M.I.T Media Laboratory Perceptual Computing Section Technical Report No. 455
(submitted to SIGGRAPH'98) – January 1998

# "It/I": A Theater Play Featuring an Autonomous Computer Graphics Character

**Claudio Pinhanez** **Aaron Bobick**
20 Ames St. – 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 an autonomous computer-graphics character *It*. We believe that "It/I" is the first play ever produced involving a character reactively controlled by a computer. Autonomous characters can bring a new dimension to the theater experience by enabling the audience to go up on stage after the performance and interact directly with the characters, re-enacting the story of the play. This paper reports the experience and examines technical developments needed for the successful production of "It/I". In particular we describe the *interval script* paradigm used to program the CG-character, the proposed SCD-architecture for story-based interactive systems, and the ActScript language for communication of actions and goals.

## 1 Introduction

While the movie and music industries have been extensively using computers for decades, theaters rarely employ any electronic equipment except for light and sound control, and ticket reservation. We believe that it is far more interesting to use computers as actors, in plays where human and computer characters share the stage — *computer theater*.

Computers have been used to generate "electronic puppets" where a human puppeteer controls a computer graphics character displayed on a stage screen [1]. More novel — and uncommon — is the case where the computer automatically controls the character constituting what we call a *computer-actor*. With autonomous computer-actors is possible to create plays where the audience can take the place of the actors and have a personal experience of the universe depicted on stage.

"It/I" was written, and later produced and performed, to test those concepts and to develop tools for interactive, immersive systems populated by characters and driven by a story. Unlike previous work involving automated characters [16, 2], our work addresses the situation where the actor or user's body is the body of one of characters, and the story is controlled by the computer. This requires the recognition of human actions as they are performed, not as written or clicked-in an interface by a displaced user.

This paper starts by describing the concept of computer theater and how the 30-minute play "It/I" transforms the theatrical stage into an interactive environment. We follow with an analysis of the computer system controlling the character *It*. We begin by discussing the advantages and disadvantages of having centralized control of story as used in the story-character-device (SCD) architecture employed in "It/I".

The two main technical developments proportioned by "It/I" were the *interval script* paradigm used for character and story scripting (based on [21]); and the language for communication between characters, story, and physical devices called ActScript. The description of both paradigms stresses the burden brought into interactive, immersive systems by the the need to recognize human action in a physical environment. We end by describing the performances and the future developments in the play and in the technology.

## 2 Computer Theater

*Computer theater* is a term referring to live theatrical performances involving the active use of computers in the artistic process. Pinhanez [17] contains a detailed exposition about computer theater, the origins of the term, and related works.

Our research has been concentrated in building automatic, semi-autonomous 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, compare to a script stored in memory, and find the appropriate line of action. Otherwise the magic of performance is lost since the actors are not responsive to each other and to the audience.

One of the most interesting aspects 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. In this scenario, the play is expanded from the ritualistic happening of the performance into a universe to be explored and lived by an user [2].

[1] Examples of use of electronic puppets are more common in the context of performance animation. See Protozoa's work (http://protozoa.protozoa.com/) and RiGBy, which appears in performances by D'Cuckoo (http://www.dcuckoo.com/).

[2] We employ the term "user" to differentiate members of the audience from the actors.

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

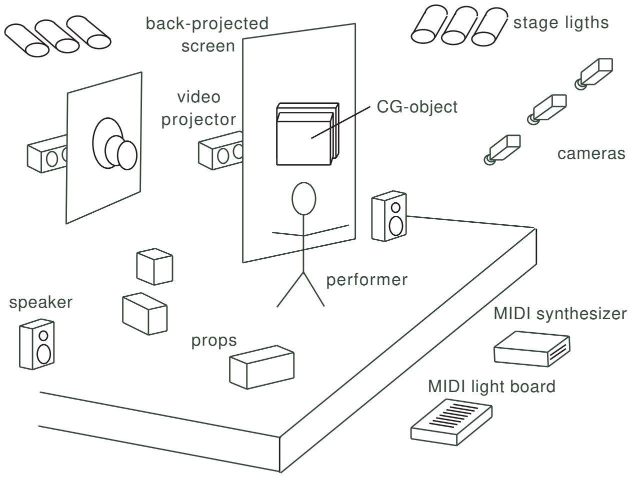


Figure 2: Diagram of the physical structure employed in "It/I".

