other servers in a separate thread. The speech recognition module is then started and based on the validity of the voice commands, the objects' state is changed, and the objects in the scene are rendered if no collision is found. Voice grammar as shown in Figure 3 is loaded when a speech recognition thread starts. There are two threads running in this system; the speech recognition thread and the main program rendering thread. The global object motion controller is part of the speech recognition thread and the local controllers are part of the main program thread.

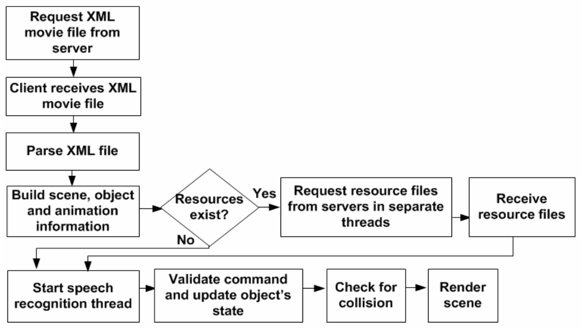
Request XML movie file from server
Client receives XML movie file
Parse XML file
Build scene, object and animation information
Resources exist?
Yes
No
Request resource files from servers in separate threads
Receive resource files
Start speech recognition thread
Validate command and update object's state
Check for collision
Render scene

Figure 6. Client implementation model

The output generated by the parser is a Document Object Model (DOM) tree of the XML that is constructed in memory. A Document Type Definition (DTD) is used to ensure proper validation of an XML movie file. An abstract description of the XMLParser class is shown in Figure 7.

```
class XmlParser {
   parseXmlFile(String *file);     // input XML movie file.
   parseSceneMeshXml(String*innerXml, String* sMeshName);
   parse3DObject(XmlNode* Node);
   buildGraph(XmlNode* Node, StreamWriter* meshwriter,
      StreamWriter* logwriter, int graphid, XmlNode* meshnode);
   parseObjectMeshXml(String* innerXml, StreamWriter*
      meshwriter, StreamWriter* logwriter, int graphid);
   parseObjectAnimationXml(XmlNode* animationNode,
      StreamWriter* meshwriter, StreamWriter* logwriter);
   parseTextObject(XmlNode* Node);
   parseAudioObject(XmlNode* Node);
   parseImageObject(XmlNode* Node);
   validate();     //validate XML movie with DTD
}; //end XmlParser class definition
```

Figure 7. The *XMLParser* class definition

An abstract description of the speech recognition process is given in Figure 8. Input strings are mapped into output strings based on the property value associated with input strings with respect to the voice grammar. The vector *Objects* denotes the global object motion controller that keeps track of an object's name in a hash table, along with its other properties. Once an object is found, the global object motion controller calls the corresponding local object controller, which changes the action for the object based on the parameters passed to it. The local object controller is a C++ object created for each 3D dynamic mesh. The local controller changes the object's action and movement parameters on the x, y and z axes until the amount of time specified is complete.

```
Algorithm: SpeechRecognition
Input: XML Movie file with Voice Grammar
For each object in a scene
    Initialize Objects[name].action to default from movie file;
Start and initialize speech recognition component thread;
If (SceneController[CurrentScene].movieExists) {
   Get the next movie script command;
   Pass through the speech emulator to recognize words;
   Disable microphone;
}
else { Enable microphone; }
while (voice input exists) {
  get recognized input string from the recognition engine;
  if input string has properties associated with it {
     Get property value for each word/phrase in the input string
          based on the voice grammar separated by spaces;
     Split the property values delimited by spaces into 4
          separate variables; name, action, direction and amount
          denoted as the output string
     Search for name in the array Objects for each object
          indexed in a hash table;
     if object name is found {
          Objects[name].direction = direction;
          Objects[name].action = action;
          Objects[name].amount = amount;
    }}
    call LocalController(name, action, direction, amount);
    if (recognized==true) show visual text recognition on screen;
}
Uninitalize speech recognition component and end thread;
```

Figure 8. Speech recognition process

5. Performance Evaluation

Various experiments were done on this system using different parameters. We present here the response time to recognize a voice command and perform the corresponding action (see Figure 9) For unambiguous commands, the average response time falls between 290ms and 346ms. An ambiguous command's average response time is almost doubled (between 595ms and 720ms). Ambiguity is introduced due to multi-word phrases as well as optional rules in the grammar. The following examples illustrate this point.

Example 2: This example illustrates ambiguity due to multi-word phrases. Consider a voice command "*Zombie walk left twenty*". The speech recognition engine needs to determine if the user will decide to say "*twenty one*" or "*twenty two*", etc… or if the user really meant to say just "*twenty*". The default time that the speech recognition engine waits before it completes recognizing a single command is 150ms. For this reason the command will take more time than a command that did not have any ambiguity such as "*Zombie walk left ten*".

