

# Behavioral Control for Real-Time Simulated Human Agents

John P. Granieri, Welton Becket,  
Barry D. Reich, Jonathan Crabtree, Norman I. Badler

Center for Human Modeling and Simulation  
University of Pennsylvania  
Philadelphia, Pennsylvania 19104-6389  
`granieri/becket/reich/crabtree/badler@graphics.cis.upenn.edu`

## Abstract

A system for controlling the behaviors of an interactive human-like agent, and executing them in real-time, is presented. It relies on an underlying model of continuous *behavior*, as well as a discrete scheduling mechanism for changing behavior over time. A multiprocessing framework executes the behaviors and renders the motion of the agents in real-time. Finally we discuss the current state of our implementation and some areas of future work.

## 1 Introduction

As rich and complex interactive 3D virtual environments become practical for a variety of applications, from engineering design evaluation to hazard simulation, there is a need to represent their inhabitants as purposeful, interactive, human-like agents.

It is not a great leap of the imagination to think of a product designer creating a virtual prototype of a piece of equipment, placing that equipment in a virtual workspace, then populating the workspace with virtual human operators who will perform their assigned tasks (operating or maintaining) on the equipment. The designer will need to instruct and guide the agents in the execution of their tasks, as well as evaluate their performance within his design. He may then change the design based on the agents' interactions with it.

Although this scenario is possible today, using only one or two simulated humans and scripted task animations [3], the techniques employed do not scale well to tens or hundreds of humans. Scripts also limit any ability to have the human agents react to user input as well as each other during the execution of a task simulation. We wish to build a system capable of simulating many agents, performing moderately complex tasks, and able to react to external (either from user-generated or distributed simulation) stimuli and events, which will operate in near real-time. To that end, we have put together a system which has the beginnings of these attributes,

and are in the process of investigating the limits of our approach. We describe below our architecture, which employs a variety of known and previously published techniques, combined together in a new way to achieve near real-time behavior on current workstations.

We first describe the machinery employed for behavioral control. This portion includes perceptual, control, and motor components. We then describe the multiprocessing framework built to run the behavioral system in near real-time. We conclude with some internal details of the execution environment. For illustrative purposes, our example scenario is a pedestrian agent, with the ability to locomote, walk down a sidewalk, and cross the street at an intersection while obeying stop lights and pedestrian crossing lights.

## 2 Behavioral Control

The behavioral controller, previously developed in [4] and [5], is designed to allow the operation of parallel, continuous behaviors each attempting to accomplish some function relevant to the agent and each connecting sensors to effectors. Our behavioral controller is based on both potential-field reactive control from robotics [1, 10] and behavioral simulation from graphics, such as Wilhelms and Skinner's implementation [20] of Braitenberg's *Vehicles* [7]. Our system is structured in order to allow the application of optimization learning [6], however, as one of the primary difficulties with behavioral and reactive techniques is the complexity of assigning weights or arbitration schemes to the various behaviors in order to achieve a desired observed behavior [5, 6].

Behaviors are embedded in a network of *behavioral nodes*, with fixed connectivity by links across which only floating-point messages can travel. On each simulation step the network is updated synchronously and without order dependence by using separate load and emit phases using a simulation technique adapted from [14]. Because there is no order dependence, each node in the network could be on a separate processor, so the network could be easily parallelized.

Each functional behavior is implemented as a sub-network of behavioral nodes defining a path from the geometry database of the system to calls for changes in the database. Because behaviors are implemented as networks of simpler processing units, the representation is more explicit than in behavioral controllers where entire behaviors are implemented procedurally. Where-

ever possible, values that could be used to parameterize the behavior nodes are made accessible, making the entire controller accessible to machine learning techniques which can tune components of a behavior that may be too complex for a designer to manage. The entire network comprising the various sub-behaviors acts as the controller for the agent and is referred to here as the *behavior net*.

There are three conceptual categories of behavioral nodes employed by behavioral paths in a behavior net:

**perceptual** nodes that output more abstract results of perception than what raw sensors would emit. Note that in a simulation that has access to a complete database of the simulated world, the job of the perceptual nodes will be to realistically limit perception, which is perhaps opposite to the function of perception in real robots.

**motor** nodes that communicate with some form of motor control for the simulated agent. Some motor nodes enact changes directly on the environment. More complex motor behaviors, however, such as the *walk motor node* described below, schedule a motion (a step) that is managed by a separate, asynchronous execution module.

**control** nodes which map perceptual nodes to motor nodes usually using some form of negative feedback.

This partitioning is similar to Firