Improv: A System for Scripting Interactive Actors in Virtual Worlds

Ken Perlin / Athomas Goldberg
Media Research Laboratory
Department of Computer Science
New York University

ABSTRACT

Improv is a system for the creation of real−time behavior−based animated actors. There have been several recent efforts to build network distributed autonomous agents. But in general these efforts do not focus on the author's view. To create rich interactive worlds inhabited by believable animated actors, authors need the proper tools. Im**prov provides tools** to create actors that respond to users and to each other in real−time, with personalities and moods consistent with the author's goals and intentions.

Improv consists of two subsystems. The first subsystem is an Animation Engine that uses procedural techniques to enable authors to create layered, continuous, non−repetitive motions and smooth transitions between them. The second subsystem is a Behavior Engine that enables authors to create sophisticated rules governing how actors communicate, change, and make decisions. The combined system provides an integrated set of tools for authoring the "minds" and "bodies" of interactive actors. The system uses an english−style scripting language so that creative experts who are not primarily programmers can create powerful interactive applications.

INTRODUCTION

Believability And Interaction

Cinema is a medium that can suspend disbelief; the audience enjoys the psychological illusion that fictional characters have an internal life. When this is done properly, these characters can take the audience on a compelling emotional journey. Yet cinema is a linear medium; for any given film, the audience's journey is always the same. Likewise, the experience is inevitably a passive one as the audience's reactions can have no effect on the course of events.

This suspension of disbelief, or believability, does not require realism. For example, millions of people relate to Kermit the Frog and to Bugs Bunny as though they actually exist. Likewise, Bunraku puppet characters can create for their audience a deeply profound and moving psychological experience.

NYU−MRL, 719 Broadway 12th Floor, New York, NY 10003
Fax: (212) 995−4122 Web: http://www.mrl.nyu.edu
Email: perlin@nyu.edu | athomas@mrl.nyu.edu

All of these media have one thing in common. Every moment of the audience's journey is being guided by talented experts, whether an screenwriter and actor/director, a writer/animator, or a playwright and team of puppeteers. These experts use their judgment to maintain a balance: characters must be consistent and recognizable, and must respond to each other appropriately at all times. Otherwise believability is lost.

In contrast, current computer games are non−linear, offering variation and interactivity. While it is possible to create characters for these games that convey a sense of psychological engagement, it is extremely difficult with existing tools.

One limitation is that there is no expert, no actor, director, animator or puppeteer, actually present during the unfolding drama, and so authors using existing techniques are limited by what they can anticipate and produce in advance.

In this paper, we discuss the problem of building believable characters that respond to users and to each other in real−time, with consistent personalities, properly changing moods and without mechanical repetition, while always maintaining an author's goals and intentions. We describe an approach in which actors follow **scripts, sets of au**thor−defined rules governing their behavior, which are used to determine the appropriate animated actions to perform at any given time. We also describe a behavioral architecture that supports author−directed multi−actor coordination as well as run−time control of actor behavior for the creation of user−directed actors or avatars. Next, we describe how the system has been implemented using an "english−style" scripting language and a network distribution model to enable creative experts, who are not primarily programmers, to create powerful interactive applications. Finally, we discuss our experiences with the system and future work.

Related Work

The phrase "Desktop Theater" was coined by Steve Strassman [Strassman91]. His philosophy was quite similar to ours. Yet because his work slightly predated the age of fast graphical workstations, it did not deal with real time visual interaction. But there was already the emphasis on expressive authoring tools for specifying how characters would respond to direction.
characters would respond to direction.
Stephenson also influenced this work. That novel posits a "Metaverse", a future version of the Internet which appears to its participants as a quasi−physical world. Participants are represented by fully articulate human figures, or avatars. Body movements of avatars are computed automatically by the system.

Snow Crash specifically touches on the importance of proper authoring tools for avatars, although it does not describe those tools. Our system takes these notions further, in that it supports autonomous figures that do not directly represent any participant.

Most autonomous actor simulation systems follow the parallel layered intelligence model of [Minsky86], which was partially implemented by the subsumption architecture of [Brooks86] as well as in [Bates92] and [Johnson94]. Several

systems have been developed which share this layered architecture with Imp**rov, yet which solv**e distinctly different problems. The **Jack system of [Bad**ler93] focuses on proper task planning and biomechanical simulation, as does [Hodgins95] The general goal is to produce accurate simulations of biomechanical robots. Similarly, the simulations of Terzopoulis et. al [Terzopoulos94] has simulated autonomous animal behaviors that respond to their environment according to biomechanical rules. Autonomous figure animation has been studied by [Badler91], [Girard85], [Morawetz90] and [Sims94].

The Alive system of [Maes95] and [Blumberg95] focuses on self−organizing embodied agents, which are capable of making inferences and of learning from their experiences. Instead of maximizing an authors ability to express personality, the Alive system use ethological mechanisms to maximize the actor's ability to reorganize its own personality, based on its own perception and accumulated experience.

APPROACH

Improv: An Expert System For Authors

As an authoring system, Improv must provide creative experts with tools for constructing the various aspects of an interactive application. These must be intuitive to use, allow for the creation of rich, compelling content, and produce behavior at run−time which is consistent with the author's vision and intentions. Animated actors must be able to respond to a wide variety of user−interactions, in ways that are both appropriate and non−repetitive. This is complicated by the fact that in applications involving several characters, these actors must be able to work together while faithfully carrying out the author's intentions. The author needs to control the choices an actor makes and how the actors move their bodies.

ARCHITECTURE

The behavior model used by Improv is similar to that proposed by [Blumberg95] in that it consists of geometry that is manipulated in real−time, an Animation Engine which utilizes descriptions of atomic animated actions (such as Walk or Wave) to manipulate the geometry, and a Behavior Engine which is responsible for higher−level capabilities, (such as going to the store, or engaging another actor in a conversation), and decisions about which animations to trigger. In addition, the Behavior Engine maintains the internal model of the actor, representing various aspects of an actor's moods, goals and personality. The Behavior Engine constitutes the mind of the actor. An run−time, an actor's movements and behavior are computed by iterating an update *cycle that alternat*es between the Animation and Behavior Engines.

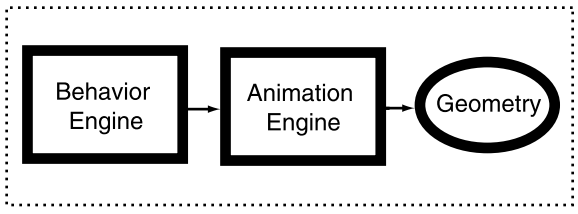

figure 1. The basic architecture for an actor in the run−time system.

ANIMATION ENGINE

The Animation Engine provides tools for generating and interactively blending realistic gestures and motions. This is a generalization of the system presented in [Perlin 95]. Actors are able to move from one animated motion to another in a smooth and natural fashion in real time. Motions can be layered and blended to convey different moods and personalities. The Animation Engine controls the bod*y of the actor*.

Geometry

An animator can build any variety of articulated character. Actors can be given the form of humans, animals, animate objects, or fantasy creatures. An actor consists parts that are connected by rotational joints. The model can be deformable, which is useful for muscle flexing or facial expressions as illustrated in [Chadwick89].

Degrees Of Freedom

Authors specify individual actions in terms of how those actions cause changes over time to each individual degree *of freedom (DOF) i*n the model. The system then combines these DOF values to make smooth transitions and layerings among actions.

There are various types of DOFs that an author can control. The simplest are the three rotational axes between any two connected parts. Examples of this are head turning and knee bending. The author can also simply position a part, such as a hand or a foot. The system automatically does the necessary inverse kinematics to preserve the kinematic chain. From the author's point of view, the x,y,z coordinates of the part are each directly available as a DOF.

The author can also specify part mesh deformations as DOFs. To make a deformation, the author must provide a "deformation target," a version version of the model (or just some parts of the model) in which some vertices have been moved. For each deformation target, the **Improv system detec**ts which vertices have been moved, and builds a data structure containing the x,y,z displacement for each such vertex. For example, if the author has provided a smiling face as a deformation target, then the (s)he can declare SMILE to be a DOF. The author can then specify various values for SMILE between 0. (no smile) and 1. (full smile). The system handles the necessary interpolation between mesh vertices. In the particular case of smiling, the author can also specify negative values for SMILE, to make the face frown.

figure 2. Flexing a deformable mesh.

Continuous Signal Generation

The author defines an action simply as a list of DOFs, together with a range and a time varying expression for each DOF. Most actions are constructed by varying a few DOFs over time via combinations of sine, cosine and coherent noise. For example, sine and cosine signals are used together within actions to impart elliptical rotations.

One of the key ingredients to realism in Improv characters is the ability to apply coherent noise. This mechanism was originally developed for procedural textures [Perlin85][Ebert94]. In the current work it is used in essentially the same way. Using noise in limb movements allows authors to give the impression of naturalistic motions without needing

to incorporate complex simulation models.

For example, coherent noise can be used to convey the small motions of a character trying to maintain balance, the controlled randomness of eye blinking, or the way a character's gaze wanders around a room. Although in real life each of these examples has a different underlying mechanism, viewers do not perceive the mechanism itself. Instead they perceive some statistics of the motion it produces. When coherent noise is applied in a way that matches those statistics, the actor's movements are believable.

The author can also import keyframed animation from commercial modeling systems, such as Alias or SoftImage. The **Improv system inter**nally converts these into actions that specify time varying values for various DOFs. To the rest of the system, these imported actions look identical to any other action.

Defining Actions

The author uses DOF's to build actions. Below are three different actions that define how an actor might gesture with his arm while talking. Each one uses several frequencies of noise to modulate arm movement. The first two are general hand waving gestures, while the third shakes the arm more emphatically, as though pointing at the listener.

On each line of an action, the part name is followed first by three angular intervals, and then by three time−varying interpolants in braces. Each interpolant is used to compute a single angle in its corresponding interval. The results are applied to the part as Pitch, Roll and Yaw rotations respectively. The angle intervals are constant over time, whereas the time varying interpolants are reevaluated at each update cycle. For example, in the first line below, if N0 possesses the value 0.5 at some time step then the resulting Pitch rotation at that time step will be 0.5 of the way between 25 degrees and 55 degrees, or 40 degrees.

```
define ACTION "Talk Gesture1"
{
R_UP_ARM      25:55   0    −35:65          { N0  0 N0 }
R_LO_ARM      55:95   0      0             { N1 0  0 }
R_HAND       −40:25  75:−25 120           { N1 N2  0 }
}

define ACTION "Talk Gesture2"
{
R_UP_ARM      10:47   0    −10:45          { N0  0 N0 }
R_LO_ARM      35:77   0      0             { N1  0  0 }
R_HAND       −53:55 −40:15  120           { N1 N2  0 }
}

define ACTION "Talk_Gesture3"
{
R_UP_ARM      45     20:15    0           {  0 N0 N0 }
R_LO_ARM      70:120  0       0           { N1  0  0 }
R_HAND        40:15   0     120           { N2  0  0 }
}
```

The variables N0, N1 and N2 are shorthand that the Improv system provides the author to denote time varying coherent noise signals of different frequencies. N1 is one octave higher than N0, and N2 is one octave higher than N1. The value of each signal varies between 0.0 and 1.0.

Note that the upper arm movement is controlled by N0, whereas the lower arm movement is controlled by N1. The result is that the upper arm will, on the average, swing back and forth about the shoulder once per second, whereas the lower arm will, on the average, swing back and forth about the elbow twice per second. Meanwhile, the hand will make small rapid rotations about the wrist. These frequencies were chosen simply because they looked natural. In our tests, frequency ratios that varied significantly from these did not look natural. Presumably this frequency ratio reflects the fact that the lower arm has about half as much mass as the total arm, and therefore tends to swing back and forth about twice as frequently.

Action Compositing

An Improv actor can be doing many things at once, and these simultaneous activities can interact in different ways. For example, an author may want an actor who is waving to momentarily scratch his head with the same hand. It would be incorrect for the waving movement to continue during the time the actor is scratching his head. The result could be strange. For example, actor might try to feebly to wave while his arm while making vague scratching motions about his cranium. Clearly in this case we want to decrease the amount of waving activity as we increase the scratching activity. Some sort of ease−in/out transition is called for.

In contrast, suppose we want an actor to scratch his head for a moment while walking downstage. It would be incorrect if the Improv system were to force the actor to stop walking every time he scratched his head. In this case, an ease−in/out transition would be inappropriate.

The difference between these two examples is that the former situation involves two actions which cannot coexist, whereas the latter situation involves two actions that can gracefully coexist. The authoring system should provide a mechanism to allow authors to make these distinctions in an easy and unambiguous way. To do this, Improv contains a simple set of rules. The approach we take is borrowed from image compositing methods. The Improv author thinks of motion as being layered, just as composited images can be layered back to front. The difference is that whereas an image maps pixels to colors, an action maps DOFs to values.

The author can place actions in different groups, and these groups are organized into a "back−to−front" order. Also the author may "select" any action. Given this structure, the two compositing rules are as follows:

(1) Actions which are in the same group compete with each other. At any moment, every action possesses some weight, or opacity. When an action is selected, its weight transitions smoothly from zero to one. Meanwhile, the weights of all other actions in the same group transition smoothly down to zero.

(2) Actions in groups which are further forward obscure those in groups which are further back.

Using this system, authors place actions which should compete with each other in the same group. Some actions, such as walking, are fairly global in that they involve many DOFs through the body. Others, such as head scratching, are fairly localized and involve relatively few DOFs. The author places more global actions in the rear−most groups. More localized actions are placed in front of these. Also, some actions are relatively persistent. Others are generally done fleetingly. Groups of very fleeting or temporary actions (like scratching or coughing) are placed still further in front.

For the author, this makes it easy to specify intuitively reasonable action relationships. For example, suppose the author specifies the following action grouping:

```
GROUP       Stances
    ACTION  Stand
    ACTION  Walk

GROUP       Gestures
    ACTION  No_waving
    ACTION  Wave_left
    ACTION  Wave_right

GROUP       Momentary
    ACTION  No_scratching
    ACTION  Scratch_head_left
```

Then let's say actions are selected in the following order:

Stand
Walk
Wave_left
Scratch_head_left
No_scratching
Wave_right

The actor will start to walk. While continuing to walk he will wave with his left hand. Then he will scratch his head with his left hand, and resume waving again. Finally he will switch over to waving with his right hand.

Because of the grouping structure, the author has easily imparted to the actor many convenient rules. For example, the actor knows to wave with either one hand or the other (not both at once), that he doesn't need to stop walking in order to wave or to scratch his head, and that after he's done scratching he can resume whatever else he was doing with that arm.

Applying Actions To The Model

At any animation frame, the run time system must assign a unique value to each DOF for the model, then move the model into place and render it. To compute these DOFs, the algorithm proceeds as follows. Within each group, a weighted sum is taken over the contribution of each action to each DOF. The values for all DOFs in every group are then composited, proceeding from back to front. The result is a single value for each DOF, which is then used to move the model into place.

There are subtleties in this algorithm, such as correctly compositing inverse kinematic DOFs over direct rotational DOFs. But these are beyond the space limitations of this paper. For a full treatment of the DOF compositing algorithm, the reader is referred to [Perlin96].

The author is given tools to easily synchronize movements of the same DOF across actions; transitions between two actions that must have different tempos are handled by a morphing approach: During the time of the transition, speed of a master clock is continuously varied from the first tempo to the second tempo, so that the phases of the two actions are always aligned. This is similar to the approach taken by [Bruderlin95] and [Witkin95].

Action Buffering

Sometimes it would be awkward for an actor to make a direct transition between two particular actions in a group. For example, let's say the actor has his hands behind his back, and then claps his hands. Because DOFs are combined linearly, the result would be that the actor passes his hands through his body!

We allow the author to avoid such situations by declaring that some action in a group can be a buffering action for another. The system implements this by building a finite state machine that forces the actor to pass through this buffering action when entering or leaving the troublesome action.

For example, the author can declare that the action hands−at−the−sides is a buffering action for hands−behind−the−back. Then when the actor transitions between hands−behind−the back and any other action, he will always first move his hands around the sides of his body.

figure 3: Otto demonstrating action buffering.

BEHAVIOR ENGINE

Motivation

Improv authors cannot create deterministic scenarios, because the user is a variable in the run−time system. The user's responses are always implicitly presenting the actor with a choice of what to do next. Because of this variability, the user's experience of an actor's personality and mood must be conveyed largely by that actor's probability of selecting one choice over another.

As a very simple example, suppose the user often goes away for awhile, keeping an actor waiting for various amounts of time. If the actor usually sits down or naps before the user returns, then the actor will appear to the user as a lazy or tired character. The user is forming an impression based on probabilities.

The influence of the author lies in carefully tuning of such probabilities. The goal of the behavior engine is to help the author to do so in the most expressive way possible.

Mechanism

The behavior engine provides several authoring tools for guiding an actor's behavioral choices. The most basic tool is a simple parallel scripting system. Generally speaking, at any given moment an actor will be executing a number of scripts in parallel. In each of these scripts the most common operation is to select one item from a list of items. These items are usually other scripts or actions for the actor (or for some other actor) to perform.

The real power of the behavior engine comes from "probability shaping" tools we provide authors for guiding an actor's choices. The more expressive the tools for shaping these probabilities, the more believable actors will be, in the hands of a talented author.

In the following sections we describe the working of the behavior engine. First we describe the basic parallel scripting structure. After that, we will describe the probability shaping tools.

Scripts For an Interactive World

If actions are the mechanism for continuous control of the movements made by an actor's body, then scripts are the mechanism for discrete control of the decisions made by the actor's mind.

The author must assume that the user will be making unexpected responses. For this reason, it is not sufficient to provide the author with a tool for scripting long linear sequences. Rather, the author must be able to create layers of choices, from more global and slowly changing plans, to more localized and rapidly changing activities, that take into account the continuously changing state of the actor's environment, and the unexpected behavior of the human participant.

In the next two sections, we first discuss how scripts are organized into layers, and then how an individual script operates.

Grouping Scripts

Like actions, scripts are organized into groups. However unlike actions, when a script within a group is selected, any other script that was running in the same group immediately stops. In any group at any given moment, exactly one script is running.

Generally, the author organizes into the same group those scripts that represent alternative modes that an actor can be in at some level of abstraction. For example, the group of activities that an actor performs during his day might be:

ACTIVITIES Resting Working Dining Conversing Performing

In general, the author first specifies those groups of scripts that control longer term goals and plans. These tend to change slowly over time, and their effects are generally not immediately felt by the user.

The last scripts are generally those that are most physical. They tend to choose actual body actions, in response to the user and to the state of higher level scripts. For example, an actor might contain the following groups of scripts, in order, within a larger set of scripts:

. . .
DAY_PLANS Waking Morning Lunch Afternoon Dinner Evening
. . .
ACTIVITIES Resting Working Dining Conversing Performing
. . .
BEHAVIOR Sleeping Eating Talking Joking Arguing Listening Dancing

We can think of the Animation Engine, with its groups of continuous actions, as an extension of this grouping structure to even lower semantic levels.

Individual Scripts

A script is organized as a sequence of clauses. At run−time, the system runs these clauses sequentially for the selected script in each group. At any update cycle, the system may run the same clause that it ran on the previous cycle, or it may move on to the next clause. The author is provided with tools to "hold" clauses in response to events or timeouts.