Example 3: This example illustrates ambiguity due to optional phrases. Consider the voice command "*Zombie punch left*". The speech recognition engine takes more time to traverse the parse tree and build hypotheses for an input voice command when ambiguity is introduced due to optional phrases that need to be resolved in the input voice command. This holds true for all input voice commands that optionally neglect the amount parameter.

The speech recognition system runs in a separate thread from the renderer and hence there is no delay in rendering one or more objects if another voice command is issued before the previous one finishes. The increase in the number of objects did not cause the speech recognition system any delay. The number of words in the voice grammar did not affect the speech recognition response time nor did it affect the recognition accuracy. This is because the grammar consisted of relatively few words. The recognition accuracy is around 98% or greater in a noise free environment for both male and female voices.

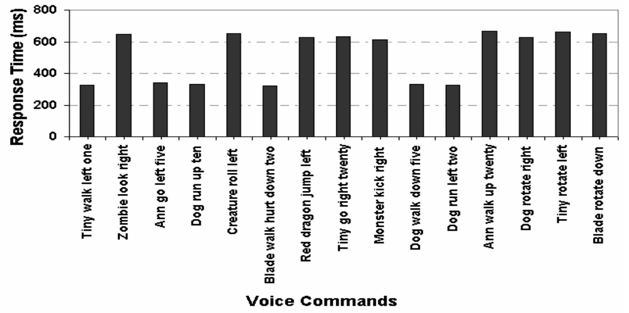

Figure 9. Voice commands vs. recognition time

The speech recognition is not affected by slight background noise such as background music that is played during the movie. Even if the volume of the background music was increased, the accuracy of recognition was almost the same, when the speakers were not directly facing the user or close to the microphone. However, the system accuracy falls in the presence of nearby noise source such as music, a printer printing a document and people talking loudly.

It was never found that the background noise/music made the objects move by causing a valid command to be recognized. The background noise would basically recognize words that were not in the grammar vocabulary but they were discarded by the system. This noise may cause some problem when vocabulary set becomes large. Since we use character identity before a command, only the animated character responds to a command. It was also found that validating voice commands has little or no significant overhead in the system.

6. Related Work

Although we have not seen any system that provides dynamic voice based modification of Internet transmitted XML based 3D movies, we have been influenced by many systems used to develop 3D animated objects and voice based interaction.

VRML [9] is a text and binary based 3D graphics standard used to model 3D content on the Internet. X-VRML [10] is an XML based extension of VRML to include variables, conditional statements, expressions, and loops to dynamically generate 3D scenes. Both X-VRML and VRML do not have voice support. VRML is not XML based and hence not completely interoperable. None of the systems have the capability to dynamically alter the scenario based upon voice interaction.

X3D [11] extends VRML's capabilities and expresses 3D VRML models using XML. XSTEP [12] is another XML-based markup language that uses Prolog like parameterization for easy interaction with agents. Voice based animation of 3D content is not incorporated in X3D or XSTEP.

Avatar Markup Language (AML) [13] is an XML based language that makes use of the MPEG-4 standard. It uses MPEG-4's low-level face and body animation parameters to create high-level command scripts. The AML processor produces MPEG-4 compliant Face and Body Animation (FBA) streams. AML has text-to-speech capabilities but has not incorporated speech recognition into the system.

Improv [14] is a system for high level scripting of interactive actors in virtual worlds. The system allows the animator to create predefined rules that govern the way the virtual actors behave. These rules need to be followed and they control the animation using scripts. The two main modules related to this system are the animation engine and the behavior engine that interact with the 3D characters. The scripting language is English like statements that are linked to the system.

STMDML [15] used hierarchical graphs to represent objects. However, the model is used to reduce the transmission bandwidth by exploiting object reuse and client end object reconstruction. However, these systems currently limited to 2D movies, do not provide voice based interactivity and extensibility of movies.

A number of speech interfaces including command-and-control interfaces, natural language interfaces and multimodal interfaces have been developed for the animation of virtual characters. In [16], natural language parsing and goal planning is used to control the behavior of virtual agents. In [17], natural language control is used for the interactive 3D animation in computer games such as Doom. A parser generates messages to an animation layer of the system, which in turn generates the animation of the players in the game.

None of these systems use an XML format to represent animation, and do not support dynamic modification of movies. Most of them have interactivity using scripts. Others have voice-based control but no recording of commands issued to create a new movie file.

7. Conclusion and Future Work

Using the system described in this paper, a user can download a 3D XML movie over the Internet along with background, textures and sound. The user can then interact with characters in a movie in real time using voice commands, thereby dynamically generating a new movie that can be used in the future by the same user or a different user. In this way, a user can realistically bring about a persistent twist to a movie's predefined story and its ending. The new movie can be archived and become part of multimedia information on the Internet.

