A Software Framework to Create 3D Browser-Based Speech Enabled Applications

Ednaldo Brigante Pizzolato [1]
Diego Daniel Duarte [1]
Marcio Merino Fernandes [1]

Abstract: The advances in automatic speech recognition have pushed the human-computer interface researchers to adopt speech as one mean of input data. It is natural to humans, and complements very well other input interfaces. However, integrating an automatic speech recognizer into a complex system (such as a 3D visualization system or a Virtual Reality system) can be a difficult and time consuming task. In this paper we present our approach to the problem, a software framework requiring minimum additional coding from the application developer. The framework combines voice commands with existing interaction code, automating the task of creating a new speech grammar (to be used by the recognizer). A new listener component for the Xj3D was created, which makes transparent to the user the integration between the 3D browser and the recognizer. We believe this is a desirable feature for virtual reality system developers, and also to be used as a rapid prototyping tool when experimenting with speech technology.

1 Introduction

In general terms, virtual reality (VR) can be defined as an advanced user interface designed to visualize, moving around, and interact within computer generated 3D spaces [12]. Sophisticated VR applications are now common place in some sectors, such as medicine, advanced training, simulation, and industrial design. Those systems often employ expensive special purpose devices for interaction between users and the virtual world. On the other hand, a basic VR system can be designed with standard components, standing at the core of those a 3D virtual scene, and a browser with visualization and interaction capabilities. Virtual scenes are often designed using VRML (Virtual Reality Modeling Language), or its successor, X3D [7]. Interaction through browsers are typically done by direct manipulation, using a mouse or some other device. Although relatively simple, this setting can be useful for many application areas, and also as a prototyping tool for more advanced systems.

One limitation of the basic system just described is its intrinsic mono modal interaction

[1]Universidade Federal de São Carlos
Departamento de Computação
São Carlos - SP - Brasil
{ednaldo@dc.ufscar.br, diego@dc.ufscar.br, marcio@dc.ufscar.br}

nature. In other words, the user is limited to a single input modality, which in many cases diverts from the original aim of having interaction with the virtual world in a way as natural as possible. The alternative to that is to provide multimodal interaction capabilities, i.e., multiple input modalities. Probably the first working implementation of this approach was the so-called "Put That There" system, which processed speech in parallel with touch-pad pointing [4]. Since then, many multimodal systems have been developed [24], and is still a very active research area.

Among the input modalities employed to build multimodal interfaces are speech, natural language, hand gestures, hand postures, and body gestures. *Speech* is the most natural way to interact among humans, and for this reason it is a very desirable way to interact with a computer system. Some of the advantages of this approach are:

- It provides the user a more natural feeling during the interaction experience;
- It may allow alternative choices on how to interact with the virtual world;
- It enables "hands-free", "eyes-free", or "hands-busy" interaction;
- It enables users to refer to objects not present in the current view of the virtual world;

So, it is no surprise that many systems provide speech recognition capabilities as an interaction modality [22, 3, 15, 11]. However, speech recognition is a rather complicated task by itself. Integrating it into another complex system such as a VR can be a burden to the entire project, which may prevent it in many cases. Thus, in this paper we propose a software framework aiming to alleviate this problem. The approach assumes the adoption of an independent speech recognizer (a common decision in this area), which then requires the VRML/X3D browser to be able to communicate with it. Therefore, our first step was to implement a component inside an Xj3D browser [8] able to listen to the recognizer and understand its codes. However, this solution implies a tight interdependence between the recognizer, the listener component, and the 3D application. Our proposal is to hide, as much as possible, the speech implementation from the VRML/X3D programmer, keeping the additional programming to a minimum. The key for the simplicity is to assign voice commands to mouse events making the virtual world to behave as if the mouse was clicked. The scripts created to function when the mouse is clicked will also function when a specific voice command is given. The remainder of this paper is organized as follows: Section 2 presents some related works that are representative of the main issues involved, followed by a brief presentation about speech recognition in Section 3. Then Section 4 describes the core of the work developed, followed by the conclusions in Section 5.

2 Related Work

Researchers have been comparing the use of voice-activated commands against conventional interface devices (i.e. keyboard and mouse) since early 1980's. The work described in [27] compares the speed and accuracy of speech input versus typed input of commands in a simulated military application over a distributed network. In that work a recognition accuracy of 96.8% was achieved when using voice input, and on average, that was 17.5% faster than typed input,with less errors. However, other studies [20, 13] showed that keyboard inputs could be faster than voice inputs (although more error prone). Later on, Karl compared mouse and voice inputs as an interface for a word processor application [17] . He showed that, on average, the task time was 18.67% quicker when speech interface was applied. In another direction, Pausch and Leatherby [25] investigated the improvement voice could bring to systems where mouse and keyboard could also be employed. In general, they concluded that the task completion time could be reduced over 10% when speech and mouse were employed together. On the other hand, Hauptmann pointed out that users may had lost their attention while waiting several seconds for the system's response and assumed that in the future speech could perform much better [14]. More recently, Pizzolato [26] and Christian [6] investigated the use of voice for the human-computer interaction. Both identified that the performance is related to the task. Pizzolato showed that continuous speech can improve the task completion time when compared to isolated speech commands or to mouse commands. Christian showed that voice control adds approximately 50% to the performance time for certain types of tasks.

