Control of a Virtual Actor: The Roach

Michael McKenna, Steve Pieper and David Zeltzer

Computer Graphics and Animation Group
The Media Laboratory
Massachusetts Institute of Technology
Cambridge, MA

Abstract
We have developed a virtual environment system which supports multiple simulations, including virtual actors. These actors exhibit motor behavior in response to activity in the environment. We present an example actor, whose low-level behavior is modeled after physiological analyses of cockroach motor behavior. The *sensorimotor* activity of our roach is generated by a hierarchical control structure. Coupled oscillators generate basic gait patterns, which are modified by reflexes feeding in from the environment. Stepping and stance are executed by kinematic motor programs, which move the legs and body. The *reactive* level associates motor behavior with events in the virtual environment, to simulate perception and implement higher level behaviors. The activity of the virtual actor is determined only when it is situated in the environment, and interacts with the user and other simulations.

1. Virtual Environments, Virtual Actors and Roaches

Understanding how to represent, control and interact with autonomous virtual actors is an urgent and timely problem. For example, in the foreseeable future, robots operating in hazardous environments -- in space, in damaged nuclear power plants, on the ocean floor -- will not be intelligent enough to operate with complete independence, but will require operator intervention to perform tasks in changing or unplanned-for circumstances. Simulator systems will be needed to design and test prototype robotic agents in virtual environments. Moreover, such simulations will allow operators to train with virtual actors that model the robots that will be encountered in working situations.

This work was supported in part by the National Science Foundation Grant IRI-8712772, NHK (Japan Broadcasting Corp.), and equipment grants from Hewlett-Packard Co., Apple Computer Inc., and Gould Inc.

All kinds of animals, as well as human beings, have the capacity to move and function in the physical world efficiently, in the face of changing conditions, apparently without conscious intervention. This is the class of routine and stereotypical, perception-driven behaviors — rather than highly developed cognitive and motor skills, such as playing piano or building a chest of drawers — that our virtual actors are intended to perform.

A major objective of our work, therefore, is to learn how to represent and control the routine behavior of graphically simulated actors that coexist with other simulations in a distributed simulation environment.

In these "simulated micro-worlds" or *virtual environments* it is important that the user interface be natural and robust, and that the activity in the environment be self-regulating. Within a virtual environment, multiple simulations interact to form the global activity of the system. Some of these simulations will be *virtual actors* with repertoires of functional behaviors.

These actors should behave independently when undirected, and respond to high-level user commands. Our prototype actor? A computational cockroach which uses an adaptive gait to negotiate cluttered, level terrain. This hexapod, which we call *roach*, chooses from a set of behaviors based on the activity in the environment. For example, in its basic behavior, the roach wanders around on its own, pushing objects out of its way. Activity in the environment elicits other behaviors from the roach. This can be as simple as a direct command from the user, such as "stop walking," or more complex, such as a "grabbing hand" which triggers escape behavior. The user, other simulations, or the roach itself can modify the environment to produce different behaviors. We chose an insect behavioral model because it is fairly well described in the neurophysiological literature.

The roach, or any virtual actor, exists in a dynamic and sometimes unpredictable virtual environment governed by interacting simulations and the results of user inputs. Our virtual environment system, *bolio* [1], uses a generalized application interface to allow diverse programs to control the behavior of objects in the shared database. A constraint network in *bolio* is used to define relationships among different simulations and user input devices, creating more complex behaviors from basic skills. User input devices include the VPL *DataGlove* and Spatial Systems *SpaceBall*. The roach actor is one of many processes which cooperate to determine the overall nature of the simulated world. The roach's behavior in the world is expressed by updating the states of its body parts. Sensory input to the roach is simulated by monitoring changes in other objects in the database. By manipulating the shared objects, other processes, alone or in combination, can direct the roach's behavior. To generate realistic behavior in this world, the roach must automatically adjust to changing conditions at various levels of control.

Interaction with virtual actors should be performed at *task-level* [1; 2], meaning that the actors should respond to high level commands, stated as goals by the user. While our understanding of the representation and control of autonomous behavior is incomplete, we have nevertheless identified critical issues in the organization and control of behavior [3; 4; 5]. In our current implementation we have roughly divided the behavioral hierarchy of our prototype virtual actor into two levels: *sensori-motor*, and *reactive*. At the *sensori-motor level*, the roach uses *coupled oscillators* and *reflexes* to coordinate stepping, and *kinematic motor programs* to control movement. At the *reactive level*, constraint triggers relate virtual world events to sensori-motor level actions, such as setting the speed and direction of walking. The observable behavior of the roach in the virtual environment emerges out of the *interactions* among the roach, different simulations, and the user through the virtual environment database (see figure 1).

2. Related Work

While virtual environments have been implemented by other researchers [6; 7; 8] none of these systems provides facilities for representing and controlling virtual actors, but instead focus on issues relating to realtime rendering or telepresence.

Various graphics researchers have concentrated on particular behaviors, such as walking [9; 10; 11] and crawling[12], or group behaviors such as flocking [13]. Others have focused on particular classes of virtual actors. E.g., Badler *et al* have developed a virtual environment that is designed to portray the execution of task protocols by virtual astronauts in space station environments [14]. These systems, however, do not support distributed simulations, interactions among a variety of autonomous actors, or gestural interaction.

A fundamental problem of task level simulation is to represent the perception-driven behaviors of actors in an environment. There is thus a strong connection with work in robotics, which, as Brady has observed, "...is the field concerned with the connection of perception to action" [15]. Brooks argues for a process model of robot behavior, and has developed robotic insects capable of simple behaviors, in much the same spirit as the work we report here[16]. Agre and Chapman have developed a reactive planner, PENGI, in which a simple actor reacts to events and other, simpler actors in a 2D videogame-type environment [17].

3. Motion and Reaction

We expect the behaviors exhibited by our actors to be modeled approximations to the behaviors seen in their real-world counterparts. Therefore, our selection of the control systems employed by our virtual actors has been motivated by ethological and psychological studies of human and animal behavior. Research into the neural control of motor behavior emphasizes the hierarchical organization of functional units [18; 19; 20].

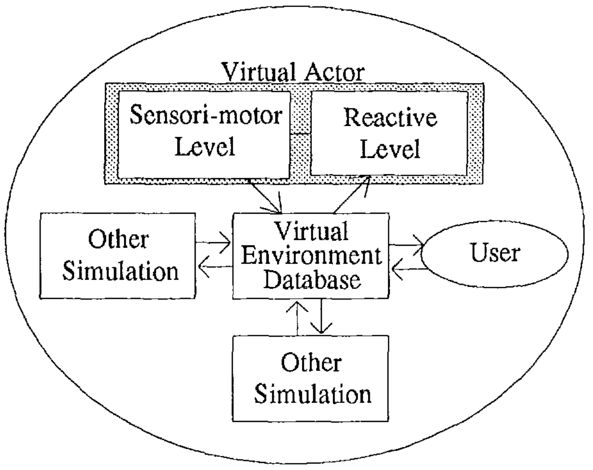


figure 1: Commands are issued by the user by manipulating the environment. Not only the user, but also simulations change the environment database. At the reactive level, the current state of the world triggers motor level responses. The sensori-motor level controls the low-level skills of the actor, and outputs its new state (body part positions, operating parameters).

Weiss proposed a hierarchy of biological motor control for producing coordinated activity[21], which we paraphrase as follows:

Level 6: The organism
Level 5: Motor organ systems
Level 4: Motor organ
Level 3: Muscle group
Level 2: Muscle
Level 1: Motor neurone

Level 1, the motor neurone, represents an individual muscle neuron and the attached muscle fiber. Taken individually, no useful work can be accomplished by units at this level. Level 2 represents a large collection of motor neurones grouped together as a *muscle*. When the motor neurones in a mass of muscle tissue are fired randomly, no organized work is accomplished, although heat is generated, and this is the behavior we call "shivering". When the neurones are activated in unison, the muscle unit can be made to contract, and this is the basis for organized motor behavior. Level 3 represents a *muscle group* which cooperates to control motion at a single joint, e.g., agonist-antagonist pairs. The *motor organ* at Level 4 represents a motor unit that controls the organized behavior of multiple muscle groups to generate useful motion of a single limb, e.g., the stepping or support motions of one leg. The level of the *motor organ system*, Level 5, controls the actions of a set of motor organs as a complex, coordinated behavior, such as walking or jumping. The highest level, according to Weiss, that of the *organism*, employs the various motor skills of the lower levels in response to external stimuli and the internal state of the organism.

In a virtual actor, the motor programs which carry out routine behaviors roughly correspond to the spinal cord and lower brain centers (of higher animals) or the ganglia (of insects). We have termed this the *sensori-motor level* of control, which corresponds to level 5 and below in the Weiss hierarchy.

In animals, the choice of motor behavior in response to sensory input is carried out in Level 6 (and higher levels) of the nervous system. We term this the *reactive* level, in which behavioral responses are triggered by environmental stimuli.

For most mobile organisms, this hierarchy governs what we might call *routine* or *instinctive* behavior. In humans and other "higher" mammals, we can speculate that Level 6, the reactive level, is augmented by enormously complex and poorly understood cognitive structures, such as those discussed by Minsky [22] and other researchers in artificial intelligence and cognitive science.

Sensory input is employed at both the sensori-motor and reactive levels of control. Sensation at the sensori-motor level mostly takes the form of peripheral and proprioceptive feedback associated with reflex arcs. For example, during walking, the step reflex increases the adaptability of a gait to unpredicted circumstances. Other stimuli are used as analog signals to low-level controllers, such as the turning servo-mechanism described later in this text.

At the reactive level, the organism responds to environmental stimuli, and composes behaviors in parallel or in sequence to satisfy some behavioral goal. For example, when the user forms the "follow" posture with the DataGlove, the roach is triggered to engage in its *follow* behavior, in which the position of the moving virtual hand serves as a target for the lower-level walking skill.

To summarize, at the sensori-motor level we define the lower level motor units -- muscle groups, motor organs and motor organ systems -- needed to implement the behavior repertoire of the actor being modeled. At the reactive level, we need a control structure such that the organism can select and sequence useful behaviors in response to internal goals as well as environmental stimuli, brought about by the user and other actors or simulations.

4. The Sensori-motor Level

The basic behavior of our roach-actor is locomotion. A hierarchical control mechanism within the sensori-motor level produces the movements of the body parts in order to walk with a specified speed and direction. The *sensori-motor level* is organized such that a gait controller produces stepping patterns which are then executed by kinematic motor programs. These programs use inverse kinematics to compute leg joint angles from the positions of the feet as they step or support.

The gait controller, a Level 5 *motor organ system*, produces the coordinated activity of stepping and stance. Each of the activities of stepping and stance involves coordinated *motor organ* activity, at level 4. The motor programs control Level 3 *muscle groups* at the joints to generate the body and limb motions that move the animal.

We are also investigating dynamic walking systems in which all motion is physically simulated [23], creating very complex and realistic motion. A frame from the computer animation "Grinning Evil Death" is shown in plate 5. The large metallic roach is dynamically simulated, but uses the same gait control mechanisms described below. Presently this system is slower than real time and precludes direct interaction.

4.1. Gait Controller

The coordinated activity of stepping and stance results in locomotion. The *gait controller* creates the pattern of stepping activity. Our controller design was motivated by the analyses of insect neurologists, such as Wilson [24] and Pearson [25], who determined that the gaits of cockroaches (as well as other animals) arise from the interaction of a small network of functional modules.

4.1.1. Coupled Oscillators

In our gait controller (as well as in the biological cockroach [25]), each leg has a pacemaker, or *oscillator*, which rhythmically triggers the leg to step. The oscillators are interconnected and coupled such that they maintain certain time and phase relationships. These relationships can be described by simple rules which were observed in the cockroach by Wilson [24]. They are summarized as:

1. A wave of steps runs from rear to head.
2. Adjacent legs across the body step 180° out of phase.
3. Stepping time is constant.
4. The frequency with which each leg steps varies.
5. The interval between steps of adjacent legs on the same side of the body is constant.

Oscillator frequency determines the generated gait, and a continuous change of frequency results in a smooth gait change. Therefore, in order to walk with a certain velocity, the higher levels of control simply set the corresponding oscillator frequency and the gait appropriate for that velocity is generated. See figure 2 for a diagram of the gait controller.

Our oscillator implementation can generate most of the gaits seen in the cockroach and many insects. At slow oscillator frequencies, a *wave gait* results, in which a wave of steps passes up each side of the body of the roach, from rear to head. As the oscillator frequency increases, the waves which are passing up each side of the body overlap more and more in time, until the fastest gait possible, the *tripod gait*, results. In the tripod gait, a stable tripod is formed by three supporting legs, while the other three legs step forward.

The oscillator model is valid for different leg counts, also. The model can coordinate an arbitrary number of leg pairs; leg count simply becomes another parameter to the controller [26].

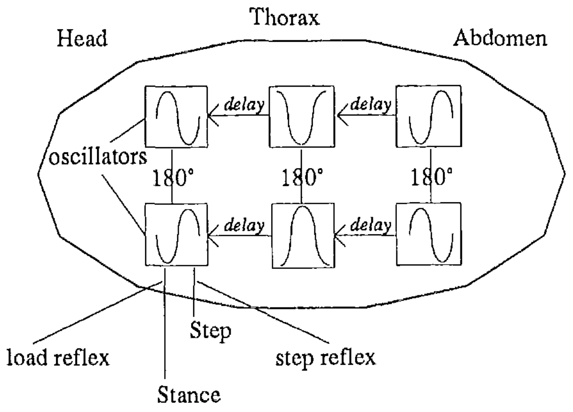


figure 2: Within the gait controller, coupled oscillators create a stepping pattern which is modified by reflexes feeding in from the environment.

4.1.2. Reflexes

Reflexes act to modify the basic stepping pattern created by the coupled oscillators. A *step reflex, load-bearing reflex* and *over-reach reflex* serve to increase the robustness of the generated gait. The first two reflexes are modeled after those found in the biological cockroach, the latter is used to accommodate for limitations of kinematic modeling. In general, the reflexes reinforce the basic pattern generated by the oscillators, but increase the adaptability of the gait. Reflexes are especially important during speed changes and turning, which can be initiated at unpredictable times from the reactive level.

The *step reflex* triggers a leg to step if its upper limb segment rotates beyond a certain angle relative to the body. In the cockroach, sensory hairs are responsible for this detection [25]. Our model simply examines the angle computed by the inverse-kinematics routine. The step reflex increases gait stability especially during turning, since legs on the outside of the turn are rotated backwards at a faster than normal rate. The reflex is also active during speed changes, which can delay the step of a leg. In fact, the step reflex alone can generate a normal gait sequence, once that gait has been initiated by the oscillator mechanism.