The performance context brings two simplifying factors to the construction of interactive, immersive systems (see also [19]). First, the human actor knows how to interact with the computer character within the limitations of the sensory apparatus. Second, after watching the performance, the members of the audience transformed in actors have probably learned the basic structure and interaction modes of the play. In this context, system design problems like the learning of the interaction and navigation become less prevalent. Also, the development of the story of the play can be shaped to facilitate the computer recognition of the actor/user activities.

# 3 "It/I": a Computer Theater Play

For testing these ideas one of the authors of this paper, Claudio Pinhanez, wrote the computer theater play "It/I" [3]. The play is a pantomime where one of the characters, *It*, has a non-human body composed of CG-objects projected on screens. The objects are used to play with the human character, *I*. *It* can also "speak" through images and videos projected on the screens, through sound played on stage speakers, and through the stage lights.

The play was written considering the sensory limitations of computer vision. That is, the actions of *I* were restricted to those that the computer could recognize through image processing automatically. In many ways, *It*'s understanding of the world 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 [6]).

Figure 1 shows a picture of a typical scene. The actor is on the stage interacting with the tv-like object projected in the screen behind him. 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: large back-projected screens; speakers connected to a MIDI-synthesizer; and stages lights controlled by a MIDI light board.

## 3.1 The Play

When scene I of the play starts, *I* is sitting at the center of the stage, distracted by the music played off-stage by a pianist. *It* attracts *I*'s attention by displaying a beautiful image of the sun on the left stage screen. When *I* stands up, the image moves away, and a CG-clock appears, running a countdown. *I* tries to hide from the imminent explosion, while *It* projects a movie showing that the clock can be stopped by a gesture. When *I* executes the gesture the clock disappears, immediately being replaced by a new one. Clocks continue to appear, in a faster rate than *I* can stop them. After some time *I* gives up and protects himself from the coming explosions which do not happen. The clocks disappear and the piano music returns.

Notice that the basic sensory capacity in this scene is the recognition of gestures. The information that *I* has stood up triggers the disappearance of the sun image and the bringing of the clocks. The stopping gestures control the turning off of the clocks and their absence causes the explosion.

In scene II, *I* is again instigated to play by an image — a picture of a family. This time, a CG-object similar to a photographic camera appears on the right screen and follows him around. When *I* makes a pose, the camera shutter opens with a burst of light and the corresponding clicking sound. On the other screen a CG-television appears and, when *I* gets close, the television 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 take yet another picture and stays in front of the television, provoking an irate reaction from *It*, which throws CG-blocks into *I* while flickering the lights and playing really loud noise.

The actions of *It* in scene II are solely based on the po-

[3] "It/I" was inspired by three plays by Samuel Beckett, *Act Without Words I*, *Ghost Trio*, and *Waiting for Godot*.

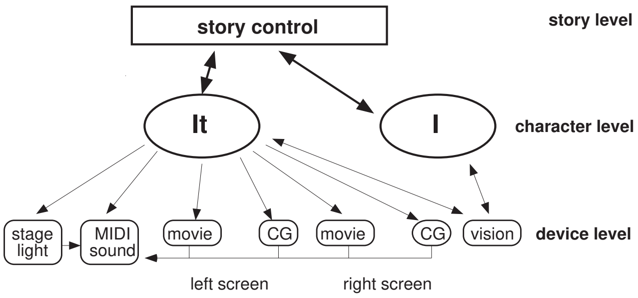


Figure 3: System architecture of "It/I".