The use of speech interfaces with 3D visualization systems is the topic of many research works, either as a conceptual approach, or concerning practical implementation issues, such as in [21]. Some experiences to apply multimodal and spoken language interfaces to a number of *electronic reality* applications are described in [5]. A bimodal, speech/gesture interface is used to control a 3D display in a virtual environment for structural biology analysis [28]. The integration of speech and other forms of multimodal interaction to the VTK visualization system is described in [18].

The complexity of the speech and natural language input modalities makes it very difficult to build them as an integrated part of a visualization system. Instead, they are typically independent components responsible for acquiring voice commands, interpreting them, and emitting a message to interact with the system. For this reason, a number of *software frameworks* have been proposed in order to increase the reusability of existing code and integration mechanisms. A general framework for building integrated natural-language and multimodal dialog systems is presented in [16]. The framework is based on a distributed component model that allows reuse and extension of existing software modules, as it was demonstrated in the development of the so-called Multiplatform framework. A multimodal interface for navigating in arbitrary virtual VRML worlds is described in [1], an approach employing a context free grammar to control the communication between individual components of the system. In another work a modular approach for integrating multiple input/output modalities in virtual environment is described [2]. The proposed system architecture was applied to case study dealing with molecular visualization, modeling and fitting, showing good results. All those three works are well developed frameworks aimming to facilitate the integration of multiple input modalities into general purpose or virtual reality systems. However, none of those is particularly concerned with the issue aimed by our work, which is to provide developers of 3D interactive applications with a way to integrate a speech interface into their systems with minimum specific knowledge and effort.

In a more general context of using voice input with browser applications, there are some standardization efforts aiming to improve portability and development time. Possibly the most accepted one is VoiceXML, an XML format for specifying interactive voice dialogues [10]. A related standard, called *Speech Recognition Grammar Specification* defines a syntax for representing dialogue grammars so that developers of systems using speech recognition can specify the words and patterns of words to be listened by the recognizer engines [9].

3 Speech Recognition

Speech recognition by machine is the process computers apply to convert speech signals into written information [22]. Researchers have been investigating how to make machines to recognize speech and respond accordingly to it for quite some time. At first glance it was supposed not to be a difficult task. Time showed how difficult the task was. It took almost 40 years of intense researches to bring speech recognition systems to the shelves. And it still has its limitations. Human speech can vary in frequency and time (women has a pattern of voice that is different from men) and the same phonemes can be longer for one person and shorter for another one. Besides, the acoustic evidences of a sound may suggest the utterance of few different phonemes. As the problem is more severe with continuous speech uttered by different people, researchers tried for quite some time to recognize isolated phonemes (and then, words) of one speaker. Then isolated words where tried. Speech recognition by machine had a huge breakthrough when Kai-Fu Lee [19] proposed in 1988 a large vocabulary (1,000 words) speaker independent (many speakers) continuous speech recognizer with excellent performance. The solution (based on Hidden Markov Models) "learned how to guess" (based on statistics) what phonemes were more likely to have been spoken and aided by a lexicon and a grammar could point the most likely phrase a speaker said.

A simplified representation of a generic speech recognizer can be seen in Figure 1. The acoustic models are similar to "hash codes" of the acoustic signals. However, they also statistically represent the variations of an acoustic signal across a large number of speakers. The pronunciation lexicon is the dictionary, i.e., how to connect acoustic signals in order to get a word. An example of a pronunciation lexicon entry is $economics \rightarrow Ek * nAmIks$.

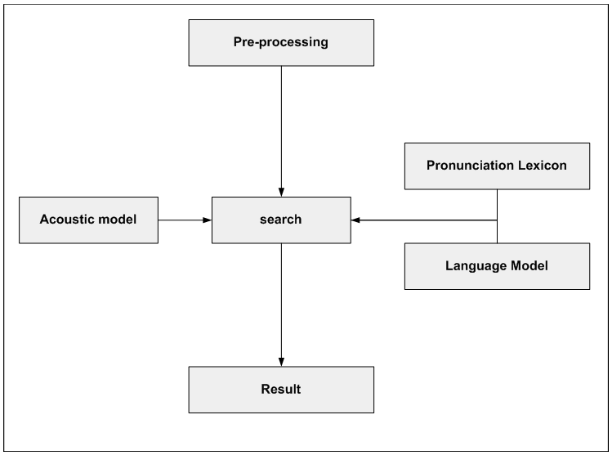

Figure 1. A schematic view of a speech recognizer

The ideal speech recognition system would be able to recognize any utterances from any speaker, at any environment conditions. But, this is too difficult. Therefore, in order to improve recognition performance, companies decided to simplify the process by reducing the complexity of the system in order to have commercial products. The dimensions they can tackle are:

- Speaker (how many speakers the system is able to cope with);
- Noise (how robust the system is to environment noise);
- Size of the vocabulary (how many words to recognize);
- Type of the lexicon (phonemes or words);
- Type of the speech (continuous speech or isolated word).

