

# Script Visualization (ScriptViz): a smart system that makes writing fun\*

Zhi-Qiang Liu

Centre for Media Technology (RCMT) and School of Creative Media  
City University of Hong Kong, P. R. CHINA

*smzliu@cityu.edu.hk*

## Abstract

We have built the first version of ScriptViz v.1.0 that allows users to visualize their screenplays in real time via animated graphics. Our system consists of a text understanding module, a high-level planning module and a scene generator. The user can input a screenplay as a set of well-formed English sentences via a graphic user interface. The text understanding module interprets the input sentences and triggers the high-level planner to construct a plan of actions for the appropriate agents. Then the agents execute the plans, and the scene generator renders the scene as the story evolves. Our system provides the user with a powerful tool to visualize his screenplays (stories) in the form of computer graphics, which makes writing story fun for students as well as for screenplay writers.

**Keywords:** Artificial Intelligence, Natural Language Processing, Motion Planning, Animation, Storyboard, Screenplay, Computer Graphics.

## 1 Introduction

There have been many projects on natural language processing and simulation in virtual environments. Among these, the project that draws a lot of attention is the AnimNL project at the University of Pennsylvania [2, 9]. The goal of the AnimNL project is to design an architecture that generates realistic animations of characters performing tasks specified through natural language instructions. The architecture is later implemented in SodaJack system [5] that animates

---

\*This research was supported by Hong Kong Research Grants Council (RGC) Project No. CityUHK #9040690-873 and a Strategic Development Grant (SDG) Project No. #7010023-873 City University of Hong Kong.

a human working at a soda fountain. SodaJack system accepts natural language instructions as input. The instructions are interpreted as a set of goals. The system then constructs plans to search for and manipulate objects, and finally it is simulated by the Jack agent. SodaJack focuses mainly on developing a set of high-level planners to equip an agent with searching and object manipulation capabilities.

Our research on ScriptViz, however, was motivated by the fact that traditional motion-picture productions, e.g., TV-shows and movies, often involve drawing storyboards. It provides a means for a screenplay producer to see the visual appearance of a particular scene and to convey information about scene settings, camera locations, character locations and poses, based on which the actually shooting can be planned. However, drawing storyboards is a slow and tedious process. In addition, storyboards cannot provide information such as characters' actions and interactions.

In this project, we use technologies in natural language understanding to process well-formed (English) sentences, from which ScriptViz is able generate animated graphics. In ScriptViz, we have developed a realistic animation of human figures and animals to carry out tasks according to a screenplay consisting of a set of well-formed sentences that are syntactically and grammatically correct and less ambiguous [1]. For instance, sentences in articles in reputable magazines and official documents are *generally* well-formed. ScriptViz provides the writer with a smart platform for visual feedback and, indeed, is a powerful tool useful in content visualization during creative writing.

In this paper, we describe the ScriptViz (v.1.0) that automates this process and enables the user to experiment and test his screenplays.

## 2 The Problem

A storyboard is a pictorial representation of a particular scene in motion-picture productions. It helps screenplay writers and film directors to demonstrate the visual appearance of a particular scene. Figure 1 shows an example from the script of the classic film, "The Birds" (1963) by a well-known film director in Hollywood, A. Hitchcock, which illustrates the use of storyboard in the production of the film.

This script has a well-defined format. For each scene, a screenplay consists of a scene heading followed by a piece of narrative description. The scene heading often has a scene number and defines shooting positions, the location and time of day of each scene. The narrative description describes what the camera can see and the actions of the casts in the scene. It may contain information about lighting, sound, and environmental conditions such as rain, snow and wind. For instance, consider the following scene description excerpted from the screenplay, "The Birds":

*There is not a sign of activity as the boat drifts just a little closer. As Melanie watches, the front door opens and a woman comes out, walks to a red pickup truck, starts the engine. A little girl comes out of the house, goes to the truck, gets in. The*

104. MEDIUM SHOT – MELANIE

picking up paddle from decks beginning to paddle in toward dock.

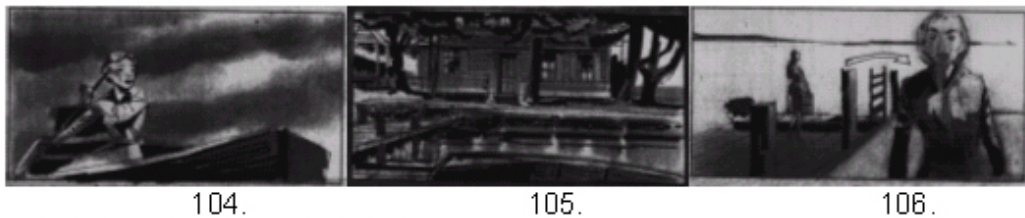
105. FULL SHOT - THE BOAT

edging in toward the dock. Closer, closer, Melanie puts down the paddle. The boat drifts in.

106. MEDIUM SHOT – MELANIE

leaping ashore, tying up the boats reaching down for the cage. She turns to look over her shoulder.

Based on the descriptions, the artist drew the following storyboards:



from which the following actual shots were made:



Figure 1: An example of using storyboard in movie production

*woman shouts something to a man - Mitch Brenner, probably, though it is difficult to tell from the distance – and he comes over to the truck. The truck grinds into gear, goes around the turnabout, and heads down the road away from the farm, a huge cloud of dust behind it. The farm is still again. Mitch stands looking after the truck for a moment, and then begins walking up toward the barn in the distance.*

All actions associated with the characters come from verbs such as *watches, opens and comes [out]*, etc.. Verbs can sometimes be modified by the following phrases and adverbs to describe the manner in which the characters perform their actions, e.g., *Mitch stands looking after the truck.*

In this paper, we base on the verb meanings in English sentences to deliver a story and discuss potential applications of ScriptViz. It is important to recognize that scene settings and the motions of the characters in the scene are well-defined in screenplays, our agents need only to carry out the described actions.

## 3 ScriptViz Architecture

### 3.1 Overview

ScriptViz has a user-friendly front-end window for the user to enter screenplay scripts using a natural language. The front-end window consists of three parts: a text-input box, a script display window, and a Virtual Stage window. The scripts are processed in real time, from which scenes and animated characters are generated automatically and shown in the Virtual Stage next to the script window. In addition, ScriptViz allows the user to choose where and how to look at the scene. After the animated scene has been completed, the user can move the characters and rearrange the props in the Virtual Stage by simply clicking and dragging the graphics objects.

- *Accept natural English sentences as input.* Script-like languages or embedded-command text inputs often make it difficult for users to produce work productively. In ScriptViz we apply technologies in natural language understanding to process English text input.
- *Realistic virtual characters with natural motions and emotions.* To achieve high realism, virtual actors should look as realistic as possible and be able to perform various natural body motions and express emotions.
- *Real-time animation generation and graphics rendering.* Natural language processing and the actor's motion generation are implemented in real time.

Broadly, ScriptViz (v.1.0) currently has three interacting modules:

- A module consists of processes for understanding well-formed sentences, e.g., extracting the semantic information of the sentences. These processes include parsing and interpretation.

- A module consists of a high-level incremental planner and an object-specific reasoner (OSR) [5]. It constructs detailed plans for relevant agents according to the extracted meaning of an input sentence.
- A scene generator triggers agents to perform the described actions and generates scenes in real time.

### 3.2 Natural Language Understanding

An efficient and robust parser is essential to any Natural Language Processing (NLP) system [1]. We use the Apple Pie parser [8] that is based on the Penn TreeBank’s syntactically bracketed corpus. The parser is a bottom-up probabilistic chart parser which finds the parse tree with the best score by the best-first search algorithm. The parser extracts the syntactical structures from the input sentences. Based on the structural information, the parser can resolve the subjects and the objects, and interpret their meanings, e.g., actions involved, according to the verbs and their associated adverbs.

Let us consider a simple sentence with a subject, verb and an object.

Sentence: “*Paul kicks the ball.*”

Parsed Result: (*S (NPL (NNP Paul)) (VP (VBZ kicks) (NPL (DT the) (NN ball))) (. - PERIOD-)*)

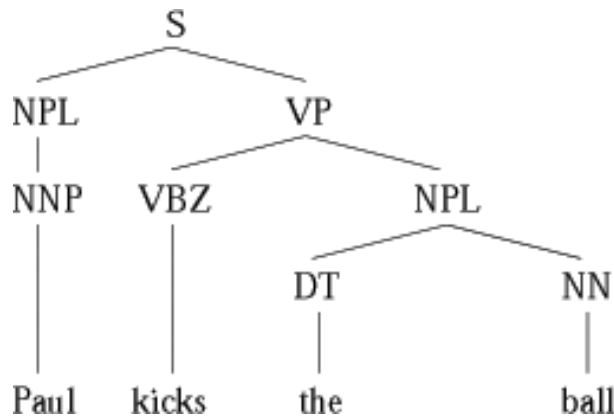


Figure 2: A parse tree for the sentence “*Paul kicks the ball.*”

From this sentence, it is easy to deduce that:

- The name of the action is *kick*.
- *Paul* is the subject of this sentence and is the agent who triggers the *kick* action.
- *A ball* in the scene is being kicked.

Now, let us take a look at another example.

Sentence: “Ann looks very angry and walks away.”

Parsed Result: (S (SS (NPL (NNP Ann)) (VP (VBZ looks) (ADJP (RB very) (JJ angry)))) (CC and) (SS (VP (VBZ walks) (ADVP (RB away)))) (. -PERIOD-))

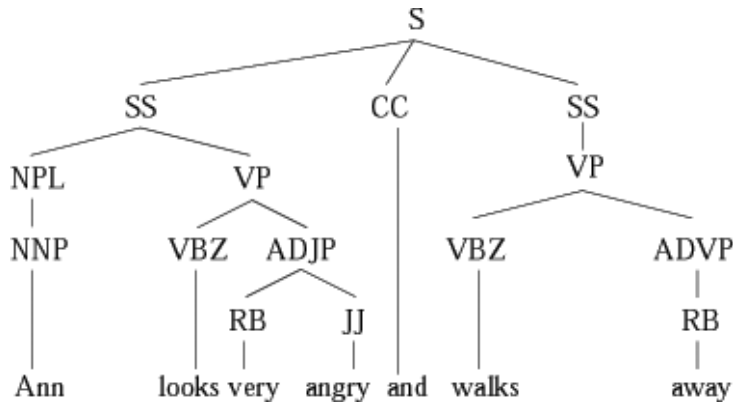


Figure 3: A parse tree for the sentence “Ann looks very angry and walks away.”

This sentence is complex because it has one subject and two verb phrases. The parser treats the sentence as having two clauses separated by a coordinating conjunction ‘and’. The following information can be deduced:

- There is an agent known as Ann, in the scene.
- *looks very angry* - triggers a change of emotion.
- *walks away* - triggers the agent, Ann, to start walking after the change of emotion.

After the sentence’s structure has been obtained, we can extract its meaning by identifying the verbs in the sentence. Then we can construct plans for the relevant agents to accomplish the task. We now turn our attention to how the planner and the OSR work in more detail.

### 3.3 High-Level Planning

The goal of this module is to convert the semantic information of a sentence into a plan of actions. Each plan describes only one high-level task and may involve a number of steps in order to accomplish the task. Each of these steps is an action primitive and is performed sequentially. Since our system processes the user’s input and generates the animated graphics in real time, the user will be able to see the animated graphics almost immediately after she has input a sentence. Based on the input, the high-level planner constructs the entire plan at once. The system ensures that states of the virtual world will not be altered while the agents in the scene are performing the required actions. As a result, it needs not to monitor the states of the virtual world and to update the plan as the agents performing the task, which makes the computation very efficient.

