M.I.T Media Laboratory Perceptual Computing Section Technical Report No. 474
(to appear in Proc. ICVS'99) – October 1998

Using Computer Vision to Control a Reactive Computer Graphics Character in a Theater Play

Claudio Pinhanez **Aaron Bobick**
20 Ames St. – Room E15-368C – Cambridge, MA 02139
{pinhanez,bobick}@media.mit.edu

Abstract

It/I is a two-character theater play where the human character *I* is taunted and played by a vision-based, autonomous computerized character — called *It* — which controls computer-graphics, sound, and stage lights. Unlike previous immersive interactive systems, the computer vision system recognizes the human character's actions by considering not only tracking and gestural information and the character's internal variables but also the context provided by the current situation in the story. This paper focuses on a methodology to represent and recognize the human and computer characters' actions that is based on *interval scripts*, a paradigm that uses Allen's temporal primitives to describe the relationships among the different actions and reactions. The system was tested in six public performances held at the MIT Media Laboratory in 1997, when the computer graphics character ran automatically during the 30-minute duration of the play.

1 Introduction

This paper reports our experience in producing a theater play — called *It/I* — which featured, probably for the first time ever, a computer-graphics character autonomously controlled by a computer. All the computer character's reactions during the play were based on the human actor's actions as detected by a 3-camera visual segmentation system invariant to lighting changes (based on [10]).

The play *It/I* was written considering the sensory limitations of computer vision. The actions of the human character — called *I* — were restricted to those that the computer could recognize automatically through image processing and computer vision techniques. In many ways, the understanding of the world by the computer character — named *It* — reflects the state-of-art of real-time automatic vision: the character's reaction is mostly based on tracking *I*'s movements and position and on the recognition of some specific gestures (using techniques similar to [7]).

However, the result of the low-level tracking system is considerably augmented by incorporating knowledge about the expected actions to happen in each moment of the play. Unlike other vision-based interactive systems (e.g. [14]), the actor's position and gestures are interpreted according to the current context provided by the script. When *I* moves towards one the stage screens, that is interpreted as "paying attention to the computer character" in the initial moments of one scene, but, some time later, as "trying to catch the computer character".

The main focus of this paper is on a paradigm called *interval scripts* to represent both the contextual elements required to recognize high-level *actions* (as defined by Bobick in [4]) and the behavior of the computer character. In this particular application — as well as in other interactive, immersive spaces like [5] — the context is largely set up by the actions of the computerized characters, justifying thus that the context for action recognition and the computer behavior to be unified in a sole structure or script.

Interval scripts have been first proposed in [23], and constitute a scripting paradigm rooted in the AI technique of temporal constraint propagation to enhance action recognition and context switching. The research presented in this paper extends the initial proposal of interval scripts by providing a well-defined scripting language and by using the paradigm to represent context for vision-based systems.

1.1 *It/I*: a Computer Theater Play

Computer theater is a term referring to live theatrical performances involving the active use of computers in the artistic process. Pinhanez [19] details the concept of computer theater, the origins of the term, and related works.

Our research in computer theater has concentrated on building automatic, story-aware computer-actors able to interact with human actors on camera-monitored stages. However, automatic control must not mean pre-timed response: computer-actors should be built as reactive autonomous systems that sense the world, consider the current place in the script, and find the appropriate line of action. Otherwise the magic of live performance is lost as the actors are not responsive to each other or to the audience.

An interesting aspect of having autonomous computer actors on stage is that we can have audience interaction in a truly novel way: if a character is controlled automatically by computers, it is possible to transform the stage into an interactive space where members of the audience can re-enact the story of the play in the role of the main characters.

With these ideas in mind, one of the authors of this paper, Claudio Pinhanez, wrote the computer theater play *It/I*. The play is a pantomime where one of the characters, *It*, has a non-human body composed of CG-objects projected on screens (fig. 1). The objects are used to play with the human character, *I*. *It* "speaks" through images and videos projected on the screens, through sound played on stage speakers, and through the stage lights.

Figure 1: Scene from *It/I*. The computer graphics object on the screen is autonomously controlled by the computer character *It*.