The first dimension indicates how well the recognizer behaves for a diversity of speakers. Speaker-dependent systems recognize well for a specific speaker. Commercial speech recognizers cannot target this dimension directly as the systems should be sold to many people. Indirectly, this dimension can be reduced if speaker adaptation is used (the system adapts its internal models to improve recognition for a particular user). In fact, a good dimension to focus on is the size of the vocabulary. As the number of words increases the recognition accuracy decreases. Keeping this dimension down is necessary to obtain good results. This is not a reasonable task when the system has to recognize a speech having no clue what so ever about the subject of the speech (or conversation). However, this is perfectly possible in a controlled conversation. By "controlled conversation" we mean a conversation were the computer controls the flow of the dialog. An example would be a purchase of some piece of clothe. Imagine a human-computer dialogue in a clothes store:

- Computer: What would you like to see?
  expecting shirts, t-shirts, paints
- Person : A t-shirt, please!
- Computer: What size?
  expecting small, medium, large, very large
- Person: Medium!
- Computer: Do you have any color in mind?
  expecting yes/no or colors
- Person: I would like red!

Given the context, this is a normal conversation that any commercial speech recognition system would have with a person. Actually, it is the context that keeps the number of words down (controlling one dimension of the problem). As this dimension is controlled, this means a hint should be given to the system at any time the recognition is requested. Companies call this a "speech grammar". It is a set of words that are expected to be uttered by a person (at some time of the conversation) in order the computer recognizes the command. Two different systems may have different "speech grammar" syntaxes.

4 Overall Software Framework

As already mentioned, the goal of this work is to simplify the task of integrating a speech recognizer into a 3D browser running a VRML or X3D model. That is accomplished by means of a software framework providing the following functionality:

- Reuse grammar templates for system-defined recognition operations;
- Automatic generation of grammar files for user-defined recognition operations;
- Automatic boot of speech recognizer based on defined grammars;
- Automatic integration of speech recognizer with interactive 3D world;

- Set-up and run of integrated system;

As shown in Figure 2, the framework integrates its three main components, i.e., the virtual reality model, the 3D browser and the speech recognizer into a single execution environment. The voice interaction among them is accomplished by means of automatic generation of grammar files, which drives the recogninition engine. Those are generated based on user defined commands found in the source X3D file, and also on pre-defined browser commands. Given a compiled grammar and a voice input, a speech engine gives back its interpretation of the utterance.

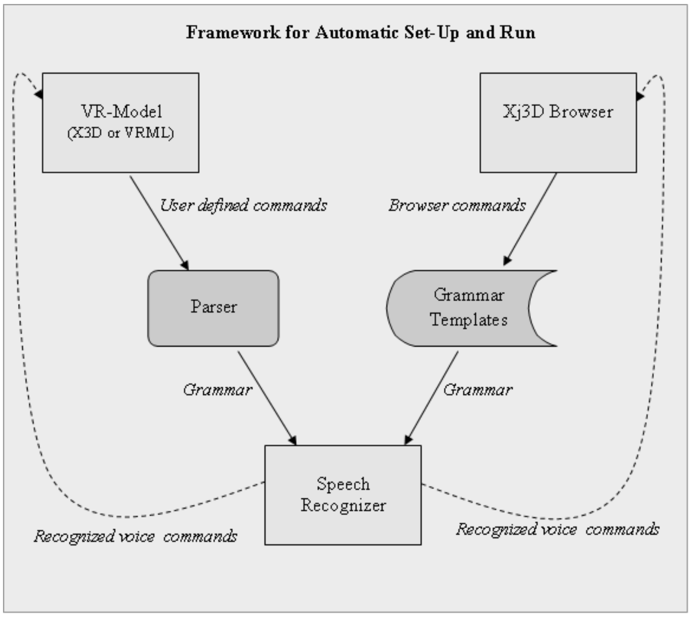

Figure 2. Overview of the proposed framework

The communication between the speech engine and the Xj3D browser is possible by including a listener component inside the browser (see Section 4.1), which was specially developed for this work. By using it, the browser can receive a command from the speech engine, checks it against the list of valid commands, and processes it with a possible modification of the virtual scene. It is also possible to add some pre-recorded voice information to guide users, telling them what command was understood, or that no valid command was recognized. A schematic view on the process of creating a grammar based on hidden text commands, and its integration to a speech recognizer and 3D browser is shown in Figure 3.

