Multimodal Animation Control

by

Hana Kim

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
May 21, 2003 [June 2003]

Author . . . .
Department of Electrical Engineering and Computer Science
May 21, 2003

Credited by . . . [illegible] . . . .
Larry Rudolph
Principle Research Scientist
Thesis Supervisor

Accepted by. . . . . [illegible] . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Multimodal Animation Control

by

Hana Kim

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, we present a multimodal animation control system. Our approach is based on a human-centric computing model proposed by Project Oxygen at MIT Laboratory for Computer Science. Our system allows the user to create and control animation in real time using the speech interface developed using SpeechBuilder. The user can also fall back to traditional input modes should the speech interface fail. We assume that the user has no prior knowledge and experience in animation and yet enable him to create interesting and meaningful animation naturally and fluently. We argue that our system can be used in a number of applications ranging from PowerPoint presentations to simulations to children's storytelling tools.

Thesis Supervisor: Larry Rudolph
Title: Principal Research Scientist

Acknowledgements

First and foremost, I would like to thank my advisor, Larry Rudolph, for everything. Without his energy, guidance, and encouragement, this project would not have been possible. When I first showed up in his office, he gave me an opportunity to work on a project which I loved from the very beginning and enjoyed every part of. He didn't even look at my resume then and I believe, to this date, he still hasn't.

I would like to thank my officemates, L. Brunsman, Glenn Eguchi, Sonia Garg, Kan Liu, Arjun R. Narayanswamy, and Jorge Ortiz for always providing me with a healthy dose of distractions and for being my test subjects.

I would like to thank Scott Cyphers and Eugene Weistein for generously sharing their expertise of SpeechBuilder with me.

I would also like to thank Reid Andersen for helping me with mathematical problems and being a wonderful listener.

I am very thankful to have my late night pho buddies, Grace Lim, Marvin K. Shin, and Minjoon Kouh. They have made my years at MIT memorable.

I am also grateful that I could live in a wonderful city like Boston for all of my college years. Its terrible weather aside, Boston is a wonderfully vibrant city full of young, energetic people.

Lastly and most importantly, I can't thank my family enough. My parents and grandparents have always been supportive of everything I do. I don't think I would be here without their never-ending encouragement. I can't leave out my little brother, Justin, for playing the devilish little brother role.

Contents

Chapter 1

Introduction

The multi-mode input mechanism plays an essential role in a human-centric computing model proposed by MIT's Project Oxygen. We propose and demonstrate a speech-driven animation control system with traditional computer inputs as fall-back methods. The proposed system assumes that the user does not have experience in animation and yet allows the user to create and control animation in real time with simple speech commands.

1.1 Human-Centric Computing

Project Oxygen proposes a human-centric computing model that allows people to interact with computers the same way they interact with other people. For many years, we had to learn how to talk like computers. Many systems, if not all of them, were designed around computers not people. The future of computing, however, lies in a human-centric model. As computer scientists, we have a task of making machines understand humans. We propose to do so by a multi-mode input mechanism.

1.2 Multi-mode Input

After years of technology revolution, we still rely on our keyboard and mouse to perform any kind of tasks on our personal computers. Although many have acquired skills to use keyboard and mouse effectively, both keyboard and mouse are, by no means, an ideal mode of input. The most ideal way of input would closely resemble the ways humans communicate with each other. Humans use a variety of input modes to communicate. Hence the multi-mode input mechanism. Not only would it be more effective than any single input mode but it would also closely resemble the human communication mechanism [1]. We present a multimodal animation control system that combines traditional computer inputs with speech. Speech allows the system to respond to users more effectively and fluently while keyboard and mouse act as back-up methods users can resort to.

1.3 Animation Control

Animation is a very effective way of delivering information. One can use animations in a variety of applications ranging from simulations to PowerPoint presentations to children's storytelling tools. Creating or controlling animations, however, is still considered to be a specialized task reserved for professionals. We are not interested in producing the state of the art animations that require an exorbitant amount of computing power. Rather, we are interested in the process of creating animations and inventing new tools to help novices create and control animation effortlessly. The focus of this thesis is on inventing "a simple, yet versatile method of animation that allows for interactive control." [2]

In particular, we present a speech-driven system that allows one to create and control animations in real time. We also provide traditional computer inputs—keyboard and mouse, as fall-back methods because the speech interface is still not quite robust to stand alone. Unlike many existing animation tools, we do not require the user to possess any knowledge of cartoon making prior to using the system. The user should be able to create and control animation in real time using plain English. It is also important to note that the user can be involved as much or as little as he wants in the process of creating and controlling animation. In the default setting, the user is given pre-defined objects and speech commands thus limited to creating the background and animating characters. However, if he wants to, he can be involved from creating objects to adding or modifying speech commands. Furthermore, the speech interface allows the user to create animation in real time—a truly novel approach to animation control. Any type of "serious" animation control system requires the user to do a significant amount of work (usually scripting or key-framing), if not all of it, beforehand. They draw a clear distinction between the creating period and the showing period. The proposed system, on the other hand, does not draw the distinction. This is a deliberate design choice to merge the two periods thereby granting the system the real-time control quality.

1.4 Outline

This thesis describes a multimodal animation control system that allows users to create and control interesting and meaningful animation in real time using the speech interface. Chapter 2 examines the problem at hand by defining animation, discussing the design goals, and surveying related work. Chapter 3 presents the detailed design and implementation of the animation control system. In Chapter 4, we suggest possible future work. Chapter 5 concludes this thesis.

Chapter 2

Problem Overview

The task at hand is to develop an animation control system based on a multimodal input mechanism. To be more specific, we are interested in developing a speech interface for creating and controlling animation. We also provide traditional computer inputs as fall-back methods. In this section, we present our task in detail and address our motivation for the project by examining terminology, design goals, and related works.

2.1 Terminology

In this section, we define perhaps the most important term for this work—animation. At an early stage of the project, the author reviewed a number of books and articles on animation tools. One of the books boldly claimed that "moving stuff around the screen is not animation" because "anyone can do that" [3]. In this thesis, however, we claim exactly the opposite. We *do* define animation moving stuff around the screen. More importantly, anyone and everyone should be able to do that. By animation, we do not mean the extravagant kind one encounters in commercials, video games, latest movies, etc. Rather, we mean computationally inexpensive and easy to create yet meaningful and interesting animation one can incorporate into PowerPoint presentations, instruction manuals, children's storytelling tools, etc.

2.2 Design Goals

2.2.1 Interactive, Responsive Interface

The main goal of this thesis is to create a speech-driven interface that would respond to the user input in real time. The user can fall back to traditional computer inputs should the speech interface fail. We first discuss the degrees of interactivity in different animation control systems and determine where the proposed system fits in the spectrum.