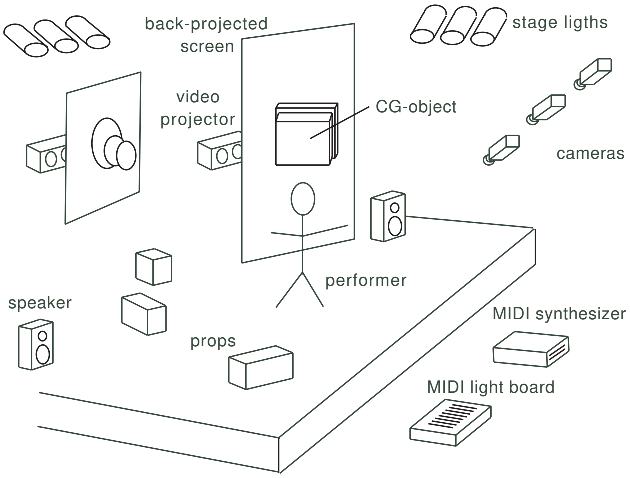

Figure 2: Diagram of the physical structure employed in *It/I*.

Figure 2 depicts a diagram of the different components of the physical setup of *It/I*. The sensor system is composed by three cameras rigged in front of the stage. The computer controls different output devices: two large back-projected screens; speakers connected to a MIDI-synthesizer; and stage lights controlled by a MIDI light-board.

The play is composed of four scenes, each being a repetition of a basic cycle where *I* is lured by *It*, is played with, gets frustrated, quits, and is punished for quitting. For example, the second scene of the play starts with *I* sitting on the stage, indifferent to everything, bathed by blue light. To attract his attention, *It* projects a vivid image of a sun on the stage screen. When *I* pays attention to the picture, the image is removed from the screen, the lights change, and a CG-object similar to a photographic camera appears on the other screen, following *I* around. When *I* makes a pose, the camera shutter opens with a

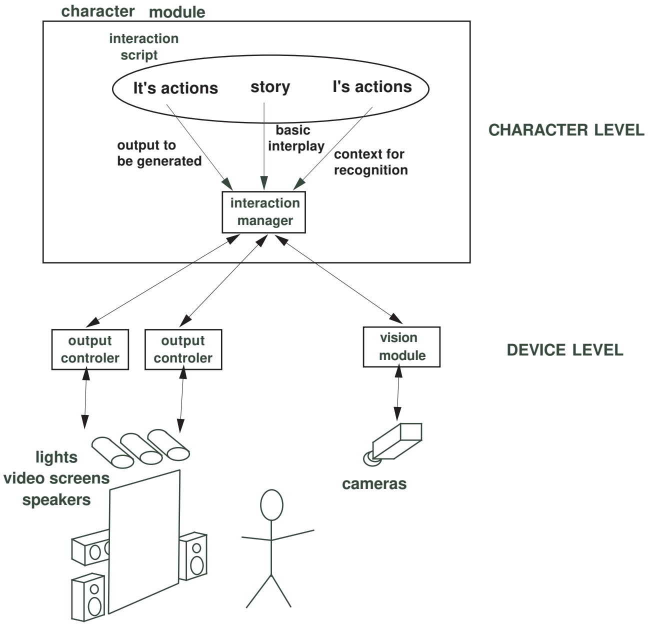

Figure 3: System architecture of *It/I*.

burst of light and the corresponding clicking sound. Following, a CG-television appears on the other screen and, when *I* gets close, it starts to display a slide show composed by silhouette images "taken" by the camera. After some pictures are shown, the camera "calls" *I* to take another picture. This cycle is repeated until *I* refuses to continue to play with the machine and remains in front of the television; this refusal provokes an irate reaction from *It*, which throws CG-blocks at *I* while storming the stage with lights and playing harsh loud noises.

The above segment exemplifies the complexity of the interaction in a typical scene of *It/I*. The scenes have a complexity level that are quite beyond previous full-body interactive systems. For example, in *ALIVE* [14], although the main character (the dog-like CG-character *Silus*) had quite a complex internal structure, the meaning of the user's gestures remains constant as the interaction proceeds. In addition, *It/I* has a clear dramatic structure not present in most previous interactive, immersive systems as, for instance, the spaces designed by Krueger [11] and Sommerer and Mignonneau [24].

2 System Architecture

Figure 3 displays the control architecture used in the performances of *It/I*. It is a 2-layer architecture in which the upper layer contains information specific to the computer character and the bottom layer is comprised of modules directly interacting with the actual input and output devices. This control model is a simplification of a more elaborate 3-layered architecture, called *story-character-device* architecture, or *SCD*, which we are developing considering more difficult problems of controlling stories in immersive environments (see [22]).

As shown in fig. 3, the computer character control system is composed of one active element, the *interaction* *manager*, that process the interaction script (described in detail in [20]). The interaction script contains three types of information: the description of the *story* in terms of actions to be performed by the computer and the human characters; the specification of *It's actions*, that is, what the computer character actually does when trying to perform the actions needed by the story; and the description of how the human actor's movements are recognized according to the current moment in the story, or of *I's actions*.