The two primary functions of a script clause are 1) to trigger other actions or scripts and 2) to check, create or modify the actor's properties

Triggering Actions and Scripts

The simplest thing an author can do within a script clause is trigger a specific action or script, which is useful when the author has a specific sequence of activities (s)he wants the actor to perform. In the following example, the actor walks onstage, turns to the camera, bows, and then walks offstage again.

```
define SCRIPT "Curtain Call"

  { "walk to center" }
  {  continue until { my location equals center } }
  { "turn to camera" }
  { continue until { "turn to camera" is done } }
  { "bow" }
  { continue for 3 seconds }
  { "walk offstage" }
```

In this case, phrases in quotes represent scripts or actions. Each of these scripts might, in turn, call other scripts and/or actions. The other information (continue, etc) is used by Imp**rov to control the** timing of the scene.

Layered Behavior

Through layering, an author can create complex behaviors from simpler scripts and actions. Take the following example:

```
define SCRIPT "greeting"

{
        { "enter" }
        { wait 4 seconds }
        { "turn  to  camera" }
        { wait 1 second }
        {  "wave" for 2 seconds
           "talk" for 6 seconds  }
        { wait 3 seconds }
        { "sit" }                { wait 5 seconds }
        { "bow" toward "Camera" }
        { wait 2 seconds }
        { "leave" }
}
```

In this example, the actor first activates the "enter" script (which instructs the actor to walk to center). The "enter" script and "greeting" script are now running in parallel. The "greeting" script waits four seconds before activating the "turn to camera" script. This tells the actor to turn to face the specified target, which in this case is the camera. The script then waits one second, before instructing the actor to begin the "wave" and "talk" actions. The script waits another 3 seconds before activating the "sit" action during which time the "wave" action has ended, returning to the default "No Hand Gesture" action in its group. Meanwhile, the "talk" action continues for another three seconds after the actor sits. Two seconds later the actor bows to the camera, waits another two seconds and then leaves.

Non−Deterministic (Stochastic) Behavior

In addition to commands that explicitly trigger specific actions and scripts, Improv provides a number of tools for generating the more non−deterministic behavior required for interactive non−linear applications. An author may specify that an actor choose randomly from a set of actions or scripts. as in the following example:

```
SCRIPT "Rock Paper Scissors"

   { choose from { "Rock" "Paper" "Scissors" }  }
```

Once an action or script is chosen it is executed as though it had been explicitly specified.

Alternately, the author can specify weights associated with each item in the choice. These weights are used to affect the probability of each item being chosen, as in the following example:

```
define SCRIPT "Rock Paper Scissors2"

  {  choose from { "Rock" .5  "Paper" .3  "Scissors" .1 } }
```

In this case, there is a 5/9 chance the actor executing this script will choose the "Rock" action, 3/9 that the actor will choose "Paper", and a 1/9 chance the actor will pick "Scissors". The decision is still random, but the author has specified a distinct preference for certain behaviors over others.

In order to create believable characters, the author also needs to be able to have these decisions reflect an actor's mental state as well as the state of the actor's environment. An actor's decision about what to do may depend on any number of factors, including mood, time of day, what other actors are

around and what they're doing, what the user is doing, etc.

In Improv, authors can create decision **rules which take i**nformation about an actor and his environment and use this to determine the actor's tendencies toward certain choices over others. The author specifies what information is relevant to the decision and how this information influences the weight associated with each choice. As this information changes, the actor's tendency to make certain choices over others will change as well.

Decision Rules

Properties

The information about an actor and his relationship to his environment are stored in an actor's properties. These properties may be used to describe aspects of an actor's personality, such as assertiveness, temperament or dexterity, an actor's current mood, such as happiness or alertness, or his relationship to other actors or objects, such as his sympathy toward the user or his attitude toward strained peas. These properties are specified by the author either when the actor is created, or else within a clause of a script, to reflect a change in the actor due to some action or event. The latter case is shown in the following example:

```
define SCRIPT "Eat Dinner"

   { "Eat" }
   { set my "Appetite" to 0 }
   { "Belch" }
```

In this case, the author specifies how an actor's behavior is reflected in his personality by reducing the actor's appetite after eating.

An author can also use properties to provide information about any aspect of an actor's environment, including inanimate props and scenery and even the scripts and actions an actor chooses from. An author can assign properties to actions and scripts describing the various semantic information associated with them, such as aggressiveness, formality, etc

The author can then uses these values in the construction of decision rules. Decision rules allow actors to make decisions that reflect the state of the world the author has created.

What Decision Rules Do

When a decision rul**e is invoked, a lis**t of objects is passed to it. The system then uses the decision−rule to generate a weight between zero and one for each object. This list can then be used to generate a weighted decision.

Each decision rule consists of a list of author−specified factors: pieces of information that will influence the actor's decision. Each of these factor**s is assigned a wei**ght which the author uses to control how much influence that piece of information has upon the decision. This information can simply be the value of a property of an object as in the following example:

```
   { choose from { "Steph" "Bob" "Sarah" }
      based on "who's interesting" }

define DECISION−RULE "who's interesting"

factor { his/her "Charisma" }          influence .8
factor { his/her "Intelligence" }                  influence .2
```

In this example, the decision rule will use the "Charisma" and "Intelligence" properties of the three actors to generate a weight for each actor that will used in the decision. In this case, the author has specified that the value of an actor's "Charisma" will have the greatest influence in determining that weight, with "Intelligence" having a lesser role. The influence is optional and defaults to 1.0 if unspecified. The equations for determining these weights can be found in Appendix A: *Decision Rule Equations.*

An author can also use the relationship between the actor and the various choices to influence a decision, by making "fuzzy" comparisons between their properties. For example:

```
  { choose from ("Fight" "Flee") based on "how courageous" }

define DECISION−RULE: "how courageous"
{
        factor { my "Courage"
              equals its "Courage Level" to within .5  }
}
```

Here, the author is comparing the actor's "Courage" property with the "Courage Level" property associated with the scripts "Fight" and "Flee". If the actor's "Courage" equals the script's "Courage Level" the decision rule will assign a weight of 1 to that choice. If the values aren't equal, a weight between 0 and 1 will be assigned based on the difference between them, dropping to 0 when the difference is greater than the "within" range. In this case, .5 . (The equations for this can be found in Appendix B: Fuzzy Logic Equa*tions)* *As the actor*'s "Courage" increases or decreases, so will the actor's tendency toward one option or the other.

An author may want an actor to choose from a set of options using different **factors to judge di**fferent kinds of items. A list of objects passed to the decision rule may be divided into subsets using author−defined criteria for inclusion. The weights assigned to a given subset may be scaled, reflecting a preference for an entire group of choices over another. For example:

```
   { choose from ("Steph" "Bob" "Sarah")
      based on "who's interesting2" }

define DECISION−RULE: "who's interesting2"
{
        subset "Those I'd be attracted to"                scale 1
        factor { his/her "Intelligence" equals
                 my "Confidence" to within .4 }

        subset "Those I wouldn't be attracted to"      scale .8
        factor {  his/her "Intelligence" equals
              my "Intelligence" to within .4 }
}

define SUBSET: "Those I'd be attracted to"
   { his/her "Gender" equals my "Preferred Gender" }

define SUBSET: "Those I wouldn't be attracted to"
   { his/her "Gender" doesn't equal my "Preferred Gender" }
```

Let's assume the actor is considered a heterosexual male (ie his "Gender" is "Male" and his "Preferred Gender" is "Female"). The weight assigned to "Steph" and "Sarah" will depend on how closely their intelligence matches our actor's confidence (being put off by less intelligent women and intimidated by more intelligent ones, perhaps). The factor used to judge "Bob" reflects a sympathy toward men who are his intellectual equal, unaffected by the actor's confidence. The scale values reflect a general preference for one gender over the other.

Coordination Of Multiple Actors

Ideally we would prefer to give an author the same control over groups of actors that (s)he has over individual actors. The proper model is that the author is a director who can direct the drama via pre−written behavior rules. To the

author, all of the actors constitute a coordinated "cast", which in some sense is a single actor that just happens to have multiple bodies.

For this reason, we allow actors to modify each other's properties with the same freedom with which an actor can modify his own properties. From the author's point of view, this is part of a single larger problem of authoring dramatically responsive group behavior. If one actor tells a joke, the author may want the other actors to respond, favorably or not, to the punchline. By having the joke teller cue the others actors to respond, proper timing is maintained, even if the individual actors make their own decisions about how exactly to react. In this way, an actor can give the impression of always knowing what other actors are doing and respond immediately and appropriately in ways that fulfill the author's goals.

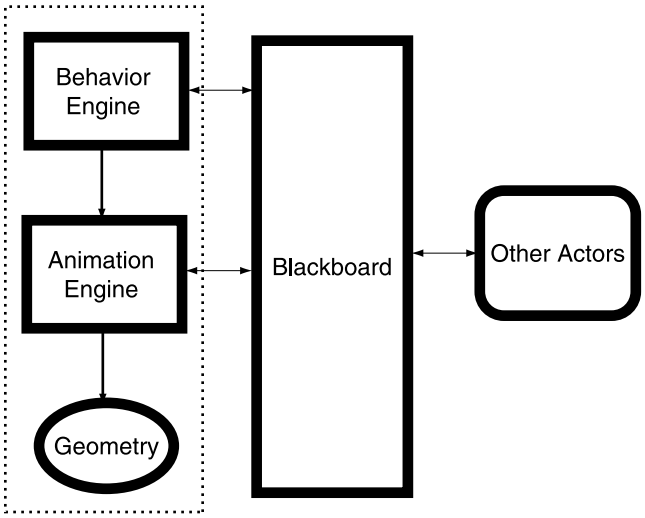

figure 4: Actors communicate with each other through a shared blackboard.

This communication occurs through the use of a shared blackboard. The blackboard allows actors to be coordinated, even when running on a single processor, on multiple processors or across a network.

USER−INTERACTION

Multi−Level Control Of Actor State

Creating and Modifying User Interface Elements

An author can also include user−interface specifications in a actor's scripts, enabling widgets to be easily generated at run−time in response to actor behavior or to serve the needs of the current scene or interaction. The user can employ these widgets to trigger actions and scripts at any level of an actor's behavioral hierarchy. This enables users to enter the virtual environment, by allowing them to direct the actions of one (or more) animated actor(s). By making this interface a scriptable element, Im**prov enables author**s to more easily choreograph the interaction between the virtual actors and the human participant.

Controlling An Actor From Multiple Levels of Abstraction

One important feature of Improv is ability for the user to interaction with the system at different semantic levels. The result of the user's actions can cause changes in the system anywhere from high level scripts to low level actions. This means that the author can give the user the right kind of control for every situation. If the user requires a very fine control over actors' motor skills, then the author can provide direct access to the action level. On the other hand, if the user is involved in a conversation, the author might let the user specify a set of gestures for the actor to use, and have the actor decide on the specific gestures from moment to moment. At an even higher level, the author may want to have the user directing large groups of actors, such as an acting company or an army, in which case (s)he might have the user give the entire group directions and leave it to the individual actors to carry out those instructions. Since any level of the actor's behavior can be made accessible to the user, the author is free to vary the level of control, as necessary, at any point in the application.

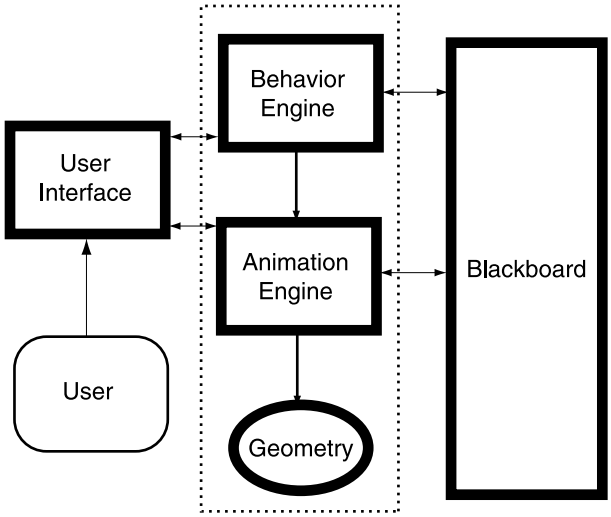

figure 5: Users interact with both the Behavior Engine and the Animation Engine through an author−defined user−interface.

IMPLEMENTATION

English−Style Scripting Language

Many of the authors and artists interested in creating interactive content are not primarily programmers, and therefore we have developed a number of "english−style" scripting language extensions to Improv that make it easier for authors and artists to begin scripting interactive scenarios. For example, all of the code examples shown in this paper were written in the current I**mprov syntax.**

Because the scripting language is written as an extension of the system language, as users become more experienced they can easily migrate from scripting entirely using the high−level english−style syntax, to extending the system through low−level algorithmic control.

Network Distribution

Improv is implemented as a set of distributed programs in UNIX, connected by TCP/IP socket connections, multicast protocols and UNIX pipes. The participating processes can be running on any UNIX machines. This transport layer is hidden from the author.