Not all animation control systems allow the same degree of interactivity. In some systems, the user merely plays the role of an audience member while in other systems the user is responsible for creating and controlling the majority, if not all, of animation. We group animation control systems into three different categories. This is somewhat arbitrary but appropriate for our discussion. We name the first category the passive user system. In the passive user system, the user is an audience member and nothing more. There is a clear distinction between the author and the audience. The author creates everything in advance and the audience passively views the author's final product. Saturday morning cartoons are an example of the passive user system. The second category is the limited control system. Most of the web animations available are examples of the limited control system. The limited control system allows the user to control a small portion of the animation through mouse clicks or buttons. The scope of control the user has in such systems is carefully limited by the author. The third category, the interactive system, closely resembles the system presented in this thesis. The user plays an active role in creating and controlling animation. While the system might provide pre-defined objects and easy-to-use animation tools, the user is largely responsible for creating animation. We examine examples of the interactive system in Section 2.3.

The proposed system is distinguished from most interactive systems by its speech interface and real-time control. Most interactive systems ask the user to write scripts, specify key frames, or define behaviors in advance [4]. Furthermore, the user is expected to have some level of animation knowledge and experience beforehand. The presented system does not require the user to have experience in animation prior to using the system. Instead of learning new or obscure animation terms, the user is invited to learn simple speech commands created with novices in mind. More importantly, with the speech interface, the system can be quick and responsive. It is clear how the speech-based system can be faster in responding to the user than the script-based system for example. Its quick response time allows the system to be real-time. In the real-time control model, animation takes place as the user speaks into the system. This is a novel approach few, if at all, systems have attempted. We believe that this approach is a very good answer to creating a truly interactive and response animation control system.

2.2.2 Extensibility

The system is easily extensible and opens up an array of opportunities for developers and designers. One can extend the system either by providing more animation content or writing applications on top of it. Adding more content is painless and, in fact, encouraged. When one adds more content, the system creates appropriate speech commands, makes them available to the user, etc. Our framework also encourages developers to build applications on top it. The framework is modular allowing developers to substitute or extend a module, whether it be the front-end code in ActionScript, the server/client code, or the speech-interface, without affecting other modules. Of course, extensibility of the system benefits the user the most. The system now can give the user more content, tools, and control he could not enjoy before. With designers and developers diligently providing interesting content and tools, the end-user can create and control a wide spectrum of animations that are useful and fun. We will discuss the details of the implementation and possible applications in Section 3 and Section 4 respectively.

2.3 Related Work

2.3.1 Multimodal Input

Multimodal input mechanism is ideal for a human-centric computing model because it closely resembles the ways humans communicate with each other. A number of research projects, ranging from artificial agents to intelligent environments, employ multimodal input mechanism. We investigate the benefits of multi-mode input in each of the projects.

In his paper in IJAAI, Kristin R. Thorisson, at the MIT Media Laboratory, presents Ymir, a computational model based on multimodal perception and multimodal action. Ymir is designed to be used for creating autonomous creatures, often called artificial agents, capable of communicating with real users[5]. Thorisson claims that Ymir can be used to create synthetic characters and robots who can carry out "task-oriented dialogues." Gandalf, the first prototype character in Ymir, is an expert of multimodal interaction. He understands hand movements, eye movements, body language, turn-taking signals, and, of course, speech. Furthermore, he also produces multimodal motor output—hand movements, facial expressions, eye movements, body language, turn-taking signals, and speech. Ymir uses sophisticated approaches from various fields such as artificial intelligence, cognitive science, and psychology. More importantly, it employs multimodal interaction to a great extent. On the downside, Ymir requires a great deal of computing power. Eight computers are used to run Gandalf's software and numerous sensors and trackers have to be attached to the user [5]. While it might not be as complex and comprehensive as Ymir, our animation control system is an persimonious system in terms of computing power and yet does its job quite well.

Michael H. Coen of MIT Artificial Intelligence Lab describes design principles behind *Intelligent Environments* (IEs) in his paper in AAAI-98. He states that the motivation for building IEs is to involve computers in tasks in physical world and to "allow people to interact with computational systems the way they would with other people: via gesture, voice, movement, and context [6]." He argues that IEs are designed to make human-computer interaction (HCI) seamless and computers invisible. Instead of relying on traditional computer UI primitives like menus, mice, and windows, IEs utilize gesture, speech, affect, and context [6][7]. His belief in "people-interfaces for computers" rather than "computer-interfaces for people" is very similar to our human-centric computing approach.

2.3.2 Interactive Animation

In this section, we examine two truly interactive animation systems, Alice, a 3D graphics programming environment developed at University of Virginia and Carnegie Mellon University, and Stagecast Creator, a simulation toolkit developed by Stagecast Software.

Alice is a 3D graphics programming environment designed for people who do not have any 3D graphics or programming experience [8]. Similar to our assumption that a broad audience will be interested in creating animation, the creators of Alice assume that a diverse audience, who doesn't necessarily have the graphics or programming experience, will be interested in creating interactive 3D animation. In particular, they choose a target audience, non-science/engineering undergraduates, and design the system with their needs in mind. Authoring in Alice takes two steps: creating an opening scene and scripting [8]. Similar to our system, Alice's users choose objects from an object gallery. Unlike our system, Alice's library contains a large number of low-polygon models. User can also import objects in popular file formats [8]. The user creates the opening scene by placing the objects in desired locations, setting the camera location, and finally saving this initial state into a *world file*[8]. Once the opening scene is ready, the user can begin scripting in Python. He can iteratively edit the script and run it. Alice is fundamentally different from our system in two ways. The obvious difference is that Alice deals with 3D animation while we are mainly interested in 2D animation. While 3D animation is prevalent and taken for granted, 2D animation is often sufficient and sometimes better than 3D animation without additional work. More importantly, Alice is essentially a script-based animation control system. The user has to write scripts ahead of time and run it to play animation. The script-based control is simply not practical for real-time control unless there are hundreds of shortcuts available and the user can script at an incredible speed. Our speech-driven system, on the other hand, is very ideal for real time control. In fact, it is specifically designed for real time control. Animation happens as the user speaks to the system. We do not require the user to learn a scripting language. We do require the user to learn a few speech commands that are in plain English. While Alice's scripting system might provide the user with a wider range of control, our system provides something no other system has provided so far—real time control with a speech interface.