For instance, in a scene where the computer attracts the attention of the human character by displaying images on the screen, the basic interplay is the goal of attracting *I*'s attention. As part of the description of *It*'s actions there is a method that associates attracting the human character's attention with the displaying of particular images, each moving with a particular trajectory on the screen, accompanied by music and warm light. This is how "attract *I*'s attention" is translated into output to be generated. At the same time, the story sets up the context for recognition of a movement of *I* walking towards the screen as "*I* is paying attention"; in most other situations of the play, such movement is not recognized as so.

We examine in the next section the vision module which uses a non-standard technique to segment the actor from the background. Following, we focus on the discussion of the various components of the computer character module.

3 The Vision System

The vision module shown in fig. 3 performs basic tracking and gesture recognition. The tracking module answers queries about the position (x,y,z coordinates) and size of the actor and three large blocks; the number of persons on stage (none, one, more than one); and the occurrence of the five pre-trained gestures.

In the performance setup we employed a frontal 3-camera stereo system able to segment the actor and the blocks and to compute a silhouette image that is used to track and recognize gestures. The stereo system, based on [10], constructs off-line a depth map of the background — stage, backdrops, and screens. Based on the depth map, it is possible to determine in real-time whether a pixel in the central camera image belongs to the background or to the foreground, in *spite of lighting or background screen changes*. This is an considerable improvement over vision systems based on background subtraction used before in many previous interactive environments [14, 5, 24], since it enables lighting change, an important dramatic element. The segmentation part of the vision system runs currently at 15 frames per second in a SGI Indy 5000 (although during the performances of *It/I* it ran at 8 frames/second). The analog video signals from the three cameras are compressed into a single video stream using a quad-splitter[1] and then digitized into 640x480 images.

Figure 4 shows a typical visual input to the system and the silhouette found. Using the silhouette, a tracking system analyses the different blobs in the image to find the actor. The analysis is performed under assumptions about the continuity and stability of movement, position, and shape of the actor. Smaller blobs are labeled as blocks only when isolated from the person's silhouette.

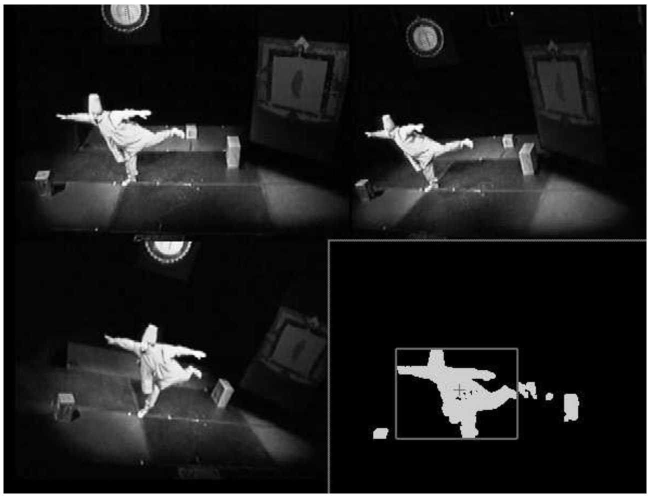

Figure 4: The 3-camera input to the vision module and the computed silhouette of the human actor (enclosed in the rectangle).

The vision system is trained to recognize the five different gestures shown in fig. 5. To perform gesture recognition we employ a simplification of the technique described by Davis and Bobick in [7]. During the performances we also employed manual detection of some of the gestures that could not be detected by the system with enough reliability. We had to resort to manual detection due to an absolute lack of time to improve the reliability to performance levels (which are very high, since during a performance accuracy is essential to avoid confusion in the actor or breaking the *suspension of disbelief* in the audience). Because the recognition of this type of gesture has already been demonstrated in computer vision (see [7]) and in real-time environments [5], we believe that there are no real technical obstacles for a fully autonomous run of the play.

4 From Tracking to Action Recognition

As described above, the vision module outputs only the position of the actor and the occurrence of some pre-defined gestures. How can this information be used by an interactive system based on a story script whose primitives refer to high-level actions such as "paying attention", "refusing to play with the machine", "posing for a picture"?

The fundamental problem has been addressed by Bobick [4] who distinguishes three categories of problems in recognizing human motion, according to an increasing use of knowledge: *movements*, *activities*, and *actions*. Basically, a *movement* is "... a space-time trajectory in some configuration space." (as defined in [4]) whose recognition is independent of contextual information, except for viewing conditions. An *activity* is a sequence of movements that can be recognized based on the statistical properties of their temporal relationships. Finally, an *action* is a movement or set of movements in a context, and its recognition requires the inclusion of knowledge about the context and

[1] A video quad-splitter takes 4 input analog video streams and produces a single signal composed of the four images in half-size and tiled.

