Proceedings of the International Conference on Internet Computing

IC'04

Volume II

Editors: H. R. Arabnia & Olaf Droegehorn
Associate Editors:
Z. Chen, P-T. Chung, H. El-Sayed, M. L. Huang, P. Langendoerfer, J. Lu, X. Meng, Y. Mun, A. Scime

&

Proceedings of the International Symposium on Web Services & Applications

ISWS'04

Editors: H. R. Arabnia & S. Chatterjee
Associate Editor: Eduardo Fernandez-Medina Paton

Las Vegas, Nevada, USA
June 21-24, 2004
©CSREA Press

Interactive Voice Modifiable 3D Dynamic Object Based Movies over the Internet

Bonita Simoes and Arvind K. Bansal
Department of Computer Science
Kent State University, Kent, OH 44242, USA
bsimoes@cs.kent.edu and arvind@cs.kent.edu

Abstract

In this paper, we describe an XML based 3D-Voice Enabled Single Transmission Multiple Display Multimedia Language (3D-VE STMDML) model and its implementation. We describe 3D-scenes, mesh based 3D objects, integration of 2D and 3D objects, object animations with collision avoidance, and voice based animation control. This model has been used to download object based movies over the Internet, dynamically modify the existing movies using voice interaction in real time, and retransmit the modified movies to others over the Internet. The performance analysis shows that voice-based modification to objects in a scene is done in real time.

Keywords: human-computer interaction, Internet, voice interaction, web movies, XML.

1. Introduction

As the Internet gets faster and more interactive, there is more demand for interactive and next-generation 3D characters and 3D movies. Due to this demand, the designing and rendering of realistic 3D interactive characters and movies will grow tremendously. Multimedia information such as textures, sounds, and 3D movies can be easily archived, transmitted, and reused over the Internet.

The combination of a generic Internet based language such as XML, multimedia technologies, Internet, and voice recognition technologies will leverage many applications in the future. For example, children could download and view cartoon movies with their favorite animated characters from different cultures and interact with them to modify their behavior in a make-believe scenario; a crime scene can be modeled using a generic story line that can be dynamically changed to reflect multiple possibilities. It is useful to create simulation environments and recreate certain scenarios.

Many researchers have worked in the areas of voice recognition, animated computer characters, social agents [4], interactive movies [7], story telling [4] and Internet based modeling. However, most 3D modeling systems are domain specific, and use fixed formats. For example, Virtual Reality Modeling Language (VRML) [10] and its successor eXtensible 3D (X3D) (http://www.web3d.org/) have built in nodes, to produce interactivity and animation. In fixed format systems such as VRML and X3D, making use of the existing animations to create new animations in a user-friendly way is fairly complex for non-programmers.

There is a need for a representation that supports the use of voice to create realistic 3D movies over the Internet. There should be a way to make a 3D movie extensible and be able to create new movies based on user commands.

In this paper, we present a new 3D-Voice Enabled Single Transmission Multiple Display model and an XML based language to dynamically interact with and alter 3D object based movies. Figure 1 shows the overall model of an implemented system. The server end consists of the 3D XML representation of a movie. The client end consists of the 3D-VE STMDML parser that parses an XML movie, an animation subsystem (3D graphics engine, a scene compositor and a renderer) that renders the movie and a voice subsystem that parses voice commands. A transformer dynamically transforms an XML script to create a new movie.

The major contributions of this paper are as follows:

1. XML has been extended to represent the modeling and animation of complex graph based 3D objects and surroundings over the Internet in a distributed multimedia environment.

2. A voice-based control mechanism of three-dimensional animated characters has been developed.
3. An XML movie is modified using voice commands to create a dynamically changing Internet transmittable movie, which is recorded, archived, and distributed over the Internet.