Stagecast Creator is a simulation toolkit that allows children and other novice programmers to build interactive stories, games, and simulations without syntactic programming languages. The goal of Creator is to enable children and novice programmers to construct and modify simulations without a programming language [12]. They accomplish this goal by using two technologies: programming by demonstration (PBD) and visual before-after rules. Users create characters and specify how the characters are to behave and interact [11]. First, users either choose characters from the given list or design their own characters. Then, they create simple rules that govern how the characters move by demonstration—specifying the before and after states. Rules can also manipulate the characters' properties and appearances. Users can make multiple instances of the same type character. Every instance will have the set of properties, appearances, and rules; however, it has its own value for a property and its own drawing for each appearance. Extensive user studies have shown that Creator's scheme, as opposed to that of traditional programming languages or scripting languages, is effective and well-received by novice programmers [11][12]. We will not discuss the details of the user studies here; however, it's important to note that Creator's visual rule-based approach was indeed successful. We did consider using the rule-based approach earlier in our project. While it might be more versatile and powerful, the rule-based control is not quite suitable for real-time control with speech interface. In our system, there are rules that govern the ways characters behave; however, the end-user does not play a role in creating them. Instead, the end-user can choose from an array of objects with different appearances, properties, and rules.

Chapter 3

Animation Control

In this section, we present a multimodal animation control system that allows the user to create and control animation in real time. We first review the main technologies used in creating the system, present the detailed design and implementation, and discuss the initial user response to the prototype of the system.

3.1 Technology Background

This project is an integration of existing technologies that have proven to be effective and powerful in what they do. For speech interface, we use SpeechBuilder, developed by Spoken Language Systems (SLS) group here at MIT Laboratory for Computer Science. For animation, a popular animation tool, Flash MX is used.

3.1.1 SpeechBuilder

Researchers in SLS created SpeechBuilder, a development tool that allows developers to build their own speech-based applications without delving into human language technology (HLT) [14]. SpeechBuilder asks developers to define necessary semantic concepts by defining actions and keys to create a domain. Developers can do this either by using SpeechBuilder's web interface (http://speech.lcs.mit.edu) or by uploading an XML file and generating domain files based on it. We choose to do the latter to allow frequent changes to the XML file.

SpeechBuilder uses an example-based method to allow developers to lay out the specifics of the domain [15]. One needs to provide example sentences for actions along with examples for keys. The following two tables, taken from the SLS group website, show example keys and actions.

Action	**Examples**
Identify	What is the forecast for Boston What will the temperature be on Tuesday I would like to know today's weather in Denver
Set	Turn the radio on in the kitchen please Can you please turn off the dining room lights Turn on the TV in the living room
Good_bye	Good bye Thank you very much good bye See you later

Table 2: Examples of "actions" in SpeechBuilder knowledge representation

In our case, actions would consist of **create**, **delete**, **move**, etc. Keys would consist of **object**, **object_name**, etc. We will discuss them in further detail in Section 3.2.4. Once the XML file is created, SpeechBuilder can generate domain files utilizing the HLT's developed by the SLS group.

SpeechBuilder provides developers with two ways to write applications—CGI protocol and FrameRelay. FrameRelay is written in Java and relays the processed data, usually the user input string or a message that says it has failed to understand the user input. We extend FrameRelay to parse the user input and extract the information we need and send it to the Flash environment via a socket connection.

3.1.2 Flash MX

Flash is a vector-based animation tool that's widely used for gaming consoles, advertisements, user interfaces, web games, and cartoons. Because it is vector-based, Flash produces animations that are scalable without compromising clarity and resolution [16]. In addition, Flash is quite suitable for interactive applications therefore quite applicable to our task at hand. What makes Flash a powerful development tool is its accompanying scripting language, ActionScript [Flash MX savvy]. ActionScript is an object-oriented scripting language based on JavaScript. In our project, we use ActionScript to manipulate characters and background objects and to communicate with a server which, in turn, communicates with the speech interface.

While we need not concern ourselves with the most of the details of the Flash authoring environment in this thesis, it's important to understand a very important feature of Flash—movie clips. Movie clips are self-contained movies that run independently of the main timeline [16]. Movie clips are ideal for controlling a number of independent objects. They come with a number of properties and built-in methods one can manipulate easily. We use movie clips as fundamental building blocks of our animation. Although end users do not have deal with them directly, our animation is essentially a collection of movie clips.

3.2 Animation

We are now ready to build a multimodal animation control system that allows the user to create and control animation in real time. The focus of this thesis is to invent new approaches to make the process of creating animation more sensible and less time consuming for both novices and experts. We design a framework that allows the end user to be involved as much or as little as he wants in the process of creating animation. The default setting is that the user creates the background and animates the characters in real time. However, experts can be involved in creating objects, the building blocks of the system, and adding and modifying speech commands as well.

3.2.1 Objects : Building Blocks

Objects are the basic building blocks of animation in our system. An object is essentially a collection of movie clips packaged together for convenience and speedy access. Objects are created for two main reasons—to generate speech commands easily and to better real time control. Generating speech commands for each movie clip can be cumbersome. By packaging a set of related movie clips together, we can generate a fewer speech commands in an organized manner. The same argument holds for the real time control. Instead of manipulating a large number of movie clips, the user should be able to handle a set of related movie clips speedily. For example, **Penguin**, an object in our prototype, can bow, dance, and eat ice cream. In the movie clip model, one needs to manipulate four separate movie clips—one for each task and a default movie clip. The object model allows the user to deal with the well-packaged **Penguin** instead of four separate movie clips. In other words, when the user creates a Penguin, four movie clips are at his disposal without doing additional work. We will discuss the details of objects and how they shape our framework in sections that follow.

3.2.2 Background Objects and Characters

In the beginning of the project, we created two separate classes of objects. We believed that we needed to differentiate background objects from characters. After developing a couple of prototypes, we've realized that there is no need to make the distinction between the two when creating objects. In fact, not making the distinction lends us generality and flexibility we didn't have before. For example, a penguin can be a character or a background object. One should be able to make a wall of penguins for his background if he wants to. The only difference between background objects and characters is that background objects can not move around. However, they should be able to perform the same type of tasks as characters so long as they are in place. This is strictly a design choice we've made. There's no rule as to how background objects should behave. We will discuss the details on background objects and characters in Sections 3.2.6 and 3.2.7. For now, we emphasize that there's no difference between background objects and characters when creating objects for end-users to use.

3.2.3 Object Naming Scheme

We devise a simple and intuitive naming scheme of movie clips to group them into related groups also known as objects. The naming scheme plays an essential role in automatically generating files needed for the speech interface later on. The naming scheme is vaguely based on the syntax of Java. Java class name corresponds to object name and method names to task names. In Java, when an instance of a class is created, all public methods of the class come with it. Similarly, when an instance of an object is created in our animation system, all tasks or movie clips for tasks are exported along as well.