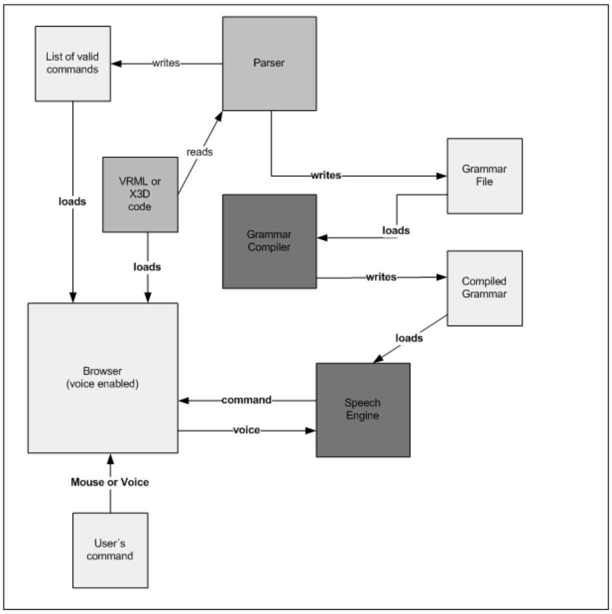

Figure 3. Integration of speech engine and browser

In order to integrate the voice interface in the whole system, the speech recognizer needs to be up and running. Standard pre-requisites include a license manager, and most important, a compiled grammar defining all possible valid voice commands and sentences expected from the user. To produce a compiled grammar, the grammar compiler needs a valid license as well. Therefore, in a Windows environment a batch file should be built in order to start the programs in a correct order, including the parser that will generate the user-defined grammar. The whole process would be similar in a Unix/Linux environment.

4.1 A new listener component

An important step to build the software framework was to implement a component (called listener) inside the Xj3D browser in order to listen to the recognizer and understand the internal codes emited by it. The new component architecture of the browser is shown in Figure 4.

The listener (inside the browser) waits for a voice signal coming from the speaker. Then, it sends this signal to the speech recognition engine and waits for its response. In order to produce the response, the engine needs to consult the compiled grammar that was prepared by the parser. When the command returns from the speech engine, then the listener verifies if it is a valid one (looking at the list of valid commands previously built by the parser). This list is loaded by the listener when the browser starts. It is worth mentioning that this list is based on commands inserted by the programmer into the VRML/X3D code. If the speaker says something wrong, then the recognizer would send back to the listener a specific code informing that it was not possible to recognize that voice input. If it is a valid command, then an event is generated. The type of event to be generated also comes from this list and it is chosen by the programmer. For instance, it can be a touchTime event (when the current time is thrown). The listener knows the correct object to send the event to. Once the object receives it, it is as if the mouse was clicked. From the programming viewpoint, it is important that the listener inherits properties and characteristics from the Event Model Evaluation component. It is this inheritance that will make scene change possible. The code fragment bellow shows how to implement the integration of the voice listener during the browser initialization steps.

```
public class BrowserTest extends JFrame {
  static final long serialVersionUID = 0x46464646;
  VoiceListener vl;

  public BrowserTest() {
    HashMap<String, Object> parameters = new HashMap<String, Object>();
    parameters.put("Xj3D_ShowConsole",Boolean.FALSE);

    System.setProperty("x3d.sai.factory.class",
                      "org.web3d.ogl.browser.X3DOGLBrowserFactoryImpl");
    X3DComponent x3dComp = BrowserFactory.createX3DComponent(parameters);

    Browser browser = x3dComp.getBrowser();
    X3DScene scene = browser.createX3DFromURL(new String[] { "sample.x3dv" });
    browser.replaceWorld(scene);

    this.getContentPane().setLayout(new BorderLayout());
    this.getContentPane().add((JComponent)x3dComp.getImplementation(),
                              BorderLayout.CENTER);
    vl = new VoiceListener(scene);
    vl.start();

    this.addWindowListener(new WindowListener() {
      public void windowActivated(WindowEvent e) {}

      public void windowClosed(WindowEvent e) {
        vl.stop(); System.exit(1);
      }
      public void windowClosing(WindowEvent e) {}
      public void windowDeactivated(WindowEvent e) {}
      public void windowDeiconified(WindowEvent e) {}
      public void windowIconified(WindowEvent e) {}
      public void windowOpened(WindowEvent e) {}
    });
    this.setSize(640, 480);  this.setVisible(true);
  }

  public static void main(String[] args) {
    new BrowserTest();
  }
}
```

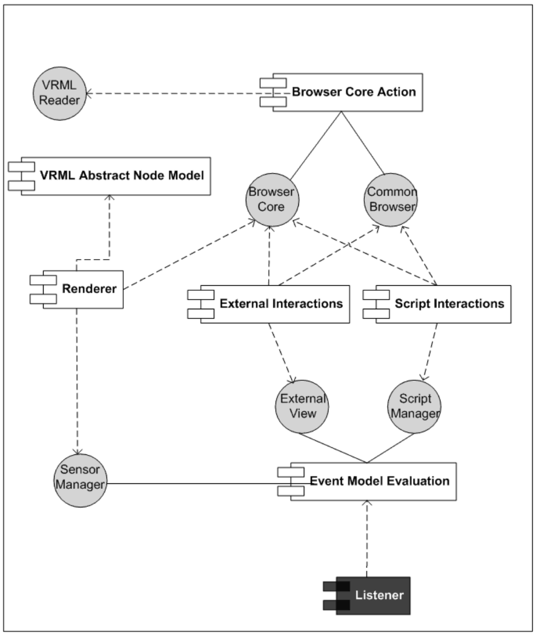

Figure 4. Component view of an Xj3D browser with a voice listener

4.2 Grammar Files

In order to build a prototype system, we have adopted the Nuance Speech Recognizer (version 8.0), a commercial product [23]. That was mainly due to its robustness, and availability of several language databases, in particular Brazilian Portuguese, as this is the language used for most of our users. Although not complicated, the standard process of creating a grammar to be recognized by this system is essentially manual, requiring recompilation for every change made to a grammar file. A grammar file in the Nuance specifies sets of valid user utterances to be used at particular points in the application. For example, in a simple application the user may choose a color to paint a 3D Cube, selecting it from a set of predefined options. In the Nuance grammar file, those options would be defined, along with possible variations of valid spoken sentences. The speech-recognition engine uses the grammar to identify which option the user is selecting, emmiting the corresponding internal command to be used by the application system. An example of such a system is shown in the following code fragment:

```
;<Encoding=1252>
;GSL2.0 encoding="1252"

.Choice ([
(paint cube) {<command_said "v4">}
(choose red) {<command_said "v1">}
(choose green) {<command_said "v2">}
(choose blue) {<command_said "v3">}
])
```

4.3 Browser Level Support

The first level of voice interface capabilities provided by the framework is included in the Xj3D browser itself (Figure 5). That is done for the standard browser commands, which can be operated by voice based on a pre-defined template that will be used as a grammar file by the recognition engine. In the prototype system implemented as part of this work, grammar templates were provided only for a subset of the standard commands, although it could be easily extended for the whole set of commands present in the browser. The provided templates were basically concerned with navigation functions, including support to select mode and type of operation, respectively. The speed of movements is originally set at 5 units per frame, but that can be changed either using mouse clicks, or voice commands. A summary of them can be seen in Table 1.

Usually, the user needs to click on one icon (in the bottom line) to choose a specific control (fly, for instance). Then, moving the mouse in any direction will produce a different view of the scenario. The same results can be obtained by using voice. The user could say "selecting walking mode" and then "go forward" instead of clicking at the icon and then moving the mouse forward.

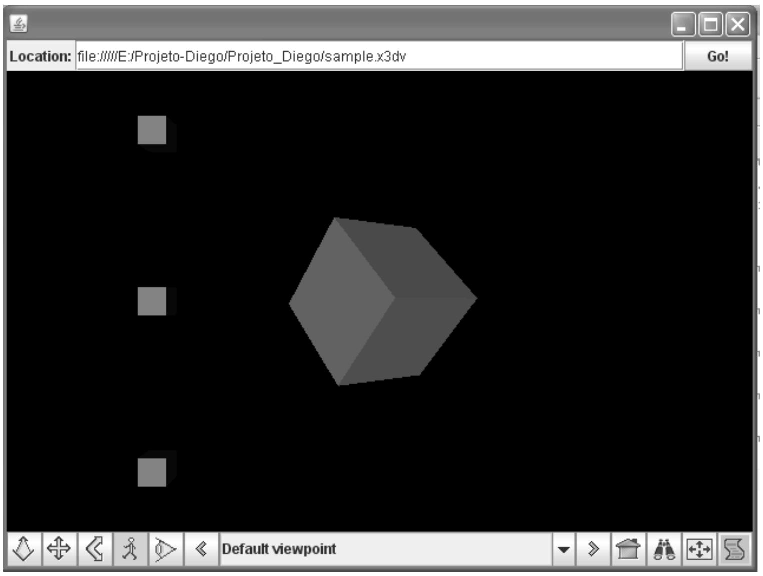
Location: file:////E:/Projeto-Diego/Projeto_Diego/sample.x3dv
Go!
Default viewpoint

Figure 5. Xj3D browser with a voice interface for standard commands

Voice Enabled Standard Browser Commands	
Navigation Modes:	
Observe	
Walk	
Fly	
Navigation Functions:	
Forward	Backward
Right	Left
Up	Down

Table 1. List of voice enabled standard browser commands

4.4 Application Level Support

In principle a VRML/X3D programmer will create a program to work as if there was no voice commands. But, he/she will also provide hidden commands that will be useful to allow the modified Xj3D browser to listen (and respond) to voice commands. All of this hidden commands can then be processed by the parser that will interpret all the hidden commands creating a file with a list of valid voice commands. This list (or lookup table) will be used by the browser to trigger a specific script when a valid command is uttered. The parser is also responsible to create a grammar file that will be used for the speech engine. Usually this file needs to be compiled to be further used by the speech engine. Therefore, the programmer only needs to create a command (like "select the sphere") to work as if the mouse was clicked on the sphere and put it in the right place at the code.

In order to illustrate how it works, lets imagine a virtual world with a single cube (Figure 5). The only action this cube can perform (to keep it simple) is to change its color (but we could move it around, enlarge it, etc.). To accomplish it only by using a mouse, the programmer would have to create small cubess with different colors and would have to make them sensors (see the small "boxes" at figure 5). Then, one would have to click on one of these color boxes to choose a desired color and then would have to click on the object in order to assign the color to the object (to generalize, as it may be possible to have more than one object in the scene).

To perform the same task using a speech interface, one would have to say "choose red" and then "Paint cube", i.e., two spoken commands to replace the two clicks in the mouse. In order to do that, a VRML programmer (X3D is very similar) would have to insert a few hidden commands inside the code (as comments between #@ and @#), as shown in the example bellow:

```
children
[
    DEF ObjectSensor TouchSensor
    {
        #@ touchTime: "Paint cube"
           command applyColor @#
    }
    Shape
    {
       appearance Appearance
       {
          material DEF Object Material
          {
             diffuseColor 1 0 0
          }
       }
       geometry Box { }
    }
]
```

Similarly, for each little box with a different color (the sensor boxes) the programmer would have to add a hidden command (also between #@ and @#) like the following one:

```
children
[
   .........
   DEF RedSensor TouchSensor
   {
       #@ touchTime: "Choose red"
           command setRed @#
   }
   .........
]
```

With the new #@ touchTime... @# entry inside the RedSensor, the parser can create i) a valid entry for the "speech grammar" file and ii) a valid entry for the list of valid commands. The list of valid commands (used by the proposed new component) is important as it creates a relationship between sensors and voiced commands. The parser creates this relationship as it knows the position of the entry in the VRML code. In the example, "choose red" is the voice command for the RedSensor that act as a touch sensor. SetRed is the command the speech recognizer will send back to the browser if this utterance is recognized and then, the proper actions will be performed by the browser when the listener throws the correct event. As previously discussed, the grammar (used by the speech recognizer) is also very important as it reduces the amount of words a system has to recognize at a given moment. As an example, a valid grammar file for the example would be:

```
Command
[
  (choose red)   {<command "setRed">}
  (choose blue)  {<command "setBlue">}
  (choose green) {<command "setGreen">}
  (paint cube)   {<command "applyColor">}
]
```

Finally, the original ChangeColor script written for the mouse (without considering speech recognition) should be kept intact, as shown bellow:

```
DEF ChangeCor Script {
  inputOnly SFTime setRed
  inputOnly SFTime setGreen
  inputOnly SFTime setBlue
  inputOnly SFTime applyColor
  outputOnly SFColor setColor
  initializeOnly SFColor color 1.0 0.0 0.0

   url ["ecmascript:
     function setRed(value, timestamp) {
   color[0] = 1;
   color[1] = 0;
   color[2] = 0;
}

function setGreen(value, timestamp) {
   color[0] = 0;
   color[1] = 1;
   color[2] = 0;
}

function setBlue(value, timestamp) {
   color[0] = 0;
   color[1] = 0;
   color[2] = 1;
}

function applyColor(value, timestamp) {
   setColor = color;
}

 "]
}
ROUTE RedSensor.touchTime
   TO ChangeColor.setRed
ROUTE BlueSensor.touchTime
   TO ChangeColor.setBlue
ROUTE GreenSensor.touchTime
   TO ChangeColor.setGreen

ROUTE ObjectSensor.touchTime
   TO ChangeColor.applyColor
ROUTE ChangeColor.setColor
   TO Object.diffuseColor
```

One can see the advantages of using speech by creating a simple scene modification (see figure 6). If the user walks around in order the main cube hides one of the sensors, then, at that particular position, it is impossible to change the cube's color with the hidden color