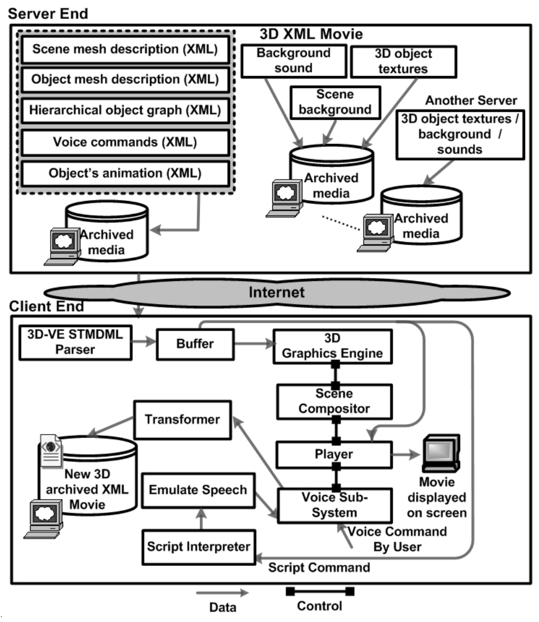

Figure 1. Overall model

This paper describes an XML based language for voice based control of animated 3D objects and 3D scene description in object based movies.

The structure of the rest of the paper is as follows: Section 2 gives some background definitions. Section 3 describes the 3D-VE STMDML model along with a simplified grammar. Section 4 introduces the voice interaction model and parametric animation. Section 5 discusses the architecture and implementation. Section 6 describes the performance evaluation of the system. Section 7 discusses relevant related work. The last section concludes the paper.

2. Background and Definitions

A *scene graph* is a directed acyclic graph that classifies all media objects needed to render 3D scenes in a movie. Each media object or sub-component of an object is attached to a node in a scene graph.

A *hierarchical graph* is a multilayered graph wherein a sub-graph might be embedded inside a node or an edge. Each lower level sub-graph represents a lower level abstraction. An edge between two nodes v_j and v_k represents one or more edges between the embedded sub-graphs corresponding to the nodes v_j and v_k. A media object or a lower level sub-graph is associated with every node or edge in a layer. The graph at the lowest layer containing media objects is referred to as the *object graph*.

A *3D mesh* is a set of faces represented as an ordered set of vertices. Realism is directly proportional to the number of vertices in a mesh.

Hierarchical graphs are used to model animated objects in a modular fashion. Nodes or edges could map to other hierarchical sub-graphs, 2D objects, or 3D objects modeled as meshes. Motion of 3D objects [8] is generated by dynamically changing the position and orientation of nodes and edges of an object graph at different time instances.

Movement of one node can be propagated to neighboring nodes to achieve realistic and fluid movement. The movement of neighboring nodes is achieved by associating weights w_{ij} with the edges (v_i, v_j). The matrix of these weights, denoted as the *animation matrix*, represents the effect that movement of one node has on connected nodes. By multiplying the weights w_{ij} and w_{ik}, the effect of movement of a node v_i on v_k (such that v_i is not directly connected to v_k) is computed, and the vertices of a mesh attached to the corresponding nodes follow along with the movements based on their weights.

3. 3D-VE STMDML Description

This model transmits graph based 3D character movies over the Internet, and controls the movies at the client end. The use of 3D (using meshes) provides more realism to a movie than previous STMD models [1, 2]. Once a 3D object, modeled as a 3D mesh has been sent to a client, the 3D mesh of the object is deformed on the client end using voice commands without any retransmission of any variation of the component. This 3D model is in a true sense, a Single Transmission Multiple Display model. The animation and movement of a 3D object is calculated in real time at the client end and changes are made to the object graph, which in turn deforms the mesh.

3.1 Representing a scene

A movie is characterized by a sequence of scenes. Each scene has multiple interacting media objects or group of media objects. Objects are static or dynamic.

The *background* is modeled as a collection of one or more high-resolution photorealistic 2D images. *Surroundings* consist of 3D immovable objects such as a room (mesh), lighting effects, and sound effects, referred to as static objects. For the efficiency of rendering, a background image is modeled as a matrix of mosaics, a collection of horizontal mosaics, a collection of vertical mosaics, or a combination of all three representations [1, 2]. The images and sounds in a scene are retrieved from different locations. A scene also stores a perspective, [8] which refers to the depth of a scene based on a defined angle. Camera information is stored in a scene as a direction or as a movement and a rotation.