In order to create an object, one needs to create the default movie clip—the movie clip to be played when first created or when the object is not performing any task, plus one movie clip per task. The default movie clip for **Penguin** would be named simply Penguin. For each task, a movie clip will be created and named "Penguin" followed by ".task." For instance, to add dance to **Penguin**'s task list, one would make a movie clip of a penguin dancing, name it "Penguin.dance" and place it in the library of Flash authoring environment. Table 3 is an example of the kinds of movie clips one would create for the object **Penguin**. Of course, it can have more tasks, therefore more movie clips. It can also fewer tasks or even no task at all.

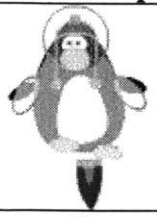
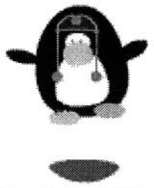
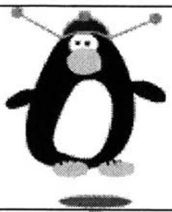
Name	Type	Movie Clip
Penguin	Default	
Penguin.bow	Task	
Penguin.dance	Task	
Penguin.eat_ice_cream	Task	
Penguin.hop	Task	

Table 3: Movie clips are created and named according to the naming scheme for Penguin.

3.2.4 Actions and Keys

In order to create the XML file for the speech domain of our system, we need to define actions and keys that contain all necessary functions and key concepts for an animation control system.

The actions consist of **create**, **delete**, **name**, **perform**, **place, move, freeze** and **unfreeze**. Names speak for themselves. **Create** is used to create an object. **Delete** is for deleting an object. **Name** is used to name the object just created. **Perform** is called to make an object perform a task. **Place** is called to place a background object. **Place** places a background object on the 10 x 10 grid. **Move** moves a character from one background object to another. It is important to note that **move** is not the only way to move a character. One can add "from" and "to" to any **perform** command. For example, one can say "Make Bob dance from Pizzeria to Library" and expect Bob to move from Pizzeria to Library while dancing. **Freeze** and **unfreeze** are used in creating the background. The background consists of a collection of objects that remain in place while the end-user is animating characters. When he finishes placing objects in the background, the user can "freeze" the background thereby makes all the objects on the screen unmovable during character animation. All the objects added before "freeze" would be background objects and all the ones added after "freeze" would be characters the end-user would be animating in real time. One can also "unfreeze" and modify the background.

The keys consist of **object, object_name,** and **task**. **Object** is a list of all objects created. **Object_name** lists names the end-user could give to objects. Unfortunately, as of now, the end-users have to use names from the list of names we provide. **Task** is a list tasks objects can perform. We do not specify which task belongs to which object at this point. Type checking is done upon receiving user input and appropriate tasks are exported depending on the type of the object.

Action	**Function**	**Example Sentence**	**Comment**
Create	Creates an object on the screen	"Create a Penguin"	Must use an object name not a task name. Ex) "Create a dancing penguin" is not allowed.
Delete	Deletes an object on the screen	"Delete Bob"	Can delete an object anytime. Ex) Can delete a penguin while it's dancing
Name	Names the object just created	"Name him Bob"	Names an instance not the class. Can use the name to access the object for later use.
Perform	Makes an object perform a task	"Make Bob dance" "Make Bob hop from Pizzeria to Library"	Can use "from" and "to" to move the object while performing the task
Move	Moves a character from one location to another	"Move Bob from Library to Pizzeria"	Locations are background objects.
Place	Places a background on a grid point	"Place Bob at (3,4)"	A 10 x 10 grid is given initially for placing background objects
Freeze	Makes objects created before the command unmovable during animation	"Freeze!"	Used to make the background. Called when done placing objects in the background
Unfreeze	Makes the background objects movable again	"Unfreeze"	Called to modify the background—to add more objects, delete objects, and move existing objects, etc.

Table 4: Actions to be included in the XML file for the speech interface

Key	**Example**	**Comment**
Object	Penguin	All the objects created by the designer
Object_name	Arjun Bob Glenn Sonia	All the names that can be assigned to objects—both background objects and characters
Task	bow dance eat_ice_cream hop	Tasks objects can perform

Table 5: Actions to be included in the XML file for the speech interface

3.2.5 Generating Speech Domain Files

With the smart object naming scheme and keys and actions described in the previous section, we can easily generate speech domain files. The GALAXY framework use speech domain files to understand user input and produce desired output. Domain files are generated based on an XML file that contains all the necessary information to create a speech domain—keys and actions. The XML file is easily generated using the object naming scheme discussed in the Section 3.2.3.

A text file is created to list the names of the movie clips created for all objects . Only one file is needed for all objects. A sample text file would read:

```
Begin Penguin
Penguin.bow
Penguin.dance
Penguin.eat_ice_cream
Penguin.hop
End Penguin
```

If one wants to add more objects, he can add them right after the Penguin movie clips. There are three simple rules in writing the text file. We need to place one movie clip per line. The default movie clips does not need to be listed. All the movie clips for an object need to be grouped together, begin with `Begin Object` end with `End Object`. For example, `Dog.bark` can not appear in the middle of the **Penguin** movie clips. It needs to appear somewhere between `Begin Dog` and `End Dog`.

Because of the intuitive naming scheme, the XML file can be easily generated based on the text file. The XML file is grouped into a number of blocks. The header of each block is generated based on actions and keys. The text file is used to fill in the values of keys and generate example sentences for actions. For example, **task** is a key and consists of tasks objects can perform. The XML file would contain the following lines for **task**:

```
<class type="Key" name="task">
   <entry>bow</entry>
   <entry>dance</entry>
   <entry>eat_ice_cream</entry>
   <entry>hop</entry>
</class>
```

For an action, entries would consist of example sentences. For **create**, an action, the following lines would be generated:

```
<class type="Action" name="create">
   <entry>create a penguin</entry>
   <entry>make a penguin</entry>
</class>
```

It's important to note that above segment not only specifies two different ways of invoking **create** but also indicates that it can be applied to all **objects**. In other words, using one member of the key in an example sentence is sufficient to indicate that the action applies to all members of the key. Here's another example:

```
<class type="Action" name="perform">
   <entry>make Bob dance</entry>
</class>
```

Here, dance is a member of the key, **task**. Therefore, the entry above indicates that users can say, "make Bob bow," "make Bob eat ice cream," and "make Bob hop." As a result, the XML file generated is a concise yet easy-to-understand representation of our speech domain. In addition, if one wants add or modify speech commands for an action, he is welcomed to modify the relevant block of the XML file. For instance, the following block is generated for the action, delete:

```
<class type="Action" name="delete">
   <entry>delete Bob</entry>
</class>
```

One can easily add another entry to the block:

```
<class type="Action" name="delete">
  <entry>delete Bob</entry>
  <entry>pop Bob</entry>
</class>
```

We are now ready to generate domain files. To review, domain files are a collection of files that the Galaxy framework would communicate with to understand and process user input. At this point, with a well-crafted XML file, we are only one click or one command line away from generating them. We do not delve into the details of how the domain files are generated for it is not the focus of this research. We will simply user the tools in Speech Builder to generate them.