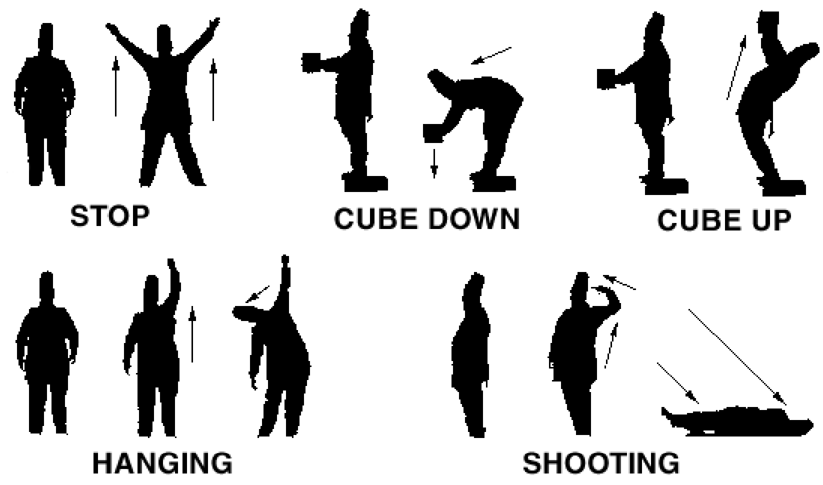

Figure 5: The 5 gestures used in *It/I*.

the domain and the hypothesizing and verification of goal achievement.

Most attempts of performing action recognition have relied in choosing very restricted domains where the needed contextual knowledge can be codified without explicit inclusion of knowledge (for example, [15, 17]). Our approach in the *It/I* project differs from those because the action context is obtained from the story: different movements are recognized as different actions according to when they happen. For example, if the actor gets close to the screen when the computer character is showing attractive images, the movement is labeled as "paying attention"; or if the human character does not move immediately after the computer called him to play, the system recognizes the situation as "refusing to play".

Basically, our approach relies on setting up a situation where only some actions are expected, each of them with different spatial-temporal signatures. The idea has been used before in, among others, *The KidsRoom* [5], an interactive story-based bedroom for children. But unlike those cases, the *It/I* project specifically focused on the development of technology to easily and explicitly represent the story, the context, and the structure of the actions to be recognized, through the use of *interval scripts* as described in the next section.

5 Script Representation

The major technical novelty in *It/I* is the scripting language used to describe the story, the interaction with the human actor, the actions of the human actor, and the behavior of *It*. The 30-minute interaction between *It* and *I* is written as an *interval script*, a language for interaction based on the concept of time intervals and temporal relationships. Interval scripts have been first proposed by Pinhanez, Mase, and Bobick in [23], but in *It/I* we employed a new, revised, and improved version of the concept.

Previous scripting languages (for example, *Director* [13], or [6, 9]) lack appropriate ways to represent the duration and complexity of human action in immersive environments: hidden in the structure is an assumption that actions are pin-point events in time (coming from the typical point-and-click interfaces those languages are designed for) or a simple sequence of basic commands.

Also, unlike some previous reactive autonomous characters like *Silas* [3] or *NeuroBaby* [25], the interaction script of *It/I* contains the story of the play and *It*'s role in it. In this sense, *It* is more a *computer-actor* (as defined in [19]) than an autonomous creature. Specifically, *It* is pro-active in relation to a story, instead of reactive to human interaction or to internalized, creature-like goals [3]. Pro-active computer characters require scripting methods that allow representation of complex story patterns, including parallel actions, multiple paths, and some sense of cause and consequence.

5.1 Interval Scripts

An *interval script* associates a temporal interval with every action in the script. During run-time a label, `past`, `now`, or `fut`, or a disjunction of them, is dynamically assigned to each interval corresponding to the situations where the action has occurred, is occurring, or have not yet occurred, characterizing what we call the `PNF`-*state of the interval.*

The interval script paradigm allows two modes of definition of actions: a *descriptive* mode and a *constraint-based* mode. In the descriptive mode, the semantics of each interval is defined by three functions: `STATE`, a method to compute the current `PNF`-state of the interval; `START`, a function to start the action or sensing corresponding to the interval; and `STOP`, a function to end it. In the constraint-based mode, temporal constraints can be defined between different intervals, limiting their possible `PNF`-states during the run of script.