3.2 Representing 3D complex objects

A complex 3D object is divided into smaller components (nodes or sub-graphs). An object is superimposed on a node or an edge. A media object can be of type text, audio, 2D or 3D. A 3D object is reconstructed on the client end using the object graph and related information such as textures, meshes, sounds, and animations. A mesh is modeled using the vertices and faces and is superimposed on the object graph. Each face has (*x, y, z*) coordinates for three vertices, a normal, and a reference of the superimposed material to display the texture. A mesh is associated with a node in an object graph. Nodes in an object graph have weights associated with them.

Figure 2 shows a graph based representation of a Zombie character available for free from *Psionic Design* (http://www.psionic3d.co.uk/). A node can have one or more media objects superimposed on it such as audio and 3D mesh for the face of an object.

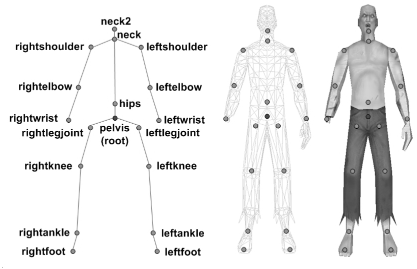

Figure 2. An object graph representation

An object's *animation set* stores time instances of an object's animation. The animation of a node in an object graph could be represented as a *rotation*, a *scaling* (if increase/decrease in size is desired), a *position* (exact coordinates for the object to move to), or a *matrix* which is a 4 x 4 matrix that could be a combination of various transformations.

An object controller associated with an object contains global information about an object such as the movement speed, animation speed, default animation, position of the object in a scene as well as scaling and rotation, if needed.

If an object is an image or text, the screen rendering position is needed. In case of audio objects, the start time and duration is specified.

Figure 3 describes a simplified grammar for the representation language.

```
/*Movie description*/
<movie>::='<movie'<movie_attr>'>'  {<scene>}+ '</movie>'
<movie_attr> ::= <fps> <width> <height> <xpos> <ypos>
               <style>  <num_scenes> <short_description>
<style> ::= 'window' | 'fullscreen'

/*Scene description*/
<scene> ::= '<scene' <scene_attr> '>' <background>
          <resources><perspective><camera>
          <static_mesh>{<objects>}* <voicecommands>
          <script>'</scene>'
<scene_attr> ::= <snumber> <num_objects> <texfname>
               <music> <scene_height>
<background> ::= '<background' <background_id> '>'
                 <background_type><texture_type>
                 <num_images><num_rows>
                 <num_columns> '</background>'
<background_type> ::= '<background_type>' "matrix" |
                      "horizontal" | "vertical" | "hybrid"
                      '</background_type>'
<resources> ::= '<res>' {<location>}+ '</res>'
<location> ::= '<loc>' <uri> <filename> '</loc>'
<perspective> ::= '<perspective' <field_of_view>
                 <aspect_ratio> <z_near> <z_far>'/>'
<camera> ::= '<camera>' <point> | <move> <rotate>
           '</camera>'
<static_mesh> ::= '<smesh' <sname> '>' <vertices>
                 <faces> <materials> '</smesh>'
<materials> ::= '<materials' <num_materials> '>'
               <material_indices>{<features>}+
               '</materials>'
<material_indices> ::=  '<material_indices>'
                      {<material_number>}+
                      '</material_indices>'
<features> ::= '<features' [<materialname>] '>'
                 <facecolor> <power>
                 <specular> <emissive> '</features>'

/*Object description*/
<objects> ::= <object3D> | <image> | <audio> | <text>
<object3D> ::= '<object3D' <objectid><objname> '>'
              <graph>{<dynamic_mesh>}+ <animations>
              <object_controller> '</object3D>'
<graph> ::= '<graph' <graphid> <nodecount> '>'
          <graph_elements>   '</graph>'
<graph_elements> ::= {<node>}+
<node> ::= '<node' <node_name> <transform> '>'
         [<node>] | [<embedded>] '</node>'
<embedded> ::= '<embedded>'{<sub_graph>}* |
              {<objects>}* '</embedded>'
<sub_graph> ::= <graph>
```

```
<dynamic_mesh> ::= '<objmesh' <objmesh_attr> '>'
            <vertices> <faces> <graph_weights>
            [<meshnormals>] [<meshtexturecoords>]
            <materials> '</objmesh>'
<objmesh_attr>::= <objmeshname> <meshnum>
                  <attach_to_graph> <attach_to_node>
                  <max_weights_per_vertex>
                  <max_weights_per_face>
                  <maxedges_affected>
<attach_to_node> ::= <node_name>
<attach_to_graph> ::= <graphid>
<vertices> ::= '<vertices' <ovtotal> '>'{<set_of_vertices>}+
                  '</vertices>'
<faces> ::= '<faces' <oftotal> <sides>  '>'
                        {<set_of_faces>}+  </faces>'
<graph_weights> ::= '<graph_weights' <graphid> '>'
                          {<tnode>}+ '</graph_weights>'
<tnode> ::= '<tnode' <node_name> <numweights> '>'
                  <vertexindices> <weights> <transform>
                '</tnode>'
<meshnormals> ::= '<meshnormals>' <normalvectors>
                   <normalfaces> '</meshnormals>'
<normalvectors> ::= '<normalvectors' <nvtotal> '>'
                   {<set_of_vectors>}+ '</normalvectors>'
<normalfaces> ::= '<normalfaces' <nftotal> <sides> '>'
                        {<set_of_normalfaces>}+
                    '</normalfaces>'
<meshtexturecoords> ::= '<meshtexturecoords'
                  [<texturefilename>] '>'  {<uvcoordinates>}+
                  '</meshtexturecoords>'

/*Animation description*/
<animations> ::= '<animations>' {<animationset>}*
                  '</animations>'
<animationset> ::= '<animationset' <setname> <looping>
                  [<animationspeed>] [<movementspeed>] '
                  <nodeproperties>  {<timeinstance>}+
                  '</animationset>'
<looping> ::= 'true' | 'false'
<nodeproperties> ::= '<nodeproperties>' {<graphnode>}+
                    '</nodeproperties>'
<graphnode> ::= '<graphnode' <node_name>
                  <numinstances> <transform_type> '/>'
<transform_type>::="rotation"| "scale"| "position"| "matrix"
<timeinstance> ::= '<timeinstance' <value> '>' {<node>}+
                  '</timeinstance>'
<objectcontroller> ::= '<objectcontroller>'
      <defaultAnimationSpeed> <defaultMovementSpeed>
      [<defaultAnimation>] <move> <scale> <rotate>
      <rotate_movement> '</objectcontroller>'
<rotate_movement>::= '<rotate_movement' <x><y><z> '/>'

/*Image, audio and text description*/
<image> ::= '<image' <image_id><texture_format> <xpos>
            <ypos> '>' <location> '</image>'
<audio> ::= '<audio' <audio_id> <audio_format>
          <start_time> <duration> '>' <location> '</audio >'
<text> ::= '<text' <text_id> <xpos> <ypos> '>'
        {<font>}* <text_string>  '</text>'

/*Script description*/
<script> ::= '<script>' {<command>}+ '</script>'
<command> ::= '<command>'<characters> '</command>'
```

Figure 3. A simplified grammar

A movie has a predefined script for each scene that is played immediately after it's transmission from a server. During the execution of a script, a client can interrupt the movie by speaking to the characters in a scene. From that point on, the movie records all the activity of the client. The client has an option of transmitting the modified movie script over the Internet.

4. Voice Based Parametric Animation

The interface between the voice subsystem and a movie is illustrated in Figure 4. A 'command-and-control' voice recognition model is used. Each object in a scene has animation sets and an associated object controller. Each *animation set* is an ordered set of *time instances*, at which the object graph (uniquely associated with an object) changes. An animation set consists of multiple associated attributes such as *animation speed*, *movement speed,* and *looping*.