ScriptViz is entirely different from the simulation systems developed for autonomous agents such as that in [4], in the sense that in such systems, the agent's actions are not foreseeable and cannot be planned ahead, and the agent needs to revise its plans from time to time according to the current state of the virtual environments, its feelings and intentions. This is also one of the major reasons why ScriptViz is more practical and realistically achievable.

The planning process in our system can be categorized into four phases:

- Plan outline: The high-level planner retrieves from its library a plan outline based on the meaning of the input sentence and the objects involved in this action.
- Perform object resolution: The high-level planner attempts to resolve the subject and object in a sentence with the objects in the current scene.
- Feasibility test: It determines whether or not the requested action can be performed by the desired agents. For example, Is there a path for an agent to reach an object? or, Is the agent capable of performing the required action?
- Plan construction: Based on the current state of the virtual world, the system constructs a plan to accomplish the task. This process generates one or more Parameterized Action Representation (PAR) [3, 2] that specifies the steps to carry out a high-level command.

### 3.3.1 Object Resolution

When the user inputs a sentence, the parser immediately generates a parsed tree. The high-level planner needs to consult the perception module for the state of the virtual world. Let us consider the following sentence.

*John feels very hungry, so he picks an apple and eats it.*

The first clause tells us that there is an agent known as John in the scene. Since John is a proper noun and each agent has a name, the planner immediately knows which agent *John* is referring to, and so it knows the agent's type and position. In the second clause, it has a pronoun, *he*, and a common noun, *apple*. For pronouns, the high-level planner always looks at the previous clauses or sentences for clues to determine which agent or object a pronoun is referring to. In this case, since the planner already knows that there is an agent known as John, and it is of type human (male), so the planner infers that the pronoun *he* is referring to John. The pronoun, *it*, in the third clause can be resolved similarly. For common nouns, e.g., *apple* in this example, the planner needs to look at the type of each object in the scene to determine whether there is an object of type *apple*. One thing to note is that if there are many instances of *apple* in the current scene, the planner is not able to figure out which object this apple is referring to. By default, it chooses the instance which is the closest to the agent, *John*, instead.

After OSR has determined the types of agent and object, OSR has sufficient information to perform high-level planning. We discuss the functions of OSR in the next section.

The goals of the object-specific reasoner (OSR) are: (1) it determines whether or not a task can be performed by an agent, (2) it constructs and refines a plan to execute the action task. We discuss these points separately in the next two sections.

### 3.3.2 Feasibility Test

As discussed above, the primary inputs to OSR are the types of the objects involved in an action-task and a plan outline that provides a detailed task description. Since for most action-tasks, the interaction is performed between an agent and an object that can also be another agent. The OSR first checks the agent's type against the ones specified in the outline. If successful, the OSR then proceeds to check whether the task permits the agent to interact with the object. Finally, based on the state information, OSR examines whether this action is feasible and selects an appropriate plan from the outline.

Let us consider an example, the task is **FEED(John, Billy)**. For the action to be performed, OSR first verifies that John is a human, and Billy is an animal. Secondly, it has to compute the distance between Billy and John. If John cannot reach Billy, and there is a path for John to walk to Billy, then the OSR needs to insert a plan to trigger John to walk to Billy before the required actions can be performed. In addition, the type of the animal needs to be considered in plan selection since feeding a bird is different from feeding a dog: Feeding a bird may require John only to stretch his arm forward, whereas to feed a dog, John may have to kneel down before he stretches his arm.

### 3.3.3 Plan Construction

After the OSR has selected a plan, it refines the plan and binds parameters of the motion primitives based on the state information. The OSR presents the action information in the parameterized action representation (PAR) to bridge the gap between the natural language and animations [3, 2]. In our current system, we use a simplified version of PAR as shown in Figure 4 for simplicity. The PAR specifies the agent of the action as well as any relevant objects and information about the path, location, manner, and the purpose of a particular action.