3.2.6 Creating the Background

In the early stages of development, we provided a ready-to-use background for end-users. They did not have any say in it and had to live with it. This was obviously not a good choice for most users would like to create their own background. We still provide the default background; however, we also give the user an environment to create his own background. The background not only sets the look and feel of animation but more importantly contains all the background objects which characters can move to and from. Let's clarify a couple of concepts before we move further in our discussion. We've said, in earlier sections, that there's no distinction between background objects and characters. This is true in a sense that a penguin can be both a background object and a character. In other words, all objects are designed to be background objects as well as characters. Once added to the scene, however, an object is either a background object or a character but not both. The difference between the two is that background objects remain in place during character animation. In addition, they serve as locations characters can move to and from and reside in. This is an important role because it frees users from using a number coordinate system when moving characters in real time. It is much easier and meaningful to say "Move Bob to Pizzeria" than "Move Bob to (3, 4)."

We will now delve into the details of how to create the background. When the end user first starts the system, the system prompts the user to choose either the default background or to create his own. If he chooses to create his own, the system enters the background creating mode. It does not contain any objects but does provide the list of objects available to users. Using **create** command, the user creates background objects. The system will immediately ask him to name the object just created. The user then can place the object in a desired location using **place** command. A typical background creating session would go:

1. User: Create a library
2. Sys: Please name it
3. User: Name it Bookworm Library
4. User: Place Bookworm Library at (5, 4)
5. User: Create a pizzeria
6. Sys: Please name it
7. User: Name it Fatty Pizzeria
8. User: Place Fatty Pizzeria at (2, 4)

9. User: Create a park.

10. Sys: Please name it

11 User: Name it Central Park

12. User: Place Central Park at (5, 5)

13. User: Place Bookworm Library (2, 4)

14. User: Freeze!

* Note: We did not include the system's reply to user input in the above session. The system repeats user input to acknowledge it. If it didn't understand what the user said, it asks him to repeat.

Above, the user created three background objects, named them, and placed them at different places. Once he names the objects, he can change locations of them anytime. In line 13, the user moves Bookworm Library to (2, 4) although Central Park was the last object created. When he is finished with the background, the user can "freeze" the background. **Freeze** signals that the user is finished with creating the background, exits the background creation mode, and enters the character animation mode. Figure 1 is a screen capture of the background just created. One can also "unfreeze" and re-enter the background mode if he wants to modify the background. The user could create as many background objects as he wants and place them anywhere on the screen; however, it is important to keep in mind that all background objects serve as locations characters would later travel to and from therefore create an appropriate number of them and place them in appropriate places.

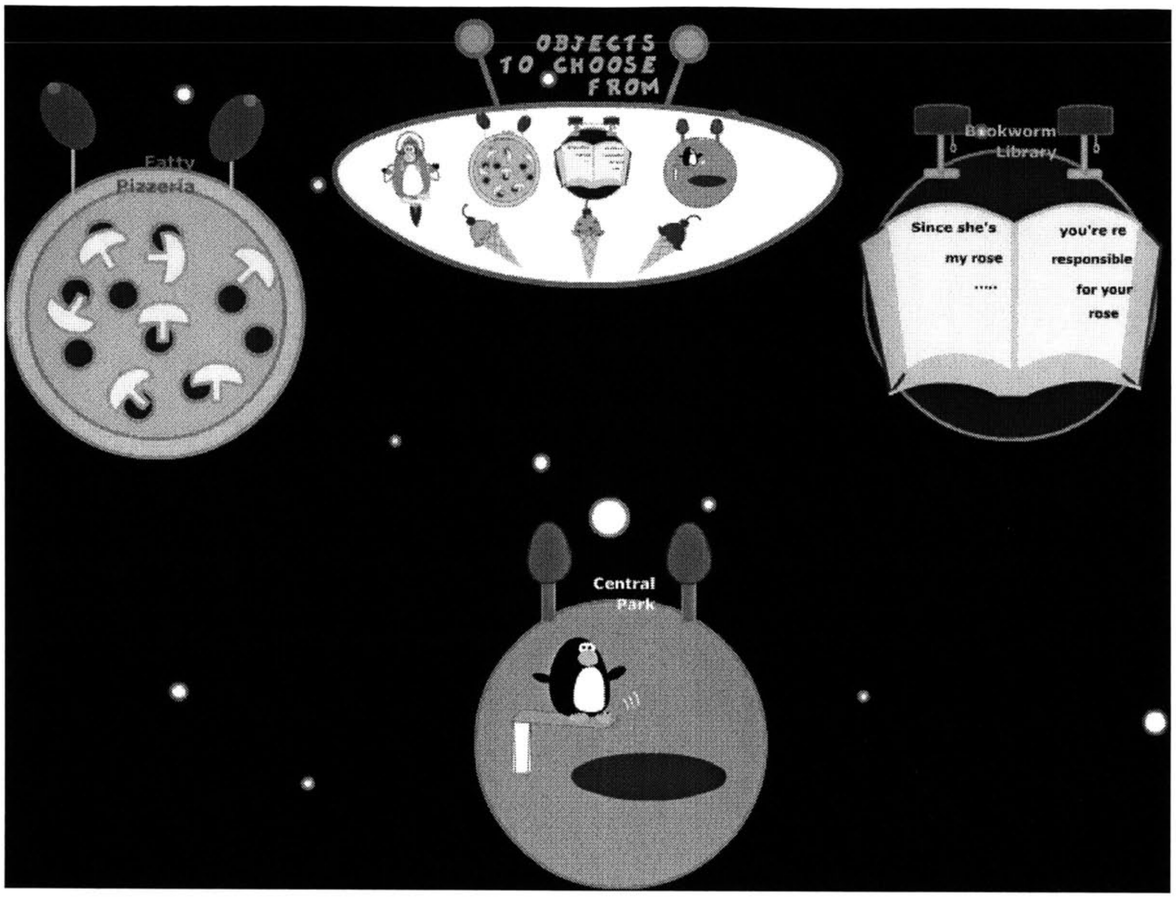


Figure 1: The background created by the user contains 3 background objects, Fatty Pizzeria, Bookworm Library, and Central Park.