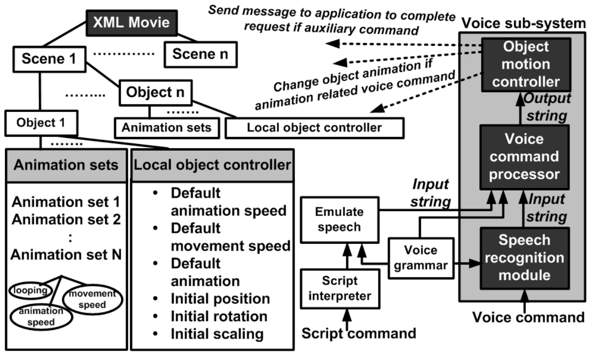

Figure 4. The voice subsystem interface

The speech recognition module takes an input voice command, transforms the voice commands to a textual command denoted as an *input string*, and passes it to the voice command processor. The voice command processor also accepts a text script from an XML movie. A script interpreter passes a text script to the voice command processor via a speech emulator. An *input string* can have a varying number of words and is terminated by a pause above a time threshold. The voice grammar is listed in Figure 5.

There are two types of input commands: *animation* and *auxiliary*. Each of these commands is described by a rule in the voice grammar. An *animation rule* is a quadruple of the form (*person*, *action*, *direction*, *amount*). *Auxiliary* commands are handled by the application.

The voice command processor transforms the *input string* to an *output string*. An *output string* is a sequence of *categorized phrases* that an *object motion controller* understands. A categorized phrase is a *recognized_text* that maps to one of four categories: *object_name, action_set, direction*, and *animation_amount*.

For *animation* commands, the object motion controller sends the *output string* to a *local object controller*, which instructs the object to update its current animation. A rule is activated (or deactivated) by setting the *rule_state* attribute to "active" (or "inactive"). An *object_name* uniquely refers to an animated object. The *input string* might refer to the same object using aliases. Specifying these aliases makes the system user friendly. All object name aliases map to the same *object_name* in the *output string*. An action could be *simple* or *composite*. A simple action refers to one animation set. A composite action is a partially ordered set of actions (composite or simple). Each action in the composite action refers to a separate animation set. The *direction* can be 'left', 'right', 'up' or 'down'. The *amount* specifies the duration of animation rendering.

```
<input_string>::='<commands>'{<rule>}* '</commands>'
<rule>::='<rule' <rule_attr>'>' {<output_string>}*
        {<rule_reference>}* {<optional>}* '</rule>'
<rule_attr> ::= [<id>] <name> [<rule_state>]
<id>  ::= {<digit>}+
<name> ::= "animation" | "auxiliary" | "person" | "action" |
           "direction" | "amount"
<rule_state> ::= "active" | "inactive"
<output_string> ::= '<list>' {<categorized_ phrase>}+
'</list>'
<categorized_ phrase> ::= '<phrase' <category> '>'
                            <recognized_text> '</phrase>'
<category> ::= <object_name> | <action_set> |
                   <direction> | {<animation_amount>}+
<object_name> ::= <characters>
<action_set> ::= <characters>
<direction> ::= "left" | "right" | "up" | "down"
<animation_amount> ::= <number>
<recognized_text> ::= <characters>
<rule_reference> ::= '<rule_reference' <name> '/>'
<optional> ::= '<o>' <rule_reference> '</o>'
```

Figure 5. Voice grammar for the system

5. Architecture and Implementation

The implementation uses C++ with the DirectX API (http://www.microsoft.com/directx) and SAPI (http://www.microsoft.com/speech) for speech recognition. First, the XML movie file is parsed and information about the scene such as objects in a scene and their animations are constructed. This information is stored in separate scene and object databases. A scene is characterized by a quadruple of the form {*scene_id, scene_properties, scene_background, scene_mesh*}. An object is characterized by a 5-tuple of the form {*object_id, object_properties, object_graph, object_mesh, object_animations*}. A typical object animation set is a quadruple of the form {*setname, timeinstance_value, nodes_affected, node_transform*}. Next, the voice command grammar is loaded and the movie player and the speech recognition are launched. An abstraction of the scene rendering algorithm is presented in Figure 6. The movie player is shown in Figure 7.

```
Abstract process: Scene Rendering
Input: An XML movie file;
parse the XML movie to build scene, object and animation
information;
load the voice command grammar;
launch the 3D-VE STMDML player at the location and with
dimensions specified in XML;
launch the speech recognition thread and open the speech
engine listening box;
if XML movie contains text script yet to be played {
   disable microphone; load script commands in memory; }
else  enable microphone;
while true do {   //rendering loop
   z-buffer = false;
   draw the 2D background;
   if (image or text objects exist) {
      draw image and text object;}
   z-buffer = true;
   draw the 3D surrounding scene mesh;
   for (each object in a scene) {
      if (Move-State[Object]  || Anim-possible[Object] )
         {draw the object's animation based on  state;}
      else { draw the object's default animation;}
   }
   if ((user interrupts script) || (end of script commands))
      { enable microphone; }
   VoiceRecognition updates action, direction and
      amount  of the object;
      Anim-possible[Object] = true;
      Move-State[Object] =  true;
   if (save) create new XML movie with interaction saved;
   if (exit)  free memory used; exit the system;
}
```