An interval script is a computer file containing the description of each action and statements about how the intervals are related temporally. Interval script files are con-

```
"bring sun image"=
{ START:
   [>
    ScreenCoord centerSmall (0.0,0.0,-1.0);
    ScreenCoord startFrom (0.0,0.0,-40.0);
    leftScreen.move (sunimageId,centerSmall,
          startFrom,NULL,0,40,"decrescendo");
   <].
  STOP:
   [> leftScreen.stopMove (sunimageId);<].
  STATE:
   [> return leftScreen.moveState (); <].
}.
"remove sun image"=
{ .... (similar to "bring sun image") .... }.
"sun image on screen"={}.
"bring sun image" START
   "sun image on screen".
"remove sun image" FINISH
   "sun image on screen".
```

Figure 6: Example of an interval script from the first scene of *It/I.* The five intervals described here control the movements of an image of the sun on the stage screen.

```
"I is close to big screen"=
{ STATE:
   [> if (close(vision.wherePerson(),
          bigScreeeen.where))
        return NOW; else return PAST-OR-FUTURE;
   <];
}.
"I pays attention to sun image"=
{ STATE: IF "I is close to big screen" IS NOW
     AND "sun image on screen" IS NOW
    ASSERT NOW.
}.
"bring sun image" BEFORE OR MEET OR OVERLAP
  "I pays attention to sun image".
"I pays attention to sun image" BEFORE
  "remove sun image".
```

Figure 7: Part of the interval script from the first scene of *It/I*, showing an example of a mapping of a movement into a higher level action through the use of the story's context.

verted into C++ code through a special compiler that encapsulates the different intervals and their functions into C++ functions attached to the interval control structure. A complete description of the syntax and semantics of interval scripts is beyond the scope of this paper (see [22]). We opted instead to present here some examples which illustrate some of the main features of the paradigm, followed by a basic description of the temporal formalism used in interval scripts and its run-time processing.

Figure 6 shows the definition of three intervals occurring in the beginning of the first scene of *It/I.* They control a short segment of the scene where *It* brings an image of the sun to the screen and moves it away when *I* shows interest on it by getting close to the screen. The first two inter-

```
"sun image scene"=
{ START: TRYTO START "bring sun image".
  STOP: TRYTO START "remove sun image".
  WHEN "I pays attention to sun image" IS NOW
    TRYTO START "remove sun image".
}.
"sun image on screen" FINISH
   "sun image scene".
"I pays attention to sun image" ONLY-DURING
   "sun image scene".
WHEN "sun image scene" IS PAST
   TRYTO RESET "I is close to big screen".
```

Figure 8: Example of definition by an interval script of an interval solely based on previously declared intervals (from the first scene of *It/I*).

vals, `"bring sun image"` and `"remove sun image"` exemplify the descriptive mode of interval scripts. In an interval script, the `START`, `STOP`, and `STATE` functions can either be written as C++ code (inside the special symbols `[>` and `<]`) or as a combination of previously defined intervals. In the case of these two intervals, the interval functions call methods of the C++-object `leftScreen`, sending requests to the CG module of the left screen to move the sun image appropriately.

The interval `"sun image on screen"` defined in fig. 6 is a case where the interval is defined solely as a function of others through temporal constraints. Although the interval's own definition is empty, the two last lines of the fig. 6 define two temporal constraints determining that `"sun image on screen"` is started by `"bring sun image"` and finished by `"remove sun image"`. That is, although the interaction script does not provide any computational definition of the state of the interval `"sun image on screen"`, during run-time its state is determined as a result of the current state of the two other intervals.

Figure 7 contains a simple example of the process of mapping a sensor element into a higher level action through the use of contextual elements from the story. It basically states that if the character *I* is close to the screen while the image of the sun is there, then *I* is paying attention to the screen. Of course under everyday circumstances proximity can not be immediately mapped into attention. However, the development of the play on the stage (or in its user-interactive version) creates surprise elements in this scene that propel the actor or user towards the screen when he is interested in exploring the new image.

This is expressed in the segment of interval scripts depicted in fig. 7. As in the preceding example, the interval `"I is close to big screen"` determines its `PNF`-state by calling a method of a C++-object, `vision`, that queries the vision module at the device level and compares that information to the position of the big screen as obtained through a class function.

The interval `"I pays attention to sun image"` shows another feature of interval scripts: the possibility of defining interval functions based on the state and interval functions of other intervals. As defined by its `STATE` function, the state of the interval is `now` if both `"I is close to big screen"` and `"sun image on screen"` are also `now`, and undetermined otherwise. However, this indeterminacy is constrained temporally by the two following statements that declare that "bring sun image" starts before "I pays attention to sun image", and that the interval happens before "remove sun image". Moreover, the temporal constraints assure that the action "I pays attention to sun image" is recognized only in the context of the presence of the sun image on the screen.