The *load-bearing* reflex inhibits stepping if the leg is supporting a load. In the exoskeleton of the cockroach, cuticle stress-receptors measure the leg support. Because our model is kinematic, weight measurements are not directly available. We could estimate the weight, based on the geometry of supporting legs. Instead, we simulate the effect of this mechanism by using a table of stable stepping configurations, which allows for rapid execution of the reflex. The table is an approximation of the weight measurement, or supporting polygon calculation, and

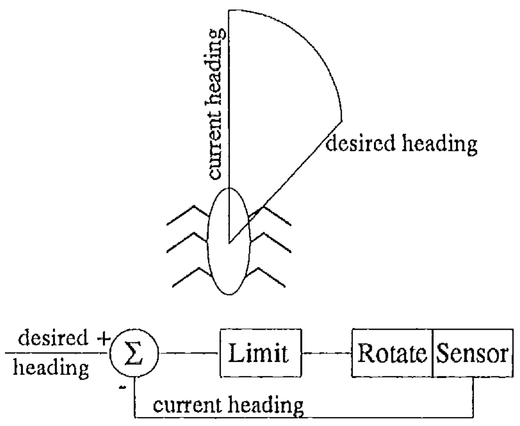


figure 3: turn-to-heading servo-mechanism. To turn to a specific heading, the roach computes the difference between its current heading and the desired heading. The difference is limited to a specified maximum turning angle (per unit time). The body is rotated by the computed amount, and the new heading feeds back into the servo-mechanism for the next time step.

basically prevents stepping of adjacent legs (across the body or adjacent on the same side). An additional stepping configuration is allowed when both middle legs step, and the four remaining legs support. Other, less stable stepping patterns can be allowed under software control. One such pattern is when the two middle legs and one front leg step together. This reflex is especially useful for adapting to missing limbs.

We have implemented a reflex not found in the biological cockroach, which we term the *over-reach reflex*. This reflex, useful in our kinematic simulation, is triggered when a leg over-reaches its kinematic extent, i.e. as a supporting foot becomes too far from the body for the leg to reach. This happens mostly when very tight turns are executed. A number of motor responses are appropriate when this reflex is triggered. We have chosen to simply back-up one time step, so that the foot is again within reach. Forward body motion is halted until the overextended leg's oscillator again triggers stepping, and forward body motion is resumed.

4.2. Motor Control

After the stepping pattern has been coordinated by the gait controller, the leg and body positions must be updated appropriately. Kinematic motor programs are used to position the end-effectors, and to update the body position. Inverse kinematics solve for the position of the legs from the computed body and foot positions. The inverse-kinematics solution for the legs is solved directly for the simple three degree of freedom legs, resulting in rapid execution time. General inverse kinematics is available through *bolio* if other leg configurations are desired.

The end-effector motor programs are responsible for stepping and stance. During stance, the motor program maintains the foot position in world space, as the body moves forward (and possibly turns). The stepping motor program raises, then lowers the foot, and brings it forward in the body local space.

The body-movement motor program translates the body forward with a velocity determined by the oscillator speed. The over-reach reflex suspends the action of the body motor program when a leg has reached its kinematic limit.

Turning towards a given heading is controlled by a servo-mechanism which rotates the body incrementally towards the correct heading (see figure 3). As the body turns, the stance motor programs keep the supporting feet in place on the ground. The resulting motions visually approximate pushing by the legs.

The turning servo-mechanism can be controlled directly from the reactive level by setting the desired heading and turning radius (which limits the amount the roach can turn each time step). Alternately, the roach can coordinate its own turning using a *path-following* controller which plans turns to follow a given path of connected points (see figure 4). The design of this controller was motivated not by insect study, but from our desire to have the actor remain on the piece-wise linear path as closely as possible.

4.3. Implementation

The gait oscillators are modeled as sine waves which trigger when they reach the peaks of their cycles. Reflexes act as conditional units. Both of these set the state of

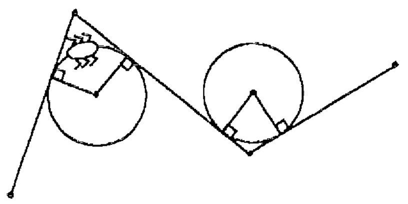


figure 4: path following. Given a set of points which form a path, the roach plans turns ahead of time in order to remain on the linear path segments as closely as possible. To plan the turn, the roach uses a specified *turning radius* which forms a circle, tangent to the two path segments. The roach begins to turn at the contact point, and uses the turn-to-heading servo-mechanism to come around to the new heading.

the kinematic motor programs, which create the actual motion. The controlling gait parameters, such as oscillator frequency, are set with scripting commands at the reactive level.

An alternate method of modeling the oscillator mechanism has been researched by Beer, Chiel and Sterling, who employ a heterogeneous neural network to simulate the stepping patterns exhibited by the cockroach [27; 28]. Their gait generator employs 37 neurons: three motor neurons, two sensory neurons, and one pacemaker neuron for each leg. Walking speed is set by one command neuron. By varying the firing frequency of that command neuron, a variety of gaits is generated by the network. As with our model, wave gaits result for slower walking and the tripod gait results for fast walking. Their implementation uses only simple two dimensional motion, however.

We can interact with the sensori-motor level of our actor using a task-level program or script which prescribes a series of motor goals to satisfy. Between program commands, the sensori-motor level generates the appropriate behavior, based on its operating state. Speed, position, direction, paths and gait parameters are specified in a simple ASCII scripting language. The positions of body parts and the state of operating parameters are output at the request of other commands.