3.2.7 Animating characters

Finally, the moment, we've been waiting for, has arrived. It's time to add characters and animate them. All the steps taken so far are preparations for what we are about to do. Let's review what we have done so far before we put on our big show. First, we've created objects in the Flash authoring environment to be used as background objects and characters using movie clips as our building blocks. We devised a sensible naming scheme to name objects so we can generate the XML file which contains semantic concepts necessary for creating animation. The XML file is then used to create speech domain files. The speech domain files define the speech interface end-users would be using. Objects stored in the Flash authoring environment and the speech interface built, it's the end user's turn to step in. Using **create**, **move**, **freeze**, and **unfreeze** commands, the user creates the background. This is where we are left off. With the background "frozen," the user can now add characters and animate them in real time.

Animating characters is not too different from creating the background. Characters are born out of the same set of objects background objects are created from. Any object added after **freeze** command is considered a character. One needs to create, name, and delete characters the same way he would with background objects. However, one needs to use a different approach to move characters around. Moving a character to a grid point as done with background objects is neither interesting nor meaningful. More importantly, this approach is not practical when the user is trying to produce animation in real time. We've decided that one good approach would be to move a character from one background object to another. This approach reduces the number of commands the user has to speak, thereby resulting in more continuous animation. Furthermore it gives the user full control over characters' actions since he is responsible for placing the background objects.

In addition to moving, characters are capable of performing tasks. When we made objects a while back, we assigned a number of tasks to each object. Tasks are designed to give the user freedom to create more interesting and content-rich animation. One can think of tasks as clip arts for animation only less annoying and more dynamic. For example, **Penguin** can bow, dance, eat ice cream, and hop. Novices would have to spend a great deal of time and effort if they were to create the effects themselves. We diligently provide objects for novices to create fun and useful animation so that they don't have to delve into the details of cartoon making. Ambitious novices and experts need not worry. They can always choose not to use already provided objects and instead create their own as long as they adhere to the naming scheme discussed in Section 3.2.3.

Without further ado, let's look at a typical character animation session:

1. User: Create a penguin.
2. Sys: Please name it.
3. User: Name him Bob
4. User: Move Bob to Fatty Pizzeria
5. User: Make Bob dance.
6. User: Make Bob eat ice cream from Tom's Fatty Pizzeria to Central Park

Above, the user created a penguin and named him Bob. Bob then moves to Fatty Pizzeria. Fatty Pizzeria is a happy place and Bob dances out of joy. Bob also gets an ice cream from Fatty Pizzeria. This part is not included in animation. Suppose Fatty pizzeria has an excellent selection of ice cream and Bob couldn't help getting one. Then Bob eats ice cream strolling to Central Park. Figure 2 through Figure 5 are screenshots of the character animation session described above. In step 4, we demonstrate how to move a character to a background object. Step 5 shows how to make the character perform a task. In step 6, we illustrate how to make the character perform a task while moving. It is entirely possible to add more than one character and make it as complex as the user wants. We put on "name tags" on the characters to help the user identify them. Before moving them,

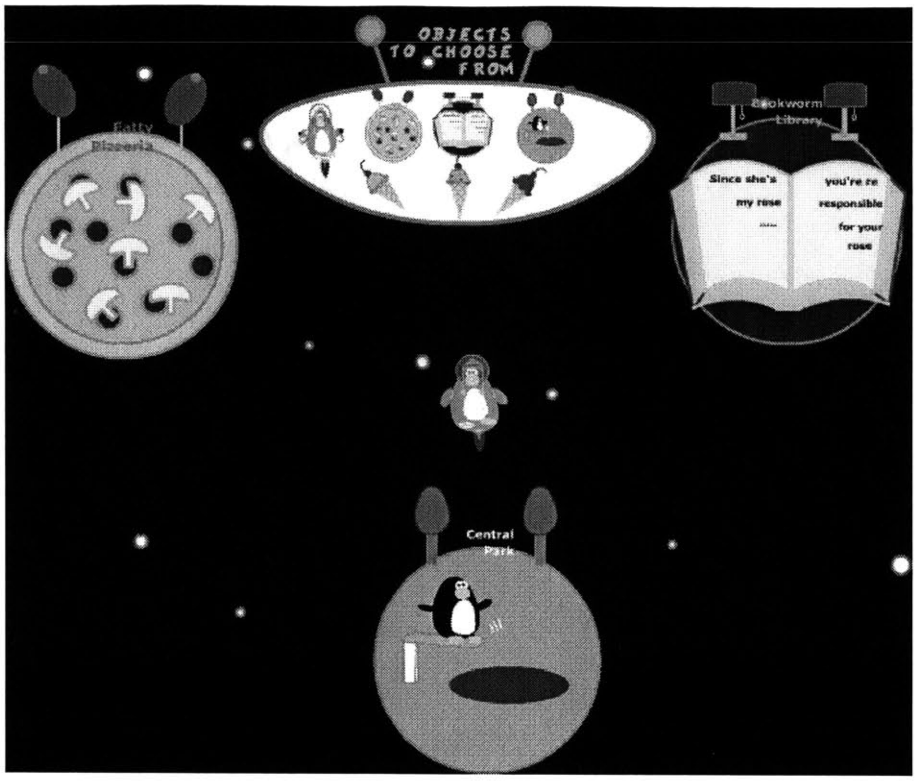


Figure 2: "Create a penguin." A penguin is created in the default position.