Figure 8 shows a third segment of the script of the first scene of *It/I* in which the previously defined intervals are combined. As noted before, the syntax of interval scripts allows the definition of start and stop functions in terms of previously defined intervals. For example, the `START` and `STOP` functions in fig. 8 are defined by calls to the `START` functions of "bring sun image" and "remove sun image", respectively. When the interval "sun image scene" is set to start (by an interval not shown in the figure), the executed action is to call the `START` function of the "bring sun image" interval, executing its corresponding C++ code.

The definition of interval "sun image scene" also includes a `WHEN` statement that works as a trigger: when the character *I* starts paying attention to the image on the screen, the image is removed. `WHEN` statements are in fact macros of the language, being translated into an interval where the `STATE` and `START` functions are generated automatically such as to perform the "triggering" function. `WHEN` statements proved to be an intuitive way to include event-triggered intervals.

The last line of fig. 8 shows another mechanism allowed by interval scripts, the `RESET` function. To facilitate scripting, intervals can be "recycled" by the invocation of the `RESET` function of an interval. In this example, the end of the interval "sun image scene" triggers via a `WHEN` statement the process of reseting the sensing interval "I is close to big screen"[2].

5.2 Temporal Relationships

One of the major concerns of our work on scripting languages is to provide a structure which can handle complex temporal relationships. Human actions take variable periods of time; also, the order of the performance of actions to achieve a goal is often not strict. In other words, actions — and thus, interaction — can not be fully described neither by events (as *Director* does), nor by simple tree-forking structures as proposed in [18, 2], nor by straight encapsulation such as suggested by structured programming.

We adopted Allen's *interval algebra* [1] as the temporal model of interval scripts. Temporal relationships between intervals can be described as disjunctions of Allen's primitives and easily incorporated into an interval script. For instance, the statement of fig. 6 declares that "I pays attention to sun image" can only happen during "sun image scene", through the macro `ONLY-DURING`[3]. This statement creates a temporal constraint linking the `PNF`-state of the two intervals and preventing, for instance, the "I pays attention to sun image" interval from happening if "sun image scene" is in the `past` state, even if all the conditions listed in the `STATE` declaration apply.

Events, tree-structures, and encapsulated actions and other basic elements from other scripting languages are subsumed by Allen's algebra temporal relationships [1, 16]. Therefore, with explicit declaration of temporal constraints, the interval script paradigm allows the description of complex relations that occur in real interaction, like parallel and mutually exclusive actions, and even causality.

During the compilation of the interval script, those temporal constraints are pre-processed using Allen's path-consistency algorithm [1]. But to achieve speed during run-time constraint propagation, the resulting network is converted into a `PNF`-valued constraint network called a `PNF`-network. In [21] the `PNF`-theory and algorithms are explained in detail in the context of action detection.

5.3 Run-Time Processing

During run-time, the control cycle starts by gathering the state of all sensors and the previous `PNF`-state state of all intervals. An arc-consistency algorithm [12] is then run on the `PNF`-network, what, according to [21] determines an approximation of the minimal domain of the constraint network. Through this reduction process the `PNF`-state of an interval becomes more specific: for example, an interval whose end was undetermined (`past-or-now` state) can go to a a `past` state if, for instance a successor interval (defined through an exclusive *after* constraint between the two), is happening `now`.

Start and stop functions are called directly by intervals, as shown in fig.8, or by the run-time system when certain changes in the `PNF`-state of an interval is detected. Start functions are called automatically whenever an interval changes from the `fut` state to the `now` state. Similarly, stop functions are called by the run-time system if an interval goes from `now` to `past`.

6 Performances and Audience Participation

It/I was produced in the summer/fall of 1997 with direction of Claudio Pinhanez, art direction of Raquel Coelho, and actor Joshua Pritchard. The control of the computer graphics, sound, and lights was performed automatically by a system composed of four SGI workstations. The script of each scene comprised between 100 and 200 intervals describing the behavior of *It* according to the story and the human actor's actions. Supporting the upper level character module were seven lower layer modules running the controllers of specific interfaces, such as the low-level and middle-level vision modules, two CG-generators, movie and sound players, and the interface to the light-board.

The play was performed six times at the MIT Media Laboratory for a total audience of about 500 people. Each performance was followed by an explanation of the workings of the computer-actor. After that members of the audience were invited to go up on stage and play a scene from the play, first in front of the audience, and individually afterwards (see fig. 9).