To illustrate the idea, we have created an example plan for a high-level task, **WALK\_TO(John, Susanna)**. The PAR for *WALK\_TO* specifies that:

- *John* and *Susanna* are the participants of this action.
- Since this action involves translation and rotation of the agent, *John*.
- Path specifies the displacement of the agent in order to accomplish this action: In this case, the agent, *John*, needs to move from (30,0,0) to (0,0,0) in the 3D space.
- Manner is used to specify whether there is any additional constraint for this action, e.g., *slowly*.

## PAR

<b>participants:</b>	<b>agent:</b>	<i>AGENT</i>	
	<b>objects:</b>	<i>OBJECT</i>	<b>list</b>
<b>core semantics:</b>			
	<b>object:</b>	<i>OBJECT</i>	
	<b>caused:</b>	<i>BOOLEAN</i>	
<b>motion:</b>	<b>translational:</b>	<i>BOOLEAN</i>	
	<b>rotational:</b>	<i>BOOLEAN</i>	
	<b>direction:</b>	<i>DIRECTION</i>	
<b>path:</b>	<b>start:</b>	<i>LOCATION</i>	
	<b>end:</b>	<i>LOCATION</i>	
	<b>distance:</b>	<i>LENGTH</i>	
<b>manner:</b>		<i>MANNER</i>	
<b>subactions:</b>	<i>PAR</i>		<b>constraint-graph</b>
<b>previous action:</b>	<i>PAR</i>		
<b>next action:</b>	<i>PAR</i>		
<b>parent action:</b>	<i>PAR</i>		

Figure 4: A simplified version of PAR

- Subactions represents the breakdown of the action into its sub-steps: *ROTATE* and *GOTO* that are performed sequentially.
- Parent action is the action of which the particular action is a sub-step. It can be Nil.
- Previous and Next action are action done immediately before or after this action.

### 3.4 Scene Generation

When a plan arrives, the scene generator examines the content of the plan and forwards the plan to the relevant agents for animation. In addition, it also updates the state information as the agents move and manipulate objects in the virtual world, thus it provides a “sensory” feedback to the high-level planner and the OSR for incremental action planning.

One important feature of our scene generator is that each animated agent has a hierarchical structure of “bubbles”; and each bubble is an object. Each of this bubbles can be decomposed into primitives, which can be a motion-capture data file or a graphics file in a database. This hierarchical structure of bubbles has several advantages. Firstly, each bubble can be easily added or removed from its parents, enable or disable an agent applying such features in scene generation. Secondly, each bubble is capable of knowing which primitives are available and providing methods to blend the primitives to achieve smooth transition using techniques described in [7] and to make the appearance of an agent look natural [6]. Thirdly, only the bubble’s interface is visible. The details in implementation are hidden from all other bubbles. This makes it easier for maintenance and reduces various code coupling. In addition, it encourages code reusability.

## 4 Implementation

We have implemented our platform entirely in Java and used an OpenGL binding for Java, GL4Java<sup>1</sup>, to render the animated graphics. The system was tested on two machines. One is a Dell Pentium Workstation PWS530 2.2GHz equipped with a 3DLabs Wildcat III 6110 video card and 1GB physical memory. Another one is a Intel Pentium 2 300MHz Linux server with only 64MB physical memory. To achieve high realism, we used figures with very high polygon counts: on average about 19000 polygons for each human figure and about 10000 polygons for each animal figure. For a scene that involves a man, a woman, a dog and a tree, the system can generate animation at 5 frames per second on the Dell workstation. Complex scenes with many objects and agents, figures with high polygon counts and the Java performance problem may result in a poor system performance. However, we can always increase the frame rate by reducing polygon counts for each figure and optimizing the drawing routine.

We have written two short stories for evaluation.