Because the task-level programming interface makes use of standard programming protocols, the roach can easily be interfaced with other programs, even across a network.

Task-level programming was used to script the animation *Cootie Gets Scared* (see plates 1 and 2). The Cootie character was given paths and speed commands which it used to generate gaits and motions. The simulation was also used to generate MIDI soundtrack corresponding to the footfall events. In plate 2, the colored dots on the ground form a path that the Cootie follows. The body motion was fed into a dynamic simulator which created head and antennae motions in response to gravity, body motions and head-turning actuators. These dynamic effects can be seen in plate 1.

In addition to the programming interface, task-level guiding tools have been implemented to control the sensori-motor level. An X-window panel with sliders allows the user to interactively set the oscillator frequency, overall actor "metabolism," and the turning radius to steer the roach's heading. We have also implemented higher level guiding interactions such as using the DataGlove to define the target point for path following behaviors. These interactions are implemented using the constraints available at the reactive level, and are described more fully in the remaining text.

5. The Reactive Level

The reactive level of control relates the motor response of the actor to stimuli it receives from the environment. Currently, we simulate the perception of environmental stimuli as a constraint network which allows us to associate events with motor acts.

Virtual actors generate behavior in response to *messages*, which serve as stimuli in the virtual environment. Virtual actors are signaled once at the start of a frame at which point they update their component objects in the global database to reflect their behavior during the current simulation time step. Other messages to the virtual actor are sent in response to changes in the objects which are to influence the actor's behavior. These messages are used to modify the goal states for behaviors such as object tracking, or to modify the virtual actors' internal state when, for example, its own position is changed by user input.

Exchange of environmental information is implemented through *bolio*, a graphical simulation platform which maintains a global object database and implements message passing among distributed processes. This is similar in spirit to message passing systems such as those described in [29; 30].

Information about the environment is organized in terms of the *graphical objects* used to represent the scene. Non-graphical information (or *attributes*) describing the state of the virtual actor can be dynamically associated with the graphical objects through the underlying object system.

Processes can receive messages, query and modify the database of these objects. Processes may either be compiled into the bolio executable, in which case they access the database through a function call interface, or they may be external programs (possibly executing on other machines) which are connected to bolio at run time. These external processes send and receive messages and access the database through UNIX pipes and sockets. Processes in the system include physical simulations, actor behavior, and user input devices.

Constraint triggers built into the object system generate messages as a function of changes in object state. These messages carry programming commands and data to processes. The configuration of constraint triggers defines the information flow through the system, and is called the *constraint network*. Constraint triggers in the network can be combined into named *constraint sets* which can then be collectively deactivated to inhibit message passing in all the constraint triggers in the set. Constraint triggers can generate messages to activate or deactivate constraint sets. This allows construction of hierarchies where conditions at the top levels influence the way in which the lower levels respond to events.

An instance of a virtual actor includes the set of objects defining the actor, a process which implements the sensori-motor level activity, and a set of constraint triggers. These send sets of sensori-motor level programming commands to the motor skill process when specified events in the global database occur.

At the beginning of each simulation time step, *bolio* signals a start-of-frame event and puts all messages into a pending message queue. *Bolio* then enters a loop which sequentially removes the top message from the pending queue and sends it. Upon receipt of a message, a process can query and modify the object database and directly (or indirectly through constraint triggers) add messages to the end of the pending message queue. The loop ends when all messages have been sent, at which point the graphical objects are rendered. Implications of this architecture are discussed in [1; 31].

To illustrate the implementation of the reactive level, consider an instance of interaction between the roach and the user through the DataGlove. The process responsible for the DataGlove converts raw hand measurements into the position, orientation, and shape description for the graphical objects which make up the virtual hand [32; 33]. The DataGlove process also compares this shape with a set of recorded postures. If a match is detected, a "posture" attribute of the virtual hand is set to the posture name. Constraint triggers associated with that object are then checked to see if messages should be sent out for that state change. The roach "follow" behavior refers to the activity in which the roach is continually instructed to walk to the position of the virtual hand using its path-following mechanism (see figure 4). This behavior is implemented by a constraint trigger which sends new path messages to the roach for each new position of the virtual hand as long as the "posture" attribute remains in the "follow" state.

6. Interaction in the Virtual Environment

While the sensori-motor level and the reactive level define *how* the roach will execute its behaviors, it is not until the virtual actor is situated in its environment that *what* the roach will do is determined. Activity in the environment stimulates the roach, and activates behaviors. The constraint triggers at the reactive level direct the stimulation to the actor from user input devices or from other simulations. This technique has been used to implement a range of behaviors in a prototype environment which contains the roach actor and a number of balls and boxes on a level surface (see plate 3). A DataGlove, Spaceball, and mouse are available for direct manipulation of cursor objects in the environment. Keyboard and X-window interface tools are available for controlling the bolio system and for sending specific messages to processes.