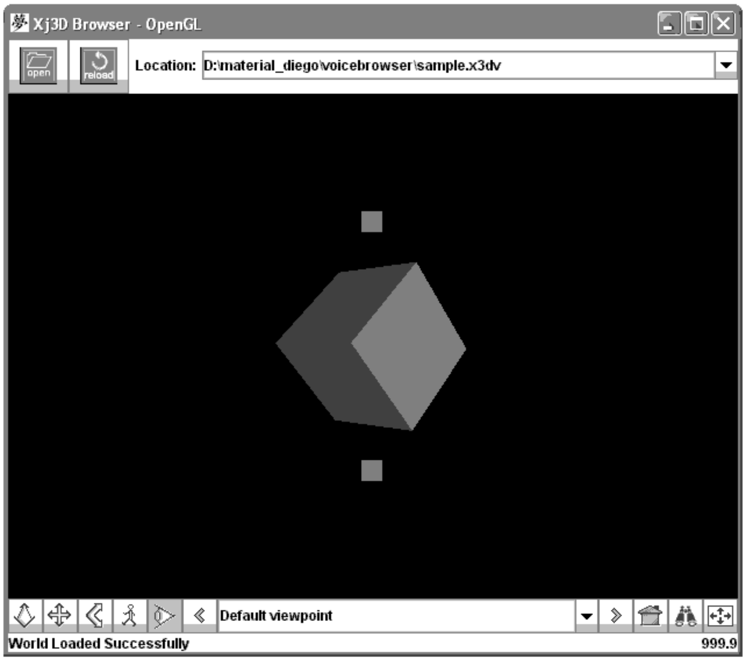
Xj3D Browser - OpenGL
open
reload
Location: D:\material_diego\voicebrowser\sample.x3dv
Default viewpoint
World Loaded Successfully
999.9

Figure 6. Hidden sensor can not be reached by mouse click

only by clicking the mouse. At this very same position, it is possible to paint the cube with any color using voice commands (the sensors are there and can be reached by voice).

The scene with only one cube is very simple. A more complex scene can be created with more objects (like in figure 7). In this case, one would only have to add the voice commands to select each object (as the other commands we have already shown). One could replace spheres, cubes, cones and cylinders by cars and transform it into a commercial application (some companies like Nissan or Fiat have similar applications at their sites). This framework would allow a programmer to easily add voice commands into such environments and the user would be able to pick a car model and then change its color using either speech or mouse.

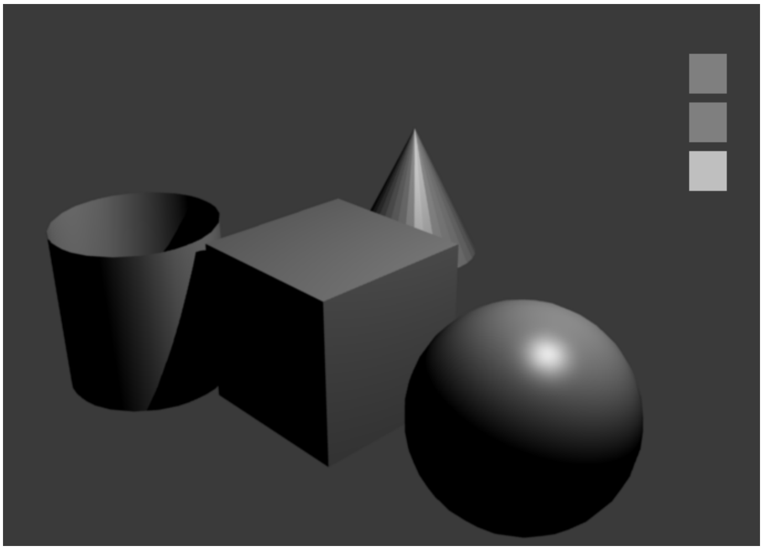

Figure 7. A virtual environment with more objects

5 Conclusions

We have proposed a new browser architecture (based on the Xj3D) for VRML and X3D programs in order to make programmer's life a bit easier when he/she decides to have a voiced-enabled virtual world. The simple commands are hidden in the VRML/X3D code so as it is compatible with the current browser version being used. When the programmer decides to use our browser, then he/she can control the scenario or modify the scenario's objects using voice commands. Along with a new listener component, a parser has also been developed in order to identify the hidden commands and create the necessary configuration files for the system (speech engine and browser). We have successfully tested our proposal for some simple 3D virtual worlds. From the object selection viewpoint we are currently investigating more complex environments (similar objects in the same scene). We would like to introduce some intelligence into the whole process in order the end user to choose an object by simply saying "the sphere on the left corner" or "the red cube" and so on.

References

[1] Frank Althoff, Herbert Stocker, Gregor McGlaun, and Manfred K. Lang. A generic approach for interfacing vrml browsers to various input devices and creating customizable

3d applications. In *Proceedings of Web3D 2002 - 7th International Conference on 3D Web Technology*, pages 67–74, Tempe, Arizona, USA, 2002.

[2] Rajarathinam Arangarasan and George N. Phillips Jr. Modular approach of multimodal integration in a virtual environment. In *Proceedings of ICMI m2002 - 4th IEEE International Conference on Multimodal Interfaces*, pages 331–336, 2002.