---

<sup>1</sup>GL4Java is licensed under the ‘GNU Library General Public License (LGPL)’

1st story - In a national park

*Billy [a dog] runs to Ann.  
She kneels down and feeds the dog.  
They then run to John. Ann kisses him.  
They walk slowly to the tree.*

2nd story - Inside a shop

*Helen walks to Julian. She shouts at him.  
She looks very angry and walks away.  
Julian looks very sad.*

Figure 5 - 7 are snapshots of the two stories. Although the two stories are relatively simple, they have demonstrated the key features of our system. These includes:

- Accept natural language sentences as input.
- Parse the sentences, one at a time, and extract their meanings according to the verb-adverb pairs.
- A high-level planner retrieves from its library a plan outline and performs object-resolution for each given sentence.
- OSR performs feasibility tests and constructs plans to trigger relevant agents to execute the required tasks.
- The simulator coordinates all agents in the scene and updates the state of the world as the agents move and manipulate objects in the virtual world.
- The simulator supports emotion, full-body motion as well as localized body motion and the “bubbles” architecture.
- The simulator provides six standard views, a close up view for each human figure and a camera view. It also allows the camera to be set at any position and orientation in the virtual environment.

## 5 Conclusion

We have presented ScriptViz v.1.0 for visualizing stories in screenplays. The system allows users to experiment and test their screenplays, scene settings and camera locations.



Figure 5: A snapshot for story #1.

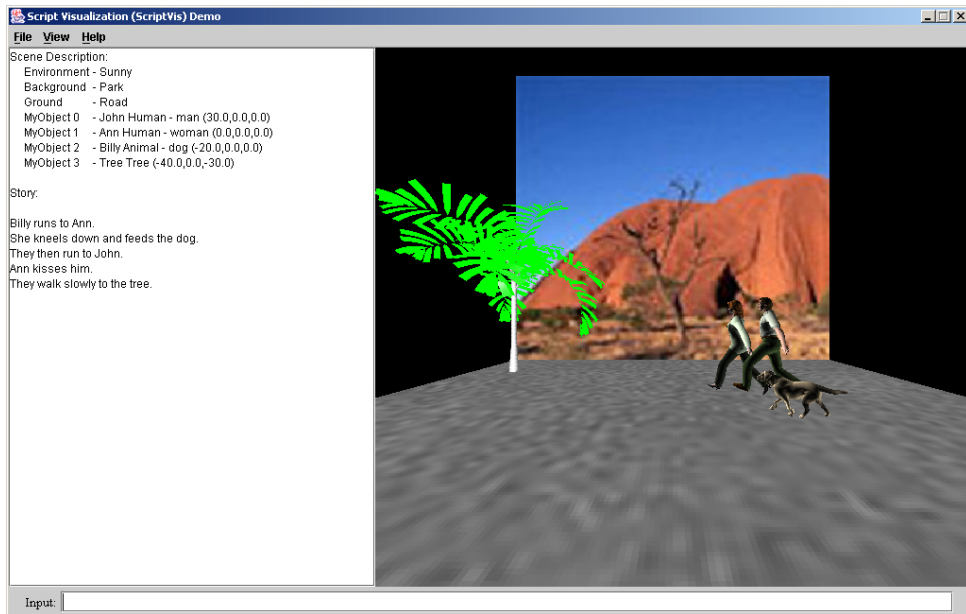


Figure 6: Another snapshot for story #1.