sition of *I*. *I*'s interest in *It* is assumed when he gets close to either the camera or the television. For instance, the refusal to continue taking pictures is detected when the camera "calls" three times and *I* does not move from the front of the TV screen. However, although simple, the sensing allows quite flexible interaction in this scene. The number of take-picture-watch-tv cycles is not pre-determined: the human actor decides to refuse to play when he — as a performer — believes the character (and maybe the audience) has reached the emotional potential to do that. Also, there is a lot of room for improvisation in terms of when and how the pictures are taken since the camera clicks only when the actor has been still for some seconds.

In scene III the saga between *It* and *I* continues, with *I* being haunted by an agitated, restless electric switch-like CG-object which always keep itself far from the reaching of *I*'s hands. *I* is then told that he can control the switch's position by making special gestures on the top of blocks. When he finally gets close enough and seems to be able to turn the switch off, *I* discovers that the objects are only projections on a screen, and, in a Beckettian way, tries to hang himself — without success — and provokes another angry reaction from *It*.

In the beginning of scene IV, *I* decides to ignore *It*, which then brings all the objects to the screen, trying to attract *I*'s attention. Finally, given the lack of response, *It* brings the switch to the screen and turns it off, leaving *I* in an empty, silent, lightless world.

# 4 Control Architecture

The production of "It/I" is part of our continuing research on developing technology for interactive, immersive environments which started with *SingSong* [21], followed by *The KidsRoom* [4], and after "It/I", by *PAT*, a virtual aerobics personal trainer [7].

Figure 3 displays the control architecture for "It/I". It is a 3-layer architecture where the upper layer contains the story control module, followed by a layer of systems to control each individual character, and a final layer of modules to deal with the actual input and output devices. This constitutes what we call a *story-character-device architecture*, or simply, a SCD architecture. Notice that in the SCD architecture the characters are semi-autonomous since they receive commands from the story control module although possessing the ability to sense the on-going action and to coordinate the accomplishment of their goals according to the reality of the stage.

It is arguable whether story control should be centralized. Perlin & Goldberg [16] and Bates et. al. [2] built (semi-) autonomous computer-actors with such characteristics. However, in both cases, the story was distributed among the characters, or seen as the natural result of the interaction between the characters and the user.

We believe that centralized story control is very important for an interactive system to achieve successful story development. As pointed by Langer [11] and Murray [14], well-constructed stories require coordination and synchronicity of events and coincidences; also, dramatic actions have to forecast the future, especially in the context of theatrical stories[4]. Such coordination, in our view, is only possible with centralized story control.

The first performances of "It/I" used a system control module that had to collapsed the story and character levels because the protocol for communication was still under development. This was feasible in "It/I" because the character *It*, according to the play, controls the whole environment and to some extent the story itself. We are currently working on separating characters and story control to allow much better characterization of the roles and a simplification in the scripting structure.

# 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, 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.

Previous work on scripting languages can be divided into two types. The first deals with languages for scripting movements and reactions of characters, like the work of Perlin [15], Kalita [10], and Thalman [24]. In these cases, the emphasis is on realistic-looking ways of describing human motion with little provision for scripting interaction and sensory input. The other type corresponds to languages for description of interaction as for example *Director* [12] and the works of Buchanan & Zelllweger [5] and Hamakawa & Rekimoto [8]. Those languages, however, 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-pointed events in time (coming from the typical point-and-click interfaces those languages are designed for) or a simple sequence of basic commands.

## 5.1 Interval Scripts

An *interval script* associates a temporal interval to every action in the script. To each interval a label, `past`, `now`, or `fut` is assigned, corresponding to the situations where the action has occured, is occuring, or have not yet occured, characterizing what we call the `PNF`-*state* of the interval.

An interval script is a computer file containing the description of each action and statements about how the intervals are related temporally. A complete description of the syntax and semantics of interval scripts is beyond the

[4]See Langer [11], chapter 7, for a complex but insightful argumentation about how in theater it is fundamental that actions in the present are determined by the happenings in the future.

scope of this paper. We opted to present an example which illustrates some of the main features of the paradigm.

Figure 4 shows the description of five 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 standing up.

Interval script files are converted into C++ code through a special compiler that encapsulates the different intervals in function calls. 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.

These functions can either be written as C++ code (inside the special symbols `[>` and `<]`) or as a combination of previously defined intervals. For example, in fig.4, the interval `"I is seated"` determines its `PNF`-state by calling a method of the C++-object `vision` that queries the vision module at the device level. `"bring sun image"` and `"remove sun image"` define `START` functions calling methods of the C++-object `leftScreen`, sending requests to the CG module of the left screen to move the sun image. The interval `"minimum time of sun image"` is defined by the macro `TIMER`, a C++-object that takes 10 seconds to make the interval go to the `past` state after the interval is started.

The intervals described so far simply encapsulate C++ code. One of the goals of the design of interval scripts was to have easy ways to build more complex intervals from basic ones. The interval `"sun image scene"` is such a case. The definition of this interval describes event relationships between the intervals defined before using the two `WHEN` statements and the interval's own `START`, `STOP`, and `STATE` functions. The first `WHEN` asserts that when the bringing of the sun image is over the timer `"minimum time of sun image"` is started; the second starts to remove the image when *I* is not seated anymore, provided that the minimum 10-second period of image exposure is completed.

The syntax of interval scripts allows the definition of start and stop functions in terms of previously defined intervals. Two examples are the `START` and `STOP` functions of the interval `"sun image scene"` which are defined by calls to the `START` function of `"bring sun image"` and `"remove sun image"`, respectively. When the `"sun image scene"` interval is set to start (by a `WHEN` statement 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. Interval scripts allow complex combinations of start and stop functions of different intervals when describing higher level functions. The `STATE` of `"sun image scene"` is also defined based on previously defined intervals: if either `"bring sun image"` or `"remove sun image"` are happening `now`, the `PNF`-state of the interval is defined to be `now`; if `"remove sun image"` is over, the state is `past`; otherwise, an undetermined `PNF`-state is assumed as default.

The interval script language also provides mechanisms — not shown here — to recycle the interval associated to an action, enabling easy re-use of action and sensing intervals. A typical example is the `"I is seated"` interval of fig. 4 which is re-initialized (as if it had not happened before) at the beginning of each scene.

```
%%%%%%%%%%%% detection of seating %%%%%%%%%%%%
"I is seated"=
{STATE:
  [> if (vision.hasPersonAttribute (SEATED))
       return NOW; else return PAST-OR-FUTURE;
  <].
}.
%%%%%%%% bringing the image to the screen %%%%
"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 (); <].
}.
%%%%%%%% removing the image to the screen %%%
"remove sun image"=
{   ....  (similar to "bring sun image") ....    }.
%%%%%%timer to inforce permanence of image %%%%
"minimum time of sun image"=TIMER(10.0).
%%%%%%%% the scene with the sun image %%%%%
"sun image scene"=
{
 WHEN "bring sun image" IS PAST
  TRYTO START "minimum time of sun image".
 WHEN "I is seated" IS PAST
   AND "minimum time of sun image" IS PAST
  TRYTO START "remove sun image".
 START: TRYTO START "bring sun image".
 STOP: TRYTO START "remove sun image".
 STATE:
  IF "bring sun image" IS NOW
   OR "remove sun image" IS NOW ASSERT NOW,
  IF "remove sun image" IS PAST ASSERT PAST.
 "bring sun image", "remove sun image",
   "minimum time of sun image"
  ONLY-DURING "sun image scene".
}.
```

Figure 4: Example of an interval script from scene I of "It/I".