One of the processes active in the environment is the *physics* process, which performs simple dynamic simulation of the motion of objects in the scene. The roach and each of the balls and boxes in the environment have mass and velocity attributes. Each frame, the physics process evaluates the scene and calculates the forces to apply to each body, divides the force by the body's mass to get acceleration, and integrates by the current time step value to get new velocities and positions for the objects. Forces which act on the objects include gravity, elastic collision with other objects, and spring-dashpot elements between objects. When the roach is active, its internal sensori-motor level calculations override the physics process in defining the position and orientation of the roach. Even when not generating the motion of the roach, the physics process still uses the position of the roach to generate repulsive forces which act on other objects in the scene. This allows the roach to push objects out of its way rather than just walk through them. The current point mass dynamics will soon be replaced by a more complete articulated body simulation package [34].

Direct manipulation of the objects in the environment is implemented through a *grabber* process. This process is triggered by the "grabbing" state of a specified cursor object (e.g. the recorded grab posture of the DataGlove or a specified button on the Spaceball). When triggered, the grabber process calculates the distance from the cursor object to the set of objects defined as grabbable for this cursor. The grabber process then sets the "nearest_object" attribute of the cursor to be the name of the nearest object in the grabbable list. If the distance to this object is less than a specified value, the grabber process sets the "grabbed" attribute of this object to be true. While grabbing an object, the grabber process is triggered by each movement of the cursor and the grabbed object is updated to follow that motion. When the cursor exits the "grabbing" state, the "grabbed" attribute is removed from the released object. Other processes may then resume controlling its position (i.e. it may fall under gravity or crawl away). In effect, the grabber process implements a 3D version of "click and drag". Depth cues, including shadows on the ground plane and rubber band lines to the nearest object, help the user track to the desired object.

Constraint sets have been used to build an escape behavior using the roach, glove, and grabber processes. The behavior includes a constraint trigger which is activated when the virtual hand's "nearest_object" attribute becomes "roach" (i.e. the roach is in danger of being grabbed). This sends messages to the roach to increase its walking speed and to set its goal point to be a specified safe hiding place. When the virtual hand "nearest_object" attribute has a value other than "roach" (i.e. the roach is safe), the walking speed is set back to normal and the roach resumes normal travel through the environment. The escape behavior is something like a priority interrupt which must be serviced by the roach before it returns to its previous activities.

Using the *bolio* constraint facilities, the camera position and orientation can be attached to the roach, providing a "roach's eye view" of the micro world. While "riding the roach," the virtual DataGlove is attached to the moving camera, and the same "follow" mechanism now allows for relative steering, much like a "carrot-on-a-stick."

Any objects in the environment can be used as the cursor object which directs the motion of the roach. In the "heel" behavior, the roach tracks the moving hand of an animated, walking human character [10] (see plate 4).

Optionally, any of the walking commands can operate with a real-time path planning module [35], which generates the shortest, unobstructed path through a static environment

7. Conclusion

We have created a simulated, walking roach, which acts independently as well as under user command in a virtual micro-world. At the *reactive level,* constraints translate interactive input to *sensori-motor level* command sets. These commands determine the speed, direction and gait parameters for locomotion. Walking is then executed by the sensori-motor level, which uses a biologically-inspired gait controller and kinematic motor programs to create motion.

We feel the design of virtual actors should be guided by research into the behavior and neural organization of real animals, a view also shared by workers in robotics [36; 37]. In this paper, we have described our translation of a neurophysiological description of cockroach behavior into an animated model for study and play. We have also presented our paradigm for interaction with virtual actors in which coordinated behavior is triggered by changes in the environment. The user is able to modify the state of the environment through input devices which monitor the same information channels the user would use to communicate with real animals.

The reactive level of behavior is the most complex and least understood of the levels of control, since it incorporates goal directed autonomous actions, up to and including intelligent behavior. The scheme we present here allows for the definition of stereotyped behavior patterns. We are currently implementing a mechanism for controlling behavior— the *expectation lattice* [4]— based on ethological principles [20].

8. Acknowledgements

The authors would like to thank Cliff Brett and David Sturman who did development work on *bolio.* Also, to Peter Schröder, who wrote the path planning module in *bolio.* Thanks to Fran Taylor and Steve Strassmann who wrote the MIDI software for the footfall synchronization in *Cootie Gets Scared.* Thanks also to David Small and Judy Sachter who modeled objects in the *Cootie* plates. The *Grinning Evil Death* sparks are courtesy of Steve Drucker. Many thanks to David Chen and Peter Schröder who wrote the rendering software for *Cootie* and *Grinning Evil Death.* Many of the objects in the *Grinning Evil Death* plate were created by Bob Sabiston, who co-directed the animation with Mike McKenna.

9. Bibliography

[1] Zeltzer, D., S. Pieper and D. Sturman. (June 1989). An Integrated Graphical Simulation Platform. *Proc. Graphics Interface 89*, London Ontario. 266-274.

[2] Zeltzer, D. (December 1985). "Towards an Integrated View of 3-D Computer Animation." *The Visual Computer.* 1(4): 249-259.

[3] Zeltzer, D. (August 1984). *Representation and Control of Three Dimensional Computer Animated Figures*, Ph.D. Thesis, Ohio State University.

[4] Zeltzer, D. (May 14-16 1987). Motor Problem Solving for Three Dimensional Computer Animation. *Proc. L'Imaginaire Numerique.* Saint-Etienne, France.

[5] Zeltzer, D. (April 1983). Knowledge-based Animation. Proc. *ACM SIGGRAPH/SIGART Workshop on Motion.* 187-192.

[6] Brooks, F. P., Jr. (October 23-24, 1986). Walkthrough -- A Dynamic Graphics System for Simulating Virtual Buildings. *Proc. 1986 ACM Workshop on Interactive 3D Graphics*. 9-21.

[7] Brooks, F.P., Jr. (May 15-19, 1988). Grasping Reality Through Illusion -- Interactive Graphics Serving Science. *Proc. CHI '88*. 1-11.

[8] Fisher, S. S., M. McGreevy, J. Amphoras and W. Robinett. (October 23-24, 1986). Virtual Environment Display System. *Proc. 1986 ACM Workshop on Interactive Graphics*. 77-87.

[9] Girard, M. (June 1987). "Interactive Design of 3D Computer-Animated Legged Animal Motion." *IEEE Computer Graphics and Applications*. 7(6): 39-51.

[10] Zeltzer, D. (November 1982). "Motor Control Techniques for Figure Animation." *IEEE Computer Graphics and Applications*. 2(9): 53-59.

[11] Sims, K. and D. Zeltzer. (August 2, 1988). A Figure Editor and Gait Controller for Task Level Animation. *SIGGRAPH 88 Course Notes, Synthetic Actors: The Impact of Robotics and Artificial Intelligence on Computer Animation*.

[12] Miller, G. (August 1988). "The Motion Dynamics of Snakes and Worms." *Computer Graphics*. 22(4): 169-178.

[13] Reynolds, C. W. (July 1987). "Flocks, Herds and Schools: A Distributed Behavioral Model." *Computer Graphics*. 21(4): 25-34.

[14] Badler, N. I. (June 1987). "Articulated Figure Positioning by Multiple constraints." *IEEE Computer Graphics and Applications*. 7(6): 28-38.

[15] Brady, M. (February 1984). *Artificial Intelligence and Robotics*, Artificial Intelligence Laboratory Memo 756, MIT.

[16] Brooks, R. A. (February 1989). *A Robot That Walks: Emergent Behaviors from a Carefully Evolved Network*, Artificial Intelligence Laboratory Memo 1091, MIT.

[17] Chapman, D. and P. E. Agre. (1987). Abstract Reasoning as Emergent from Concrete Activity. *Reasoning about Actions and Plans*. Los Altos, CA, Morgan Kaufmann Publishers, Inc.

[18] Gallistel, C. R. (1980). *The Organization of Action: A New Synthesis*. Hillsdale, New Jersey, Lawrence Erlbaum Associates.

[19] Turvey, M. T., H. L. Fitch and B. Tuller. (1982). The Problems of Degrees of Freedom and Context-Conditioned Variability. *Human Motor Behavior*. Hillsdale, New Jersey, Lawrence Erlbaum Associates.

[20] Tinbergen, N. (1969). *The Study of Instinct*. London, Oxford University Press.

[21] Weiss, P. (1941). "Self-Differentiation of the Basic Patterns of Coordination." *Comparative Psychology Monographs*. 17(1).

[22] Minsky, M. (1987). *The Society of Mind*. New York, Simon and Schuster.

[23] McKenna, M. A. (1990). *A Dynamic Model of Locomotion for Computer Animation*. Master's Thesis, Massachusetts Institute of Technology.

[24] Wilson, D. M. (1966). "Insect Walking." *Annual Review of Entomology*. 11.

[25] Pearson, K. (December 1976). "The Control of Walking." *Scientific American*. 235(6): 72-86.

[26] Zeltzer, D. (in preparation). *Kinematic Gait Control Using Coupled Oscillators*.

[27] Beer, R. D., L. S. Sterling, and H. J. Chiel. (January 1989) *Periplaneta Computatrix: The Artificial Insect Project*. Case Western Reserve University, Technical Report TR 89-102.

[28] Chiel, H. J. and R. D. Beer. (February 1988). *A lesion study of a heterogeneous artificial neural network for hexapod locomotion*, Case Western Reserve University, Technical Report TR-108.

[29] Reynolds, C. W. (July 1982). "Computer Animation with Scripts and Actors." *Computer Graphics*. 16(3): 289-296.

[30] Agha, G. A. (1986). *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press.

[31] Sturman, D., D. Zeltzer and S. Pieper. (1989). The Use of Constraints in the *bolio* System. *SIGGRAPH 89 Course Notes #29, Implementing and Interacting with Realtime Microworlds*.

[32] Sturman, D. J., D. Zeltzer and S. Pieper. (1989). Hands-On Interactions With Virtual

Environments. *Proc. of UIST '89: ACM SIGGRAPH/SIGCHI Symposium on User Interface Software and Technology.*

[33] Zimmerman, T., J. Lanier, C. Blanchard, S. Bryson and Y. Harvill. (1987). A Hand Gesture Interface Device. *Proc. Human Factors in Computing Systems and Graphics Interface.* 189-192.