All communication between participant processes is done by continually sending and receiving programs around the network. These are immediately parsed into byte code and executed. At the top of the communication structure are routing processes. There must be at least one routing process on every participating Local Area Network. The router relays information among actors and renderer processes. For Wide Area Network communication, the router opens sockets to routers at other LAN's.

In our current implementation, each actor maintains a complete copy of the blackboard information for all actors. If an actor's behavior state changes between the beginning and end of a time step, then these changed are routed to all other actors.

Virtual Simultaneity

Typical Wide Area Network (WAN) latencies can be several seconds. This poses a problem for two virtual actors interacting in a distributed system. From the viewpoint of believability, some latency is acceptable for high level decisions but not for low level physical actions. For example, when one character waves at another, the second character can get away with pausing for a moment before responding. But two characters who are shaking hands cannot allow their respective hands to move through space independently of each other. The hands must be synchronized to at least the animation frame rate.

The blackboard model allows us to deal with this situation gracefully. We can split the Behavior Engine and Animation Engine for an actor across a Wide Area Network, and have these communicate with each other through the blackboard. For the DOFs produced by the Animation Engine, we allow the blackboard to contain different values at each LAN. For the states produced by the Behavioral Engine, the actor maintains a single global blackboard.

Computationally, each actor runs the Behavioral Engine at only a single Local Area Network (LAN), but duplicates Animation Engine calculations at each LAN. When two characters must physically coordinate with each other, then they use the local versions of their DOFs. In this way, an actor is always in a single Behavioral State everywhere on the WAN, even though at each LAN he might appear to be in a slightly different position. In a sense, the actor has one mind, but multiple bodies, each inhabiting a parallel universe. Although these bodies may differ slightly in their position within their own universe, they are all consistent with this one mind.

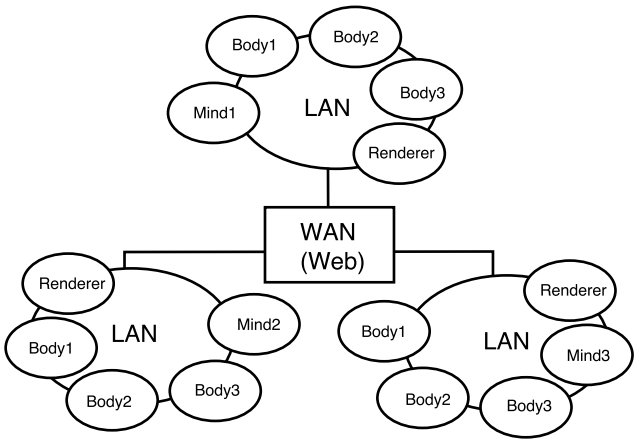

figure 6: Wide Area Network Distribution Model

This leads to an interesting and fundamental property. Let us suppose that our Improv actor Gregor is dancing while balancing a tray in an Improv scene. Further, suppose that the scene is being watched at the same time by people in Sao Paulo, Brazil, and in Manhattan, New York. Perhaps some of these people are interacting with Gregor. The connection is through the Internet.

In this scene, Gregor's Behavior Engine makes all the choices about whether to dance, whether to keep balancing the tray, how much joy and abandon versus self−conscious restraint he puts into the dance. His Animation Engine must set all the DOFs that determine how he moves when doing these things, so as to be responsive and coordinated.

If the people in NY and those in SP are talking on the telephone, they will report seeing the same thing. Yet, if a high speed dedicated video link were established, and participants could see the two Gregors side by side, they would see two somewhat different animations. In one, Gregor's hand might thrust up to balance the tray half a second sooner, in the other he might have his other arm extended a bit further out. He might be rocking right to left on one screen, while he is rocking from left to right on the other.

Thus, everywhere in the world there is only one social Gregor. He has a single mood, a single personality, he is only engaged in one task. Yet Gregor can have many slightly different physical realities, differing only up to the threshold where they might disrupt the social unity of his Behavioral State.

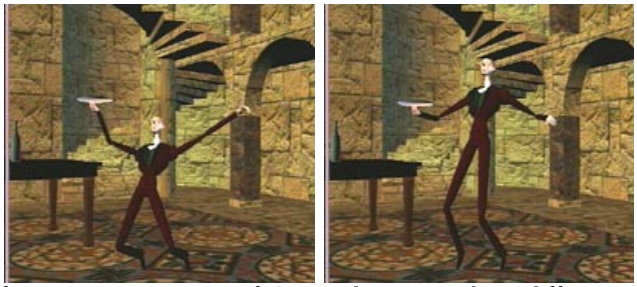

figure 7: Two versions of Gregor dancing, each on different networked computer.

In fact, if communication lag exceeds several seconds, significant differences may have occurred between the various Gregor instances. This can lead to problems. For example, suppose two actors that are temporarily out of communication each try to pick up some physical object.

This is a standard collaborative work dilemma. The only reliable solution is to make the object itself an actor (albeit a light weight one). As odd as it seems, the object itself must agree to be picked up, since it too must maintain a consistent physical reality. This was also independently observed by [Karaul95].

Communicating with Improv Actors From Outside The System

The blackboard protocol has a great advantage in terms of flexibility. To take full advantage of this flexibility, we provide a C support library that gives access to the blackboard. This allows researchers who know nothing about the Improv system, except for the names of actions and scripts, to begin immediately to control Improv actors.

For example, a researcher can write a standalone C program that links with the support library. The program can pass string arguments such as "Gregor Sit" or "Otto Walk_To_Door" to an output function. This is all that the program needs to do, in order to modify actors' behavior states.

Since the system treats the standalone program as just another actor, the program can also listen for messages by calling an input routine. These messages contain the information that updates the blackboard, saying where various actors are, what they are doing, what their moods are, etc.

In practice, this allows researchers and students at other institutions who know nothing about **Improv except its G**UI to immediately begin to use the system for their own applications. In our research collaborations we find that this is a highly successful way for our collaborators to bootstrap.

Improv also has several audio subsystems. These subsystems are used for speech generation, music generation, allowing actors to follow musical cues, and generating ambient background noise.