### 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 by Perlin's or Bates' works, nor by straight encapsulation such as suggested by structured programming.

We adopted Allen's *interval algebra* [1] as the temporal model of interval scripts. According to this algebra the temporal relationship between two intervals is defined as a disjunction of 13 basic primitives. It is beyond the scope of this paper to detail how the algebra works, our approach for fast temporal reasoning, or how the interval script is actually computed (see [20]). Our objective here is to exemplify how temporal constraints can be incorporated into interval scripts allowing a significant increase in script expressiveness.

The last line of `"sun image scene"` is a statement that declares that `"bring sun image"`, `"remove sun image"`, and `"minimum time of sun image"` can only happen during `"sun image scene"`. This statement creates a temporal constraint linking the `PNF`-state of all these intervals and preventing, for instance, the call of the start function of `"remove sun image"` if `"sun image scene"` is in the `past` state, even if all the conditions listed in the `WHEN` declaration apply.

Events, tree-structures, and encapsulated actions and other basic elements from other scripting languages are subsumed by Allen's algebra temporal relationships. But the interval script paradigm allows also the description of more complex relations that occur in real interaction, like parallel and mutually exclusive actions, and even causality.

The first version of "It/I" employed only one module (programmed as an interval script) for the control of the play. Figure 4 is part of such script, and contains intervals later separated among the story control and the characters' modules. In the final version, `"I is seated"` belongs to the "I" module, `"bring sun image"` and `"remove sun image"` to the "It" module, and `"sun image scene"` to the story control module. In this situation those three modules have to exchange information concerning actions to be and being executed, and characters' goals. This is accomplished using the communication language described in the next section.

## 6 Communicating Actions

In [18], Pinhanez & Bobick revitalized Roger Schank's *conceptualizations* [22] as a formalism to represent action that enables shallow reasoning. We are currently developing this work further into a language, ActScript, able to represent actions, requests, queries, and goals in a physical environment. Unlike previous work in languages for CG characters (for example, [16, 10, 24]), ActScript allows recursive decomposition of actions, specification of complex temporal relationships, and translation to/from natural language[5].

```
(request
 (action "left-cg-module"
  (move (object "camera")
   (direction
    to (location (0.0 0.4 0.5))
    from (location (0.0 3.0 -1.0)))
   (path
    (location (0.0 -0.03 0.0)))
  (when NOW)
  (velocity (special "crescendo"))
  (interval (timed duration 5.0)))))
```

Figure 5: Example of a request for a movement of the clock in ActScript.

```
(request
 (action "it"
  (dosomething (object "i")
   (result
    (action "i"
     (attend (object "eye")
      (direction to (location "camera")))))
  (when NOW))))
```

Figure 6: Request in ActScript for a goal to *It* that calls the attention of *I*.

One of the key features of ActScript is its small number of primitives, notably for the description of action verbs. ActScript employs 10 primitive actions: `PRODUCE`, `ATTEND`, `PROPEL`, `MOVE`, `GRASP`, `PTRANS` (physical transportation of an object), `ATRANS` (attribute transference), `MTRANS` (memory or concept transference), `MBUILD` (memory construction), and the generic action `DOSOMETHING`. Complex descriptions use causal links described by the keywords `RESULT`, `REASON`, and `ENABLE`. It is not feasible to describe the syntax and semantics of ActScript in the scope of this paper. Figures 5, and 6 show examples of a movement and an action goal, respectively, as expressed in ActScript.

Figure 5 is an example of a request from the "It" module to the `"left-cg-module"` to move the computer graphics camera from a determined location to other going through a specified `"path"` position. The movement should take 5 seconds, start immediately, and follow the pre-defined velocity profile curve called `"crescendo"`. Upon receiving this request, the CG module checks if the object is available for movement, and in the positive case, the movement is started. A message — also in ActScript — is sent back

[5] Although we have not ventured in building a natural language translator, we believe that NL translators similar to the ones built by Schank and his team are likely to successfully parse natural languages utterances from/to ActScript due to the language's structural resemblance to Schank's conceptualizations.

to the "It" module in any case to communicate the state of the request. When the movement is finished, another message is sent to "It".

Figure 6 is a higher level request from the story control module communicating a goal to the character *It*. Here, the action requested asks *It* to perform any action that will result in *I* looking towards the camera. The interval script controlling *It* contains an interval that assumes the NOW state when such messages are received. Based on the change of state of that interval, other intervals describing appropriate reactions will be triggered according to the module's perception of the state of the stage. For instance, suppose *I* is close to the left screen: *It* may try to attract *I*'s attention by moving the camera to a position on that screen. That can be accomplished by sending the message in fig. 5 to the module controlling computer graphics on the left screen, where the path coordinates are computed on the fly to make the CG-camera almost touch *I*. Alternatively, the script can determine that *I* is too close to the screen, and a better alternative is to flash the stage lights near the screen area.

Notice that, since all 3-layers of the SCD architecture use the same communication language, it is easy for the story control to detail — if necessary at device level — actions to be performed by the characters. As part of ACTSCRIPT's syntax, the `DOSOMETHING` statement of fig. 6, could be substituted by the movement in fig. 5, plus the `RESULT` declaration. That would communicate not only a goal but a suggestion of how to accomplish it. This is very important, because sometimes story development requires precise movements. As when a theater director goes up on stage and shows an actor how he exactly wants her to walk across the stage or manipulate a prop.

# 7 The Device Modules

In this section we describe two of the device modules. In our opinion they contain interesting technical solutions for the construction of autonomous interactive CG-characters.

## 7.1 The Vision Module

The vision module answers queries about the position (x,y,z coordinates) and size of the actor and four large blocks; about number of persons on stage (none, one, more than one); and queries about the occurrence of the pre-trained gestures.

In the performance setup we employed a frontal 3-camera stereo system that is able to segment the actor and the blocks, computing a silhouette image that is used to track and recognize gestures. The stereo system, based on [9], 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 screen changes.* This is an considerable improvement over vision systems used before in interactive environments ([13, 4, 23]) and enables lighting change, an important element of theatrical productions.

Figure 7 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

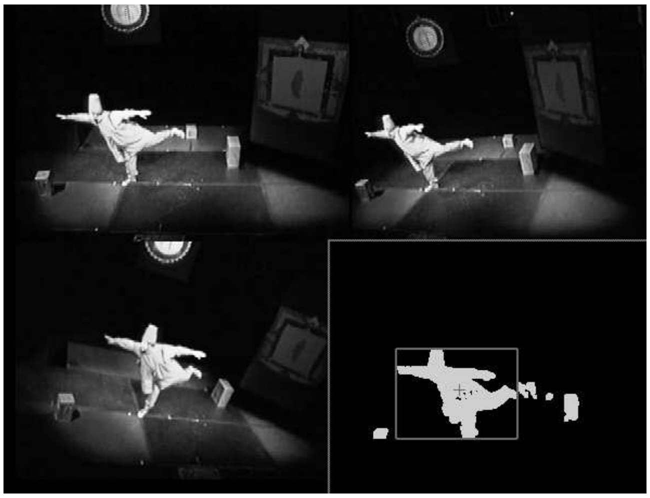

Figure 7: The 3-camera input to the vision module and the computed silhouette of the human actor (enclosed in the rectangle).

the continuity and stability of movement, position, and shape of the actor. Smaller blobs are labeled as blocks.

The vision system is also trained to recognize 5 different static gestures. For this, we employed a simplification of the technique described in [6].

## 7.2 The Computer Graphics Modules

The computer graphics modules control the generation and movement of the different objects and flat images which appear on the stage screens. Each CG module basically processes requests similar to the one displayed in fig. 5 translating them into *Inventor* commands.

In our system, whenever an object or image moves, sound is produced. The importance of sound to enhance image is well known in the movie industry, but, surprisingly, only a few interactive CG characters and environments seem to have explored this facet of animation. Here we are not talking about the inclusion of sound effects in the post-production phase of CG videos, or about synchronizing speech to talking characters. Rather, we are interested in the automatic generation of pre-defined sound effects as the objects move around the world. In the SCD architecture, the decision of using speech happens at the character level, while sound effects related to CG-objects are normally requested directly by the CG module.

We implement sound production as short MIDI-files associated to each object's translational and rotational movements through an *Inventor Engine* object. The MIDI-file is sent to the synthesizer — through a request command to the sound module — according to a threshold function describing when the file must be played. Typical examples of such functions used in "It/I" are: a function that plays the sound periodically while the movement lasts; a function that looks for peaks in the velocity; a function that triggers whenever there is a peak in the derivative of the curvature of the path. The last function is used quite successfully to automatically control swiveling sounds when the CG-camera is performing fast rotational movements.

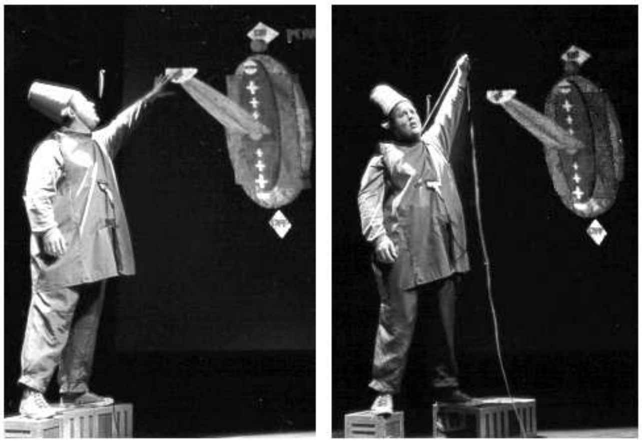

Figure 8: Scene III of "It/I". After discovering that the switch is just an image, *I* tries to hang himself.

## 8 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 (see fig. 8). 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 the audience was invited to go up on stage and play scene II — the scene where *It* plays with a camera and a television, first in front of the audience, and individually afterwards (see fig. 9).

In the performances held in November of 1997 the recognition of gestures was not robust enough and the occurrence of some gestures in two of the four scenes had to be manually communicated to the system. Besides that, the 30-minute performance was completely autonomous, even during the audience part. Since the recognition of this kind of gesture has been demonstrated in computer vision (see [3]) and in real-time environments [4], we believe that there are no technical obstacles for a fully autonomous run of the play.

## 9 Conclusion

"It/I" is part of a continuing work of understanding and developing technology for story-based, interactive, immersive environments. To our knowledge, "It/I" is the first play ever produced involving a character automatically controlled by a computer that was truly interactive.

The performances of "It/I" in November of 1997 were the first phase of our intended work with the play. Currently we are porting the structure to a smaller space where development will proceed. A possible next stage in the project considers setting up the play in a theater or art gallery where performances happen during the evening and the public is invited to visit the space in the following morning or afternoon for an individual, private exploration of the play.

The interval script language is currently being reexamined and improved. We hope to have soon a version of the compiler and a detailed manual available for use by other research groups. We are also studying the incorporation of ACTSCRIPT primitives directly into the interval script language facilitating the communication between modules.

There is still room for further development in ACTSCRIPT and its tools (interpreter, interface). ACTSCRIPT is thought as a general language for communication between physical objects and software components in a physically embedded computer system, and, as such, requires careful design and evaluation.

## 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] Aaron F. Bobick and James W. Davis. An appearance-based representation of action. Technical Report 369, M.I.T. Media Laboratory Perceptual Computing Section, February 1996. To appear in ICPR'96.

[4] Aaron F. Bobick, Jim Davis, Stephen Intille, Freedom Baird, Lee Campbell, Yuri Ivanov, Claudio Pinhanez, Arjan Schutte, and Andy Wilson. Kidsroom: Action recognition in an interactive story environment. Technical Report 398, M.I.T. Media Laboratory Perceptual Computing Section, 1996.

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

[6] 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.

[7] 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.

[8] 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.

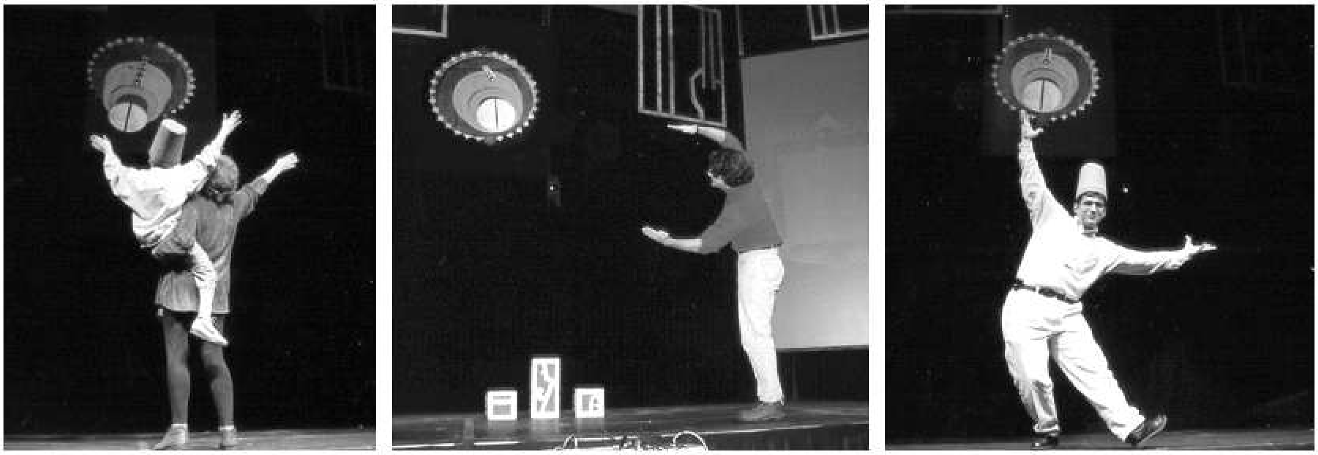

Figure 9: Audience playing with the computer character *It*.


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

[10] Jugal Kumar Kalita. *Natural Language Control of Animation of Task Performance in a Physical Domain.* PhD thesis, University of Pennsylvania, Philadelphia, Pennsylvania, June 1991.

[11] Susanne K. Langer. *Feeling and Form.* Charles Scribner's Sons, New York, New York, 1953.

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

[13] 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.

[14] Janet Murray. *Hamlet on the Holodeck: the Future of Narrative in Cyberspace.* The Free Press, Simon & Schuster, 1997.

[15] Ken Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, 1995.

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

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

[18] Claudio S. Pinhanez and Aaron F. Bobick. Approximate world models: Incorporating qualitative and linguistic information into vision systems. In *AAAI'96*, pages 1116–1123, Portland, Oregon, August 1996.

[19] Claudio S. Pinhanez and Aaron F. Bobick. Computer theater: Stage for action understanding. In *Proc. of the AAAI'96 Workshop on Entertainment and AI/A-Life*, pages 28–33, Portland, Oregon, August 1996.

[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, 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.

[22] Roger C. Schank. Conceptual dependency theory. In *Conceptual Information Processing*, chapter 3, pages 22–82. North-Holland, 1975.

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

[24] Nadia Magnetat Thalmann and Daniel Thalmann. *Synthetic Actors in Computer Generated 3D Films.* Springer-Verlag, Berlin, Germany, 1990.