[2] From a formal point of view, a call to a reset function does not reset but instead produces a new instance of the interval with identical temporal constraints.

[3] The macro `ONLY-DURING` represents the disjunction of the Allen relationships `START`, `FINISH`, `EQUAL`, or `DURING`. That is, `ONLY-DURING` establishes that the interval starts and ends during the other interval, including possibly its extremities.

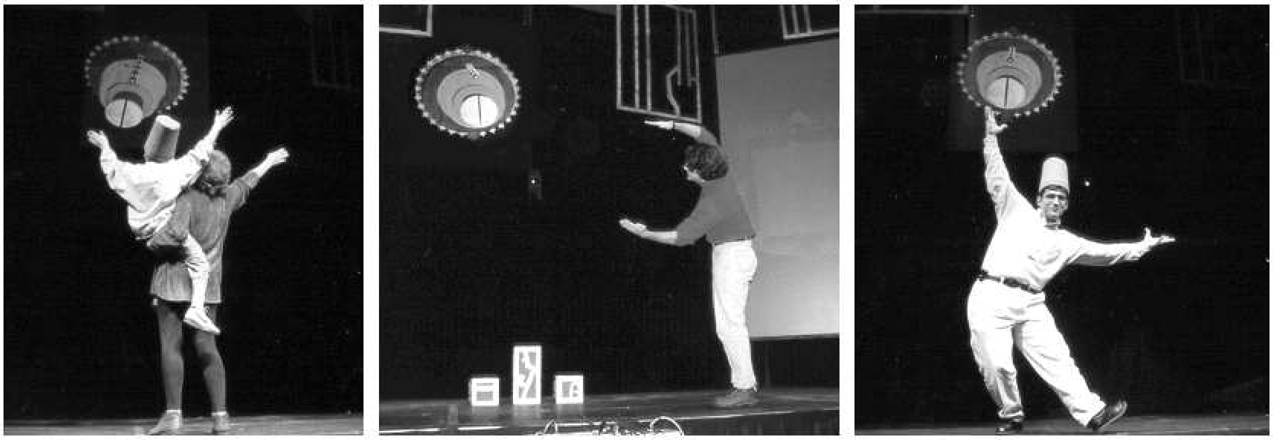

Figure 9: Audience playing with the computer character *It*.

When the spectators went on the stage to re-enact the scene of the play, they displayed a variety of reactions. Some of them could easily remember the sequence of actions in the play and could navigate through the scene without external help, but others were partially confused about what to do. From this experience, we believe that it is necessary to adapt the computer character to interact with non-actors: among the improvements, *It* must be scripted to provide some kind of help or suggestions when it is detected that the user is confused.

7 Conclusion

In this paper we describe how *interval scripts* can be used to describe story-based context to be provided to a vision system. In this paradigm, context switching is achieved naturally as a result of the story development, and different high-level labels for actions are determined according to current and past events. The core mechanism is Pinhanez and Bobick's `PNF`-propagation algorithm for action recognition that can be viewed as a fast specialization of general temporal constraint propagation (see [21]).

The concept was tested in real performances of a computer theater play. Although having an actor instead of an user simplifies the recognition task (the actor knows how an action can be more easily recognized), live performance conditions are extremely intolerant to errors, especially to system mistakes that break or violate the dramatic structure of the story. Moreover, we have also run successfully the system with non-actors from the audience who had quite diverse ways to move and act on the stage.

We see *It/I* as part of a continuing work of understanding and developing technology for story-based, interactive, immersive environments. which started with *SingSong*[23], followed by *The KidsRoom* [5], and after *It/I*, by *PAT*, a virtual aerobics personal trainer [8]. To our knowledge, *It/I* is the first play ever produced involving a character automatically controlled by a computer that was truly interactive. We believe that this was possible only because a flexible scripting system was developed to describe the story interplay and the computer and human characters' actions.

Acknowledgments

The research presented in this paper was partially sponsored by DARPA contract DAAL01-97-K-0103. *It/I* was sponsored by the Digital Life Consortium of the MIT Media Laboratory. We thank to all members of the crew, in particular Prof. Janet Sonenberg, John Liu, Chris Bentzel, Raquel Coelho, Leslie Bondaryk, Freedom Baird, Richard Marcus, Monica Pinhanez, Nathalie van Bockstaele, and Joshua Pritchard.

References

[1] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[2] Joseph Bates, A. Bryan Loyall, and W. Scott Reilly. An architecture for action, emotion, and social behavior. In *Proceedings of the Fourth European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, S. Martino al Cimino, Italy, July 1992.