Currently the voice based commands include only simple sentences, and are limited by the set of vocabulary and a limited set of parameters. For a general purpose control of the social agents, the system has to be integrated with natural language understanding systems and the semantic web, which can facilitate mapping of complex natural language commands to a core set of voice commands. The current implementation cannot discriminate between multiple voices. The system is being extended to include separation of voices in spoken words. We are also developing morph enabled 3D objects using progressive meshes [18] for dynamic scene generation.

References

[1] D. Kurlander and D. T. Ling, Planning-Based Control of Interface Animation, *Proc. CHI Conference*, ACM Press, New York, USA, 1995, 472-479.

[2] M. Cavazza, F. Charles and S. Mead, Agents' Interaction in Virtual Storytelling, *Proc. 3rd international workshop on Intelligent Virtual Agents*, Madrid, Spain, 2001, 156-170.

[3] R. Nakatsu, N. Tosa, and T. Ochi, Interactive Movie System with Multi-person Participation and anytime Interaction Capabilities, *Proc. 6th ACM international Conf. on Multimedia: Technologies for Interactive Movies*, Bristol, United Kingdom, 1998, 2-10.

[4] R. Pausch, J. Snoddy, R. Taylor, S. Watson and E. Haseltine, Disney's Aladdin: First Steps Toward Storytelling in Virtual Reality, *Proc. SIGGRAPH '96*, ACM, New York, USA, 1996, 193-203.

[5] A. K. Bansal, T. Kapoor, and R. Pathak, Extending XML for Graph Based Visualization of Complex Objects and Animation Over the Internet, *Proc. 2nd International Conf. on Internet Computing*, Las Vegas, June 2001, 750-756.

[6] S. Oviatt, Talking to Thimble Jellies: Children's Conversational Speech with Animated Characters, *Proc. International Conf. on Spoken Language Processing, ICSLP*, Beijing, China, 2000, 877-880.

[7] B. Simoes and Arvind K. Bansal, Interactive Voice Modifiable 3D Dynamic Object Based Movies over the Internet, *Proc. 5th Intl. Conf. on Internet Computing*, Las Vegas, June 2004, to appear

[8] R. Parent, *Computer animation:algorithms and techniques*, (San Francisco: Morgan Kaufmann Publishers, 2002), 175-203, 328-339, Appendix A.

[9] Virtual Reality Modeling Language (VRML) 2.0, ISO/IEC 14772,*http://www.web3d.org/x3d/specifications/vrml/ISO_IEC_14772-All/part1/concepts.html*, Date accessed: 05/24/2004.

[10] K. Walczak and W. Cellary, X-VRML – XML Based Modeling of Virtual Reality, *Proc. IEEE Symposium on Applications and the Internet, SAINT'02*, Nara City, Nara, Japan, 2002, 204-213.

[11] eXtensible 3D(X3D), *http://www.web3d.org/x3d/overview.html*, Date accessed: 05/24/2004.

[12] Z. Huang, A. Eliens and C. Visser, XSTEP: An XML-based Markup Language for Embodied Agents, *Proc. 16th International Conf. on Computer Animation and Social Agents, (CASA 2003)*, New Brunswick, New Jersey, 2003, 105-110.

[13] S. Kshirsagar, N. Thalmann, A. Vuillème, D. Thalmann, K. Kamyab and E. Mamdani, Avatar Markup Language, *Proc. Workshop on Virtual environments*, Barcelona, Spain, 2002, 169-177.

[14] K. Perlin and A. Goldberg, Improv: A System for Scripting Interactive Actors in Virtual Worlds, *Proc. 23rd annual Conf. on Computer graphics and interactive techniques*, 1996, 205-216.

[15] A. K. Bansal and R. Pathak, Transmitting High Quality Archived Object-based Movies with Reduced Bandwidth Requirement, *Proc. 2nd IASTED International Conf. on Commmunications, Internet, & Information Technology*, Scottsdale, Arizona, USA, November 2003, 553-560.

[16] H. Tanaka, T. Tokunaga, Y. Shinyama, Animated Agents that Understand Natural Language and Perform Actions, *Proc. International Workshop on Lifelike Animated Agents, (LAA)*, Tokyo, Japan, 2002, 89-94.

[17] M. Cavazza and I. Palmer, Natural Language Control of Interactive 3D Animation and Computer Games, *Proc. Virtual Reality*, Vol. 4, 1999, 1-18

[18] H. Hoppe, Progressive Meshes, Proc. 23rd annual Conf. on Computer graphics and interactive techniques, New York, NY, 1996, pp. 99-104