Figure 6. A scene rendering process

Inside a rendering loop, the Z-buffer [8] is disabled while drawing the 2D background, image, and text objects, and enabled while drawing scene and object meshes. This saves unnecessary computation. The animation state, *Anim-possible* and the movement state, *Move-State* of objects are set based on the recognition and validation of voice commands as well as on collision detection between 3D objects, 3D objects and scene mesh, and 3D objects and viewing frustum [8].

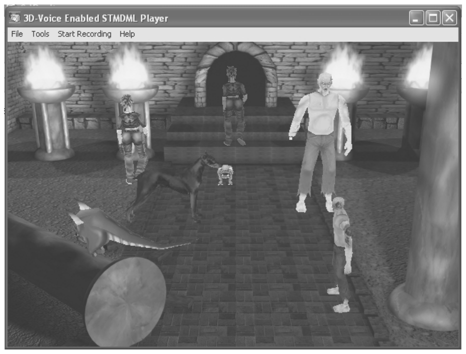
3D-Voice Enabled STMDML Player
File Tools Start Recording Help

Figure 7. A 3D-VE STMDML player interface

6. Performance Evaluation

Various experiments were conducted to study the response time as the command complexity and scene complexity were increased. Figure 8 describes the response time vs. number of heterogeneous voice commands issued. Figure 9 shows the number of complex objects vs. the initial rendering response time. The numbers are an average of a total of ten runs for each movie. Figure 10 shows the file size vs. number of mesh faces in an XML movie.

Ambiguity is present due to multi-word phrases and optional rules (caused by parameterization) in the voice grammar. The speech engine needs more time (see the top line graph in Figure 8) to build ambiguous sentences such as "*Zombie walk left twenty*" because the speech engine waits to determine if the *action* parameter is "*twenty*" or if the user will go on to say "*twenty five*".

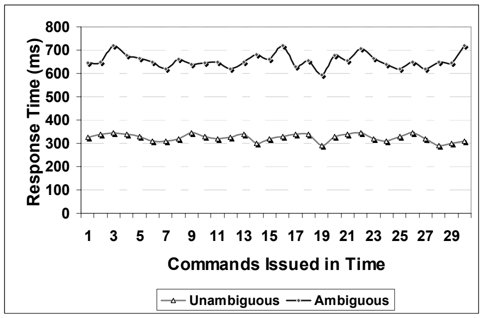

Figure 8. Response time vs. voice commands

There is an initial rendering time delay caused due to the initial setup of object states and the building up of the object and scene database (see Figure 9). The initial delay increases with the number of objects in the movie. After the initial delay, the time taken to update the state of an object and time to check for collision detection of objects was around 16ms.

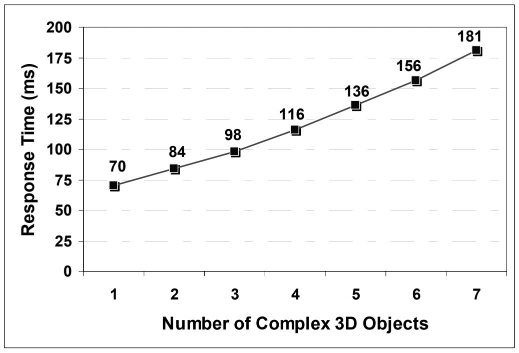

Figure 9. Rendering time vs. 3D objects

As the number of faces increase, the file size of an XML movie also increases resulting in larger response time for parsing as shown in Figure 10. Parsing a movie file with one object consisting of 238 faces takes about 1297ms, and creating the object database takes about 1356ms, for a total of 2653ms. The time increases linearly as the mesh complexity increases.