[3] Bruce Blumberg. *Old Tricks, New Dogs: Ethology and Interactive Creatures.* PhD thesis, M.I.T. Media Arts and Sciences Program, 1996.

[4] Aaron F. Bobick. Movement, activity, and action: The role of knowledge in the perception of motion. *Phil. Trans. Royal Society London B*, 352:1257–1265, 1997.

[5] Aaron F. Bobick, Stephen Intille, Jim Davis, Freedom Baird, Claudio Pinhanez, Lee Campbell, Yuri Ivanov, Arjan Schutte, and Andy Wilson. The KidsRoom: A perceptually-based interactive and immersive story environment. Technical Report 398, M.I.T. Media Laboratory Perceptual Computing Section, November 1996.

[6] M. Cecelia Buchanan and Polle T. Zellweger. Automatic temporal layout mechanisms. In *Proc. of ACM Multimedia'93*, pages 341–350, Ahaheim, California, August 1993.

[7] James W. Davis and A. Bobick. The representation and recognition of human movement using temporal templates. In *Proc. of CVPR'97*, pages 928–934, June 1997.

[8] James W. Davis and Aaron F. Bobick. Virtual PAT: a virtual personal aerobics trainer. Technical Report 436, M.I.T. Media Laboratory Perceptual Computing Section, January 1998.

[9] Rei Hamakawa and Jun Rekimoto. Object composition and playback models for handling multimedia data. In *Proc. of ACM Multimedia'93*, pages 273–281, Ahaheim, California, August 1993.

[10] Yuri Ivanov, Aaron Bobick, and John Liu. Fast lighting independent background subtraction. In *Proc. of the IEEE Workshop on Visual Surveillance – VS'98*, pages 49–55, Bombay, India, January 1998.

[11] Myron W. Krueger. *Artificial Reality II*. Addison-Wesley, 1990.

[12] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[13] MacroMind Inc. *Director's User Manual*. 1990.

[14] Pattie Maes, Trevor Darrell, Bruce Blumberg, and Alex Pentland. The ALIVE system: Full-body interaction with autonomous agents. In *Proc. of the Computer Animation '95 Conference*, Geneva, Switzerland, April 1995.

[15] R. Mann, A. Jepson, and Jeffrey Siskind. The computational perception of scene dynamics. In *Proc. of Fourth European Conference in Computer Vision*, April 1996.

[16] Itay Meiri. Combining qualitative and quantitative constraints in temporal reasoning. *Artificial Intelligence*, 87(1–2):343–385, November 1996.

[17] Hans-Hellmut Nagel. A vision of 'vision and language' comprises action: An example from road traffic. *Artificial Intelligence Review*, 8:189–214, 1995.

[18] Ken Perlin and Athomas Goldberg. Improv: A system for scripting interactive actors in virtual worlds. In *Proc. of SIGGRAPH'96*, August 1996.

[19] Claudio S. Pinhanez. Computer theater. In *Proc. of the Eighth International Symposium on Electronic Arts (ISEA'97)*, Chicago, Illinois, September 1997.

[20] Claudio S. Pinhanez and Aaron F. Bobick. Fast constraint propagation on specialized Allen networks and its application to action recognition and control. Technical report # 456, M.I.T. Media Laboratory Perceptual Computing Section, January 1998.

[21] Claudio S. Pinhanez and Aaron F. Bobick. Human action detection using PNF propagation of temporal constraints. In *Proc. of CVPR'98*, pages 898–904, Santa Barbara, California, June 1998.

[22] Claudio S. Pinhanez and Aaron F. Bobick. "It/I": A theater play featuring an autonomous computer graphics character. Technical report # 455, M.I.T. Media Laboratory Perceptual Computing Section, January 1998.

[23] Claudio S. Pinhanez, Kenji Mase, and Aaron F. Bobick. Interval scripts: A design paradigm for story-based interactive systems. In *CHI'97*, pages 287–294, Atlanta, Georgia, March 1997.

[24] Christa Sommerer and Laurent Mignonneau. Art as a living system. *Leonardo*, 30(5), October 1997.

[25] Naoko Tosa, Hideki Hashimoto, Kaoru Sezaki, Yasuharu Kunii, Toyotoshi Yamada, Kotaro Sabe, Ryosuke Nishino, Hiroshi Harashima, and Fumio Harashima. Network-based neuro-baby with robotic hand. In *Proc. of IJCAI'95 Workshop on Entertainment and AI/Alife*, Montreal, Canada, August 1995.