[3] D Bitouk and S. K. Nayar. Creating a speech enabled avatar from a single photograph. In *VR '08: Proceedings of the Virtual Reality Conference, 2008*, pages 107–110, 2008.

[4] Richard A. Bolt. "Put-That-There": Voice and gesture at the graphics interface. In *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 262–270, New York, NY, USA, 1980. ACM.

[5] A. Cheyer and L. Julia. Spoken language and multimodal applications for electronic realities. *Virtual Reality*, 4(2):114–128, 1999.

[6] Kevin Christian, Bill Kules, Ben Shneiderman, and Adel Youssef. A comparison of voice controlled and mouse controlled web browsing. In *Assets '00: Proceedings of the fourth international ACM conference on Assistive technologies*, pages 72–79, New York, NY, USA, 2000. ACM.

[7] Web3D Consortium. X3d & related specifications. http://www.web3d.org/x3d, 2007.

[8] Web3D Consortium. Xj3d - java based x3d toolkit and x3d browser. http://www.web3d.org/x3d/xj3d/, 2007.

[9] World Wide Web Consortium. Speech recognition grammar specification version 1.0. http://www.w3.org/TR/speech-grammar/, 2007.

[10] World Wide Web Consortium. Voice extensible markup language (voicexml) version 2.0. http://www.w3.org/TR/voicexml20/, 2007.

[11] Andrea Corradini, Thomas Hanneforth, and Adrian Bak. A robust spoken language architecture to control a 2d game. In *Proceedings of the AAAI International FLAIRS Conference*, pages 199–204, 2007.

[12] Kay Stanney (Editor). *Handbook of Virtual Environments: Design, Implementation, and Applications*. CRC, 2002.

[13] R. Haller, H. Mutschler, and M. Voss. Comparison of input devices for correction of typing errors in office systems. In *Proceedings of INTERACT '84, First IFIP Conference on Human-Computer Interaction*, 1984.

[14] Alexander G. Hauptmann and Alexander I. Rudnicky. A comparison of speech and typed input. In *HLT '90: Proceedings of the workshop on Speech and Natural Language*, pages 219–224, Morristown, NJ, USA, 1990. Association for Computational Linguistics.

[15] Gunther Heidemann, Ingo Bax, and Holger Bekel. Multimodal interaction in an augmented reality scenario. In *ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces*, pages 53–60, New York, NY, USA, 2004. ACM.

[16] Gerd Herzog, Alassane Ndiaye, Stefan Merten, Heinz Kirchmann, Tilman Becker, and Peter Poller. Large-scale software integration for spoken language and multimodal dialog systems. *Natural Language Engineering*, 10(3-4):283–305, 2004.

[17] L. Karl, M. Pettey, and B. Shneiderman. Speechactivated versus mouse-activated commands for word processing applications: An empirical evaluation. *Internional Journal of Man-Machine Studies*, 39:667–687, 1993.

[18] Arjan J. F. Kok and Robert van Liere. A multimodal virtual reality interface for 3d interaction with vtk. *Knowledge and Information Systems*, 11(3), 2007.

[19] Kai-Fu Lee, Hsiao-Wuen Hon, and Raj Reddy. *An overview of the SPHINX speech recognition system*, pages 600–610. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[20] John Leggett and Glen Williams. An empirical investigation of voice as an input modality for computer programming. *Int. J. Man-Mach. Stud.*, 21(6):493–520, 1984.

[21] Scott Mcglashan. Speech interfaces to virtual reality, 1995.

[22] Michael F. McTear. Spoken dialogue technology: enabling the conversational user interface. *ACM Computing Surveys*, 34(1):90–169, 2002.

[23] Inc. Nuance Communications. Nuance recognizer. http://www.nuance.com/recognizer/, 2007.

[24] Sharon L. Oviatt. Advances in robust multimodal interface design. *IEEE Computer Graphics and Applications*, 23(5):62–68, 2003.

[25] R. Pausch and J. H. Leatherby. A study comparing mouse-only input vs. mouse-plus-voice input for a graphical editor. *Journal of American Voice Input*, 9(2), 1991.

[26] Ednaldo Pizzolato, Marcio Fernandes, and Cleber Leonel. Voicecube: a tool to investigate the usability of speech and mouse interaction on a 3d world. In *Proceedings of the X Symposium of Virtual Reality*, 2008.

[27] G. K. Poock. Voice recognition boosts command terminal throughput. *Speech Technology*, 1(2):36–39, 1982.

[28] Rajeev Sharma, Michael Zeller, Vladimir Pavlovic, Thomas S. Huang, Zion Lo, Stephen M. Chu, Yunxin Zhao, James C. Phillips, and Klaus Schulten. Speech/gesture interface to a visual-computing environment. *IEEE Computer Graphics and Applications*, 20(2):29–37, 2000.