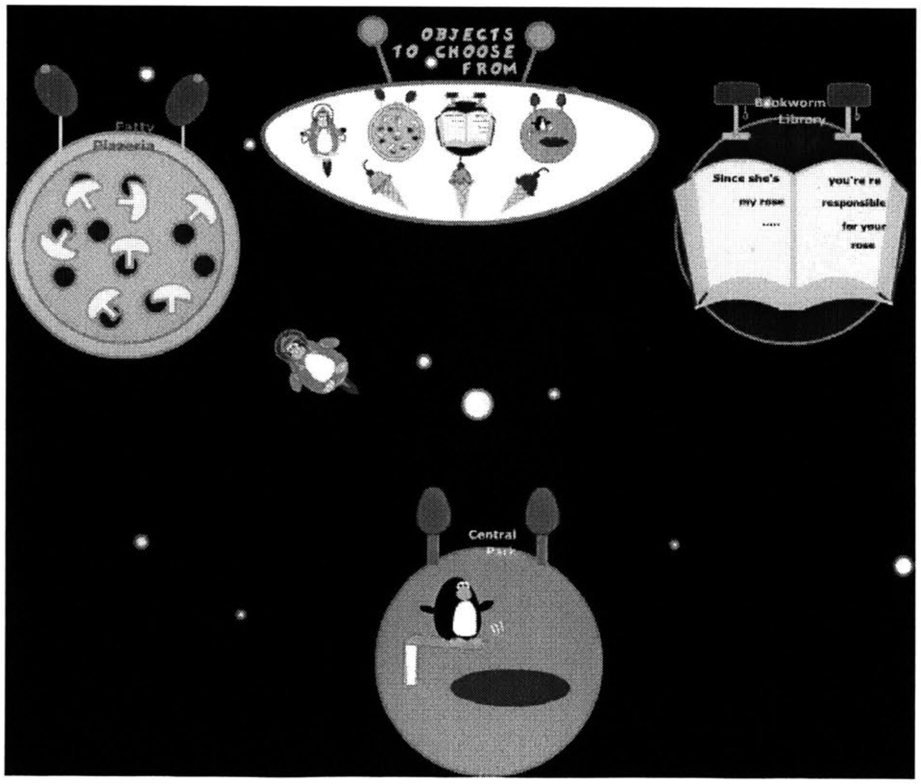


Figure 3: "Move Bob to Fatty Pizzeria." Bob is on the way to Fatty Pizzeria.

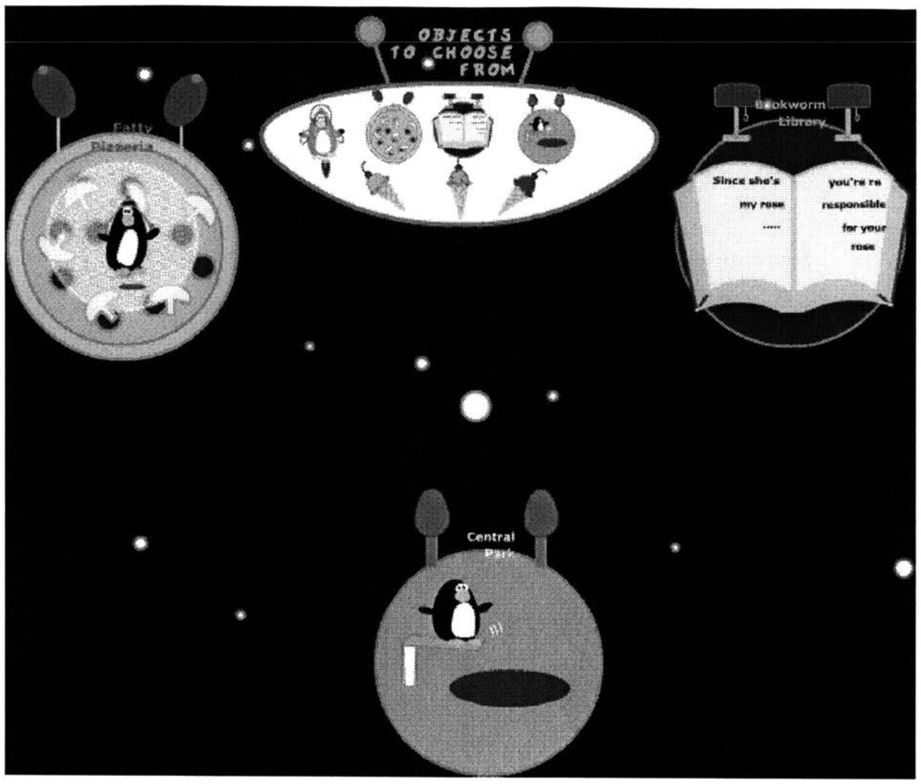


Figure 4: "Make Bob dance" Bob is dancing at Fatty Pizzeria

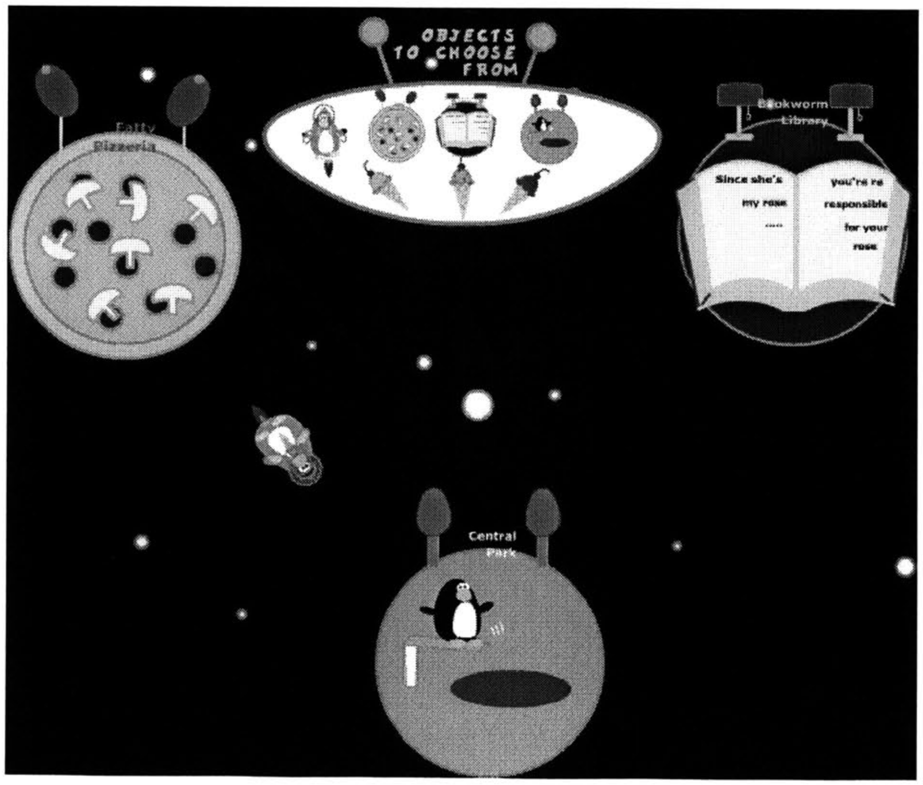


Figure 5: "Make Bob eat ice cream from Fatty Pizzeria to Central Park"

we turn the characters until they face the destination before moving them. This is implemented using the dot product algorithm.

3.3 Discussion

We've created a multimodal animation control system that lets users create and control animation in real time. Users can design the background and animate characters using the speech interface developed with SpeechBuilder. Users need not learn the specialized animation terms and techniques that other systems require them to master. It is important to note, however, that the user has an option to create their own objects and speech commands if he wants to. This is to accommodate both novices and experts. More importantly, animation happens before their eyes as they speak to the system. Although we haven't conducted formal user studies, informal user sessions have shown that users learn to use our system in matter of minutes, use the speech commands with ease, and most importantly, enjoy using our system to a great extent.