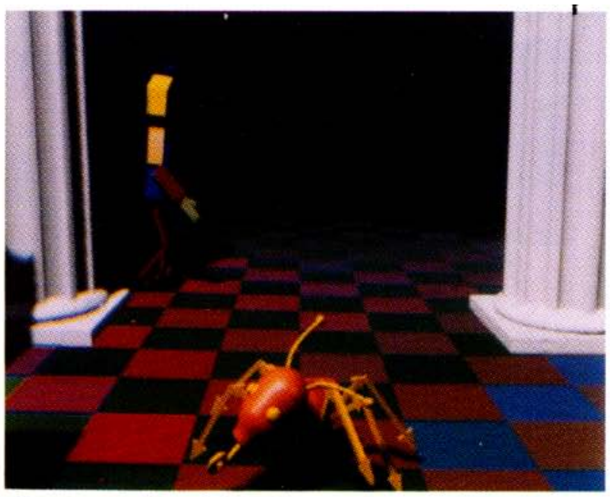
[34] Schröder, P. and D. Zeltzer. (1990). The Virtual Erector Set: Dynamic Simulation with Linear Recursive Constraint Propagation. *Proc. 1990 Symposium on Interactive 3D Graphics.*

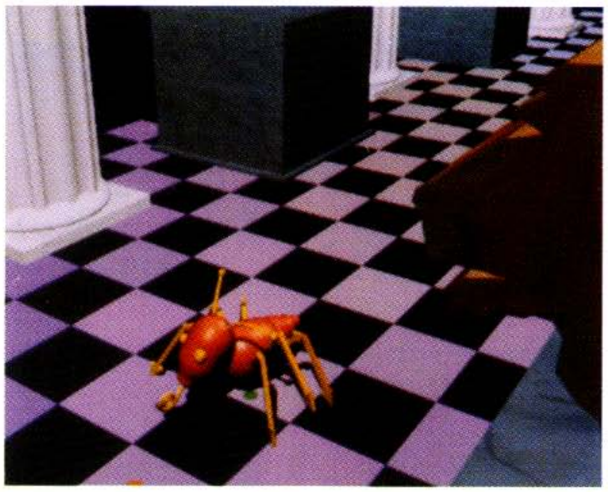
[35] Schröder, P. and D. Zeltzer. (1988). Pathplanning inside BOLIO. *SIGGRAPH 88-Synthetic Actors Course Notes.*

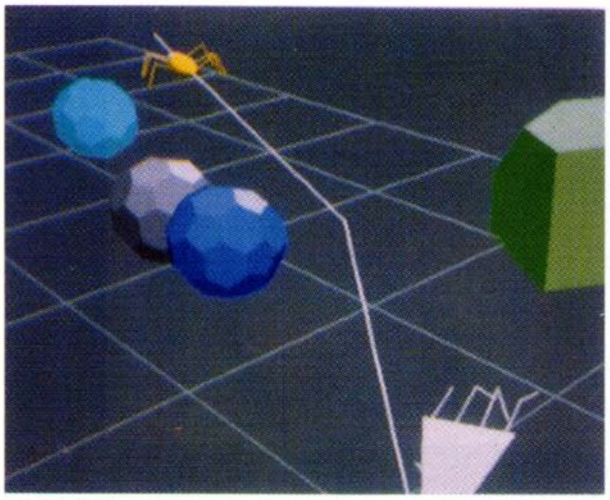
[36] Raibert, M. H. (1986). *Legged Robots That Balance.* Cambridge, MA, MIT Press.

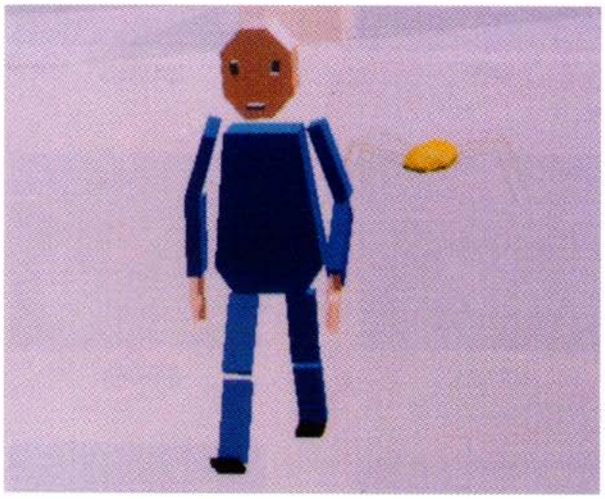
[37] Brooks, R. A. (1986). *Achieving Artificial Intelligence through Building Robots,* Artificial Intelligence Laboratory Memo 899, MIT.

Color images for this paper can be found in the color plate section.









...TASTY
MPTY
PEPSI-COLA

Plate 1: (top left) The Cootie uses walking motor skills to follow paths specified by the animator. Dynamic simulation creates bending of the antennae.

Plate 2: (top right) The Cootie's path is marked by colored dots.

Plate 3: (middle left) Path-planning software creates a collision-free path around the objects, from the roach to the virtual DataGlove.

Plate 4: (middle right) The "heel" behavior causes the roach to follow the hand of an animated biped.

Plate 5: (bottom left) Gait controlling mechanisms coordinate the stepping of this physically-simulated roach.