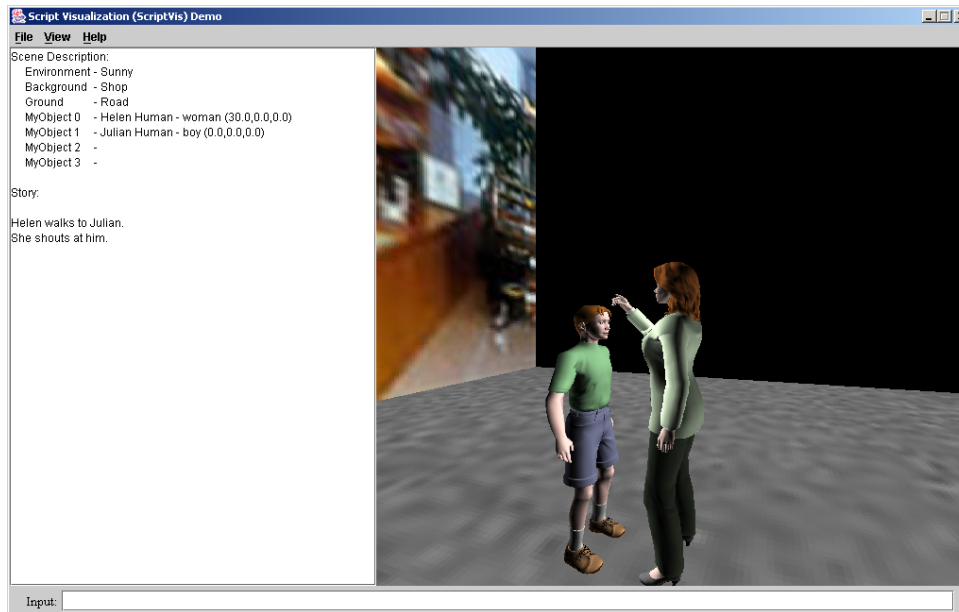


Figure 7: A snapshot for story #2.

For too long we have been relying solely on text as the only the form of presentation during our creative writing. Now with ScriptViz, we are able to view and play with what we are writing and even to collaborate with others. ScriptViz to the writer is like the piano to the composer; it provides an immediate visual feedback to the writer in her writing process.

Although screenplay writers may find ScriptViz a powerful tool, the masses with little computer knowledge will benefit greatly from it, in particular, children. Using ScriptViz in their writing and storytelling, children will find that writing is no longer a lonely and tedious process, rather, it can actually be fun and more enjoyable.

Currently, we are adding a networking module to make the application accessible on the Web. Further improvements are now underway.

## References

- [1] J. Allen, *Natural Language Understanding*, Benjamin/Cummings Publishing, San Francisco, 1995.
- [2] N. I. Badler, M. S. Palmer and R. Bindiganavale, "Animation Control for real-time virtual humans," *Communications of the ACM*, 42(7):65–73, 1999.
- [3] J. C. Bourne, *Generating effective natural language instructions based on agent expertise*, Ph.D. Dissertation, Department of Computer and Information Science, University of Pennsylvania, 1999.

- [4] E. de Sevin, M. Kallmann and D. Thalmann, "Towards Real Time Virtual Human Life Simulations," *Computer Graphics Internation (CGI)*, 2001, Hong Kong.
- [5] C. Geib, L. Levison and M. B. Moore, *SodaJack: An Architecture for Agents that Search for and Manipulate Objects*, Technical Report MS-CIS-94-13/LincLab 265, Dept. of Computer and Information Science, University of Pennsylvania, 1994.
- [6] K. Perlin and A. Goldberg, "Improv: A System for Scripting Interactive Actors in Virtual Worlds," *Computer Graphics (Proc. Siggraph 96)*, ACM Press, New York, Aug. 1996, pp. 205-216.
- [7] C. Rose, M. F. Cohen and B. Bodenheimer, "Verbs and Adverbs: Multidimensional Motion Interpolation," *IEEE Computer Graphics and Applications*, vol. 18, issue: 5, 1998, pp. 32-40.
- [8] S. Sekine, *Corpus-based Parsing and Sublanguage Studies*, Ph.D. Dissertation, Department of Computer Science, New York University, 1998.
- [9] B. Webber, N. Badler, B. Di Eugenio, C. Geib, L. Levison and M. Moore, "Instructions, Intentions and Expectations." *Artificial Intelligence Journal*, 73, 253-269, 1995.