Our system turns out to be very engaging and plain old fun. We must admit that in our prototype we've deliberately designed the objects and the tasks to be interesting and somewhat humorous to capture the user's attention. When the penguin character became noticeably chubbier after eating an ice cream, most users kept making him eat more ice cream to observe the poor penguin's dramatic increase in size. Most of the test users, mostly graduate students and undergraduate students at Laboratory for Computer Science, exhibited childlike enthusiasm when creating and controlling animation with our system. This is not to say that our system is only designed for light weight, just for fun applications. In fact, in the next section, we describe a number of practical applications that can be built upon our framework. We interpret the users' enthusiasm and enjoyment as our success in creating something novel and engaging. More importantly, with the test users' encouragement, we are now in a position to make improvements to the system as well as build applications on top of it.

Chapter 4

Future Work

In this section, we present suggestions for future work. Although we've received good feedback with our first prototype, we still have a significant amount of work to improve the system. In addition, we propose a number of possible applications that can be built on top of our system.

4.1 Improvements

4.1.1 Objects

Providing a pre-populated list of objects is one of the main features that allow novices to create and control animation in real time without having to learn the details of cartoon making. Creating animation from graphics primitives—lines, points, and even polygons, is not appropriate for a speech-based control system. For one thing, there would be way too many speech commands to create a character or a background object. It is actually a tough task to determine what kinds of objects are appropriate for a speech-based control system. For our prototype, the types of objects provided are somewhat arbitrary. The objects are designed to make interesting characters or background objects; however, they are not designed with any type of user group or application in mind. Although we wouldn't know for sure what is the "right" set of objects without extensive user study or working applications, there are a few approaches we can try to improve the general quality of objects.

One approach would be involving the user in designing his own objects. We already encourage experts to create their own objects using movie clips in the Flash authoring environment. This is not the best approach with novices who do not have much experience with movie clips. For novices, we can create a friendly environment to create and edit objects. We can provide an editor with lower-level objects and a number of tools to allow building complex objects from simpler objects. The user would definitely have a greater control over the type of the objects, the look and feel of them, the tasks they can perform, etc.

The second approach is more straightforward. We can simply provide a large number of objects. Of course, we would have to think about the types of projects the end-users would be using our system for and organize the objects into libraries accordingly. One can call this approach the "clipart" approach. This approach doesn't require the user to learn any more than he has to for our original design. On the downside, this approach can be as frustrating and tedious as any type of "clipart" approach.

4.1.2 Gesture Interface

We were successful in creating a speech interface for animation control. We also provide the traditional computer inputs as fall-back methods. For our system to be truly multimodal, we must work on other modes of communication between the user and the system. In particular, we suggest implementing a gesture interface. Incorporating the gesture interface into our current system would simplify the existing speech commands or possibly eliminate some complicated speech commands. For example, the user now can say, “move the car from here to there” with the accompanying gesture instead of “move the car from Fatty Pizzeria to Central Park.”

4.2 Applications

A number of practical applications ranging from PowerPoint presentations to simulations to children’s storytelling stories can be built on top of our framework. Incorporating our system into one’s PowerPoint presentation would make the presentation dynamic and engaging. Instead of having multiple images or graphs in a discontinuous manner, we can now have an animation that fluently describes a process, a progress report, a dynamic graph, etc. The best part of it is that animation would happen as the presenter speaks to the audience. Pausing the presentation to run the animation is unnecessary. We can also create simulations easily. We do not recommend using our system to simulate an aircraft’s landing. We do recommend using our system to simulate how gravity affects a ball falling from a table. In addition, we can create a children’s storytelling tool using our framework. One can imagine how exciting and fun it could be for children when they see the characters moving and doing what they are told to do. This would be a great improvement over many existing children’s storytelling tools which merely provide a typing interface and a number of static images children can choose to include in their stories. With appropriate speech commands and interesting objects, children can enjoy a great deal of freedom in writing their stories and viewing their stories as they speak/write to the system.

Chapter 5

Conclusion

We've developed a multimodal animation control system that allows the user to create and control animation in real time. The system is largely speech-driven with traditional input methods as fall-back. The system is designed to be accommodating to both novices and experts. For novices, objects and speech commands are provided. Experts can choose to create their own objects and modify or add speech commands. The informal user study confirmed our belief that our approach is novel and has a great potential for creating new ways novices can create animation.

Bibliography

[1] Larry Rudolph. ORG-Oxygen Research Group. *Laboratory for Computer Science Annual Report 2001*

[2] Kristinn R. Thórisson. Toonface: A System for Creating and Animating Interactive Cartoon Faces. *Learning and Common Sense Section Technical Report 96-01.* April 1996

[3] Daniel Gray, John Kuramoto, and Gary Leib. *The Art of Cartooning with Flash*

[4] ACM SIGGRAPH. http://www.siggraph.org/

[5] Kristinn R. Thórisson. A Mind Model for Multimodal Communicative Creatures and Humanoides. *Internal Journal of Applied Artificial Intelligence 449-486.* 1999

[6] Michael H. Coen. Design Principles for Intelligent Environments. *Proceedings of the Fifteenth National Conference on Artificial Intelligence. (AAAI'98).* 1998

[7] Michael H. Coen. The Future of Human-Computer Interaction or How I learned to stop sorrying and love My Intelligent Room. *IEEE Intelligent Systems.* 1999

[8] Matthew Conway, Steve Audia, Tommy Burnette, Dennis Cosgrove, et al. Alice: Lessons Learned from Building a 3D System for Novices. *Proceedings of Conference on Human Factors in Computing System.* 2000

[9] Matthew Conway. Alice: Rapid Prototyping System for Virtual Reality

[10] Alice v2.0b. http://www.alice.org

[11] Allen Cypher and David Canfield Smith. KidSim: End User Programming of Simulations. *Proceedings of Conference on Human Factors in Computing System.* 1995

[12] David Canfield Smith, Allen Cypher, and Larry Tesler. Novice Programming Comes of Age. *Communications of the ACM, 43(3), March 2000, pp. 75-81.* 2000

[13] Stage Cast Software, Inc. http://www.stagecast.com

[14] Spoken Language Systems, MIT Laboratory for Computer Science. http://www.sls.lcs.mit.edu

[15] Eugene Weinstein. SpeechBuilder: Facilitating Spoken Dialogue System Development, Master of Engineering Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, May 2001

[16] Ethan Watrall and Norbert Herber. Flash MX Savvy.