Extended Example

The following is an example of a scene involving multiple actors involved in a social interaction with a user.

define SCRIPT "Tell Joke"

```
{
   { do "Turn to Face" to
        choose from { others except player }
   }
   { cue { others except player } to "Listen To Joke" to me }
   {
     do "No Soap, Radio"
     do "Joke Gestures" }
   { wait until { current "Joke" is "completed" } }
   { do "Laugh" for 3 seconds }
   { cue { others except player } to "React To Joke" }
   { wait 3 seconds }
   {  do "React To Player" }
}
```

In this example, the actor executing the script randomly chooses one of the actors not being controlled by the user, and turns to him or her. The actor then cues the other non−user actors to execute the "Listen To Joke" script, in which the actor chooses the appropriate gestures and body language that will give the appearance of listening attentively.

```
define SCRIPT "Listen To Joke"
{
   {
    choose from { entire set of "Stances" } based on
       "appropriate listening gestures"
    choose from { entire set of "Gestures" } based on
       "appropriate listening gestures"
   }
   { continue for between 3 and 12 seconds }
   { repeat }
}
```

Here, the actor chooses from the actions in the of "Stances" and "Gestures" using the decision rule "appropriate listening gestures"

```
define DECISION_RULE "appropriate listening gestures"
{
   subset "Listening?"                    scale       1
   factor {
            my "confidence" is greater than
            its "confidence" to within 0.3
          }                               influence .5

   factor {
          my "self control" is less than
          its "self control" to within 0.3
            }                             influence .5
}

define SUBSET "Listening?"
    { it is "reactive" and "conversational" or  "generic" }
```

In this rule, the actor narrows the list down to those actions that are reactive and conversational, or generic actions that can be used in any context. The rule than compares the "confidence" and "self control" of the actor those assigned to each action, creating a weighted list favoring actions that match the fuzzy criteria. After choosing from the list the actor will wait from 3 to 12 seconds before repeating the script and choosing another gesture.

Meanwhile, The actor telling the joke then executes the "No Soap, Radio" script which contains a command to an external speech system to generate the text of the joke. At the same time, the actor executes the "Joke Gestures" script which, like the "Listen To Joke" script chooses appropriate gestures based on the actor's personality.

The actor continues until the joke is finished (the speech system sends a command to set the script's "completed" property to true) and then laughs, cuing the other actors to execute the "React To Joke" script.

```
define SCRIPT "React To Joke"
{
   {
    choose from { "Laugh" "Giggle" "Ignore" "Get Upset" }
        based on "feelings toward player"
   }
}

define DECISION_RULE "feelings toward player"
{
    factor { my "sympathy toward" player
                does not equal its "mood"  to within .4 }
}
```

Simply put, the more sympathy actor have for the player, the less likely they are to react positively to the joke.

Finally, the actor executes the "React To Player" script in which the actor chooses an appropriate reaction to the player, depending on whether or not the player tells his actor to laugh. if he does, the joke teller laughs, maliciously if her sympathy for the player is low, playfully if her sympathy for the player is high. If the player's actor doesn't laugh the joke teller executes the "Get It?" script, taunting the player until he gets mad and/or leaves.

figure 8: Izzy tells Otto (the user) and Elli a joke. Elli is amused, Otto isn't.

EXPERIENCE

SIGGRAPH 95

At SIGGRAPH 95 we demonstrated an interactive embodied actor named Sam who responded to spoken statements and requests. Voice recognition was provided by DialecTech, a company that has developed an interface for an IBM continuous speech recognition program. In our demonstration, untrained participants could conduct a game of "Simon Says". Sam would follow requests only if they were preceded by the words "Simon Says". To make it more interesting we programmed Sam so that sometimes he would also follow requests not preceded by "Simon Says", but then he would act embarrassed at having been fooled. Our experience was that the sense of psychological involvement by participants was very great and compelling. Participants appeared to completely "buy into" Sam's presence. We believe that this was due to several factors:

(i) participants could talk with Sam directly,
(ii) participants knew Sam was not being puppeteered (the participant was the only human in the interaction loop), and
(iii) Sam's motions were relatively lifelike and never repeated themselves precisely.

We have also found that allowing the participant to

appear as an embodied avatar enhances the participant's sense of fun and play, and therefore involvement. We had positive experience of this at SIGGRAPH 95. We presented the participant with a large rear projection of a room full of embodied conversational agents. The participant's position, as well as simple arm gestures, were tracked by an overhead video camera. The participant appeared in the scene as a flying bat. As the participant walked around, the bat flew around accordingly. The nearest agent would break out of conversing with the other agents, and begin to play with the bat. When the participant flapped his/her arms, the bat would fly higher in the scene, and the camera would follow, which gave the participant a sense of soaring high in the air. We found that participants, and children in particular, enjoyed this experience very much, and would spend long periods of time "being" the bat and flying in turn around the heads of each of the embodied agents.

figure 9: Participant interacting with Improv actors as a bat. From SIGGRAPH 95 *Interactive Entertainment* Exhibit.

Other Users

We have also provided a copy of Improv *to a number of re*searchers at other Universities. These researchers are pursuing their own research on top of our actor embodiment substrate. In at least one case, they plan to do comparisons with their own existing agent embodiment system.

Feedback from these collaborators on the use of Improv indicates that it is a useful tool for the embodiment of intelligent actors, especially for study of social interaction. In particular, it was suggested as a good tool for building educational VR environments, when used in conjunction with research software for virtual Interactive Theater. The combination can be used to simulate behaviors that would be likely to engage children to respond to, identify with and learn from knowledge agents.

We have added extensions to Improv so that animators can use commercial tools, such as Alias and SoftImage, to create small atomic animation components. Trained animators can use these tools to build up content. Such content can include various walk cycles, sitting postures, head scratching, etc. The procedural animation subsystem is designed in such a way that such action styles can be blended. For example, two or three different styles of walks can be separately designed from commercial key frame animation packages, and then blended together, or else blended with various procedural walks, to create continuously variable walk styles that reflect the actor's current mood and attitude, as well as the animator's style.

FUTURE DIRECTIONS

It is well known in traditional animation that human motions are created from combinations of temporarily overlapping gestures and stances. One of our current goals is to use Improv's ability to tie into commercial animation tools to build up a library of component motions, and to classify these motions in a way that makes them most useful as building blocks.

We have begun to embed Improv into a client−based application for a Java compatible browser (such as Netscape version 2.0). For use in educational and home settings, we plan to augment the full 3D subsystem with a "nearly 3D" version. This would run on a low end platform, such as a PC with an Intel processor. The user would still be able to see a view into a three dimensional world, but the visual representations of the actors would be simpler and largely two dimensional. For example, two participants to a graphical MUD, one with an SGI Onyx, and one with an Intel/486 based PC, could interact in the same scene. They would both see the same actors at the same locations, actions and personality. The only difference would be that the first participant would see a much more realistic quality of rendering.

We plan to integrate Improv's voice recognition and english−like behavioral sub−systems. This will allow a user to fully exploit the object substrate, giving access to direction of goals, mood changes, attitudes and relationships between actors, all via spoken English sentences.