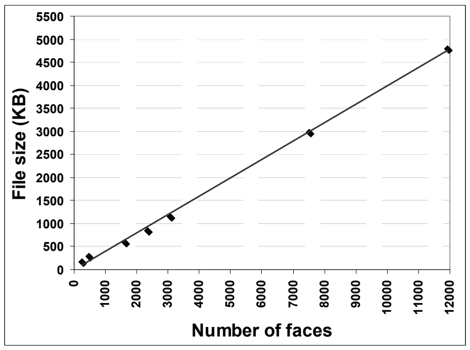

Figure 10. Movie size vs. number of faces

7. Related Work

VRML [10] is a 3D scene description language to model 3D animations. X-VRML [12] extends VRML to include variables, statements, expressions, and loops to dynamically generate 3D scenes.

XSTEP [6] uses Prolog like parameterization for easy interaction with agents. Voice based animation of 3D content is not incorporated in XSTEP.

VoiceXML [11] uses speech recognition and synthesis to drive telephone based systems. This does not have the concept of a movie, dynamic modification of scripts, or parametric voice based interaction.

Improv [9] uses high level scripts that allows an animator to create predefined rules to interactively control the behavior of virtual actors. These rules and scripts cannot be altered dynamically. Other systems use natural language for interactive 3D animation in computer games.

KidsRoom [3] is a multimodal interactive bedroom storytelling system. Children get into an imaginative narrative world by navigating a story following instructions given to them by a virtual character. During narration, children interact with virtual characters projected onto walls and with objects within the room. It allows behaviors to be undertaken only at certain points so children could not interact with the system at any time to change the plot of the story. This is done in a fixed setting environment made exclusively for this purpose.

To the best of our knowledge, none of the current systems use XML to control animations or support voice controlled dynamic modification of movies.

8. Conclusion

This paper introduces the concept of Internet based 3D voice enabled XML movies that are dynamically modified using voice based commands and retransmitted over the Internet. A movie based on this language describes scenes, 2D and 3D objects in each scene, 3D objects' animation, as well as interaction in a scene. 3D objects are static or dynamic. Dynamic 3D objects are represented as hierarchical graphs within a movie. A list of parameterized voice commands is accepted as part of a movie. A grammar has been developed to include voice based interaction, mesh based representation and parameterized motion of 3D objects.

Using this model, a user can realistically bring about a persistent twist to a movie's predefined story and its ending. This language model is continuously evolving. Currently, we are developing morph enabled 3D objects with progressive meshes [5] for dynamically changing scenes.

References

[1] A. K. Bansal and R. Pathak, "A Server Coordinated Predictive Buffer Management Scheme to Reduce Bandwidth Requirement for Synthetic Multimedia Movies over the Internet", *Proc. 4th International Conf. on Internet Computing*, Las Vegas, June 2003, Vol. II, pp. 831-837.

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

[3] A, Bobick, S. Intille, J. Davis, F. Baird, C. Pinhanez, L. Campbell, Y. Ivanov, A. Schutte, A. Wilson, "The KidsRoom: A Perceptually-Based Interactive and Immersive Story Environment", *Proc. PRESENCE: Teleoperators and Virtual Environments,* 8(4), 1999, pp. 369-393

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

[5] H. Hoppe, "Progressive Meshes", *Proc. 23rd annual Conf. on Computer graphics and interactive techniques*, New York, NY, 1996, pp. 99-108.

[6] 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, 2003, pp. 105-110.

[7] 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, UK, 1998, pp. 2-10.

[8] R. Parent, *Computer animation: algorithms and techniques*, Morgan Kaufmann Publishers, San Francisco, 2002, pp. 175-200.

[9] 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*, New York, 1996, pp. 205-216.

[10] Virtual Reality Modeling Language(VRML) 2.0, ISO/IEC 14772, *http://www.web3d.org/x3d/specifications/vrml/ISO_IEC_14772-All/index.html*

[11] VoiceXML, *http://www.voicexml.org/*

[12] 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, pp. 204-213.