CONCLUSION

We have described an interactive system that lets authors of various abilities create remarkably lifelike, responsively animated character interactions that run over networks in real time. We believe these techniques have the potential to have a large impact on many areas. These include: computer Role Playing Games, simulated conferences, "clip animation," graphical front ends for MUDs, synthetic performance, shared virtual worlds, interactive fiction, high−level direction for animation, digital puppetry, computer guides and companions, point to point communication interfaces, true non−linear narrative TV, and large scale deployment of bots for the Metaverse.

As Improv is a very large system, we could not cover many of its details in this paper. We refer the reader to [Perlin96] for a more in−depth treatment.

ACKNOWLEDGEMENTS

We gratefully acknowledge the support of Microsoft Corporation (and especially Dan Ling), the National Science Foundation, the New York State Science and Technology Foundation, and Silicon Graphics Incorporated (especially Keith Seto). Daniel Wey and Jon Meyer have both made important contributions to the Improv substrate. Mauricio Oka designed the flexible face model. Many other people have helped with this effort in many ways. In particular, we'd like to thank Cynthia Allen, Clilly Castiglia, Troy Downing, Steve Feiner, Laura Haralyi, Mehmet Karaul, Sabrina Liao, Marcelo Tocci More, Ruggero Ruschioni, Eduardo Toledo Santos, Jack Schwartz, Gerry Seidman, Eric Singer, Michael Wahrman, and Marcelo Zuffo. Also everyone at the, CAT and MRL at NYU, and LSI at USP. E Emi, com beijos.

APPENDICES

A. Decision Rules Equation

When an object is passed through a decision rule, a weighted sum is made of each of the values returned from the associated factors, modified by the scale assigned to the set of choices. This becomes the final weight assigned to the object that is used in making the decision.

The formula for this is as follows:

FinalWeight=Scale(factor1influence1factor2influence2...factor ninfluencen)

B. Fuzzy Logic Equations

The function compares how close the In*put Value comes to* the Target Value (or Target Range); returning a value of 1 at the Target Val*ue (or inside the Target Range), dropp*ing to 0 at a distance of Spread from the TargetValue. The fuzzy comparison is implemented as follows:

$$y = w\left(\frac{\text{InputValue - TargetValue}}{\text{Spread}} \right)$$

where:
y is the Fuzzy Value
· w is a bell curve weighting kernel (we use a raised cos function)

A high and low spread may be specified, in which case input values greater than the target value (or range) will use the high spread in the calculation, while input values lower than the target value (or range) will apply the low spread.

The returned value is then modified based on the type of fuzzy operation as follows:

equals	y Value
not equals	1−y, its complement
greater than	y, high spread defaults to infinity
not greater than	1−y, high spread defaults to infinity
less than	y, low spread defaults to −infinity
not less than	1−y, low spread defaults to −infinity

REFERENCES

N. Badler, B. Barsky, D. Zeltzer, Maki*ng Them Move: Mechanics, Control, and Animation of Articulated Figures Morga*n Kaufmann Publishers, San Mateo, CA, 1991.

N. Badler, C. Phillips, B. Webber, Sim*ulating Humans: Computer Graphics, Animation, and Control* Oxford University Press, 1993.

J. Bates, A. Loyall, W. Reilly, Integ*rating Reactivity, Goals and Emotions in a Broad Agent, P*roceedings of the 14th Annual Conference of the Cognitive Science Society, Indiana, July 1992.

B. Blumberg, T. Galyean, Multi−Level D*irection of Autonomous Creatures for Real−Time Virtual Environments Computer* Graphics (SIGGRAPH ’95 Proceedings), 30(3):47−−54, 1995.

A. Bruderlin, L. Williams, Motion Sign*al Processing, Comp*uter Graphics (SIGGRAPH ’95 Proceedings), 30(3):97−−104, 1995.

R. Brooks. A Robust *Layered Control for a Mobile Robot, I*EEE Journal of Robotics and Automation, 2(1):14−−23, 1986.

J. Chadwick, D. Haumann, R. Parent, La*yered construction for deformable animated characters. Co*mputer Graphics (SIGGRAPH ’89 Proceedings), 23(3):243−−252, 1989.

D. Ebert and et. al., Texturing and Mo*deling, A Procedural Approach Academic* Press, London, 1994.

M. Girard, A. Maciejewski, Computation*al modeling for the computer animation of legged figures.* Computer Graphics (SIGGRAPH ’85 Proceedings), 20(3):263−−270, 1985.

J. Hodgins, W. Wooten, D. Brogan, J O’Brien, Animating Hu*man Athletics, Comp*uter Graphics (SIGGRAPH ’95 Proceedings), 30(3):71−−78, 1995.

M. Johnson, WavesWo*rld: PhD Thesis, A Testbed for Three Dimensional Semi−Autonomous Animated Characters, MIT, 1994*.

M. Karaul, personal *communication*

P. Maes, T. Darrell and B. Blumberg, T*he Alive System: Full Body Interaction with Autonomous Agents in Computer An*imation’95 Conference, Switzerland, April 1995 .IEEE Press, pages 11−18.

M. Minsky, Society *of Mind, MIT press,* 1986.

C. Morawetz, T. Calvert, Goal−directed *human animation of multiple movements*. Proc. Graphics Interface}, pages 60−−67, 1990.

K. Perlin, An image *synthesizer. Compu*ter Graphics (SIGGRAPH ’85 Proceedings)}, 19(3):287−−293, 1985.

K. Perlin, Danse in*teractif. SIGGRAPH* ’94 Electronic Theatre, Orlando.

K. Perlin, Real Time *Responsive Animation with Personality, IEEE Transaction*s on Visualization and Computer Graphics, 1(1), 1995.

K. Perlin, A. Goldberg, The Improv Sys*tem Technical Repor*t NYU Department of Computer Science, 1996.
(online at http://www.mrl.nyu.edu/improv)

K. Sims, Evolving v*irtual creatures. C*omputer Graphics (SIGGRAPH ’94 Proceedings)}, 28(3):15−−22, 1994.

N. Stephenson, Snow *Crash Bantam Double*day, New York, 1992.

S. Strassman, Deskt*op Theater: Automatic Generation of Expresssive Animation, PhD thesis, MIT M*edia Lab, June 1991
(online at http://www.method.com/straz/straz−phd.pdf)

D. Terzopoulos, X. Tu, and R. Grzesczuk Artificial Fishes*: Autonomous Locomotion, Perception, Behavior, and Learning in a Simulated Physical World, Art*ificial Life, 1(4):327−351, 1994.

A. Witkin, Z. Popovic, Motion Warping Computer Graphics (SIGGRAPH ’95 Proceedings), 30(3):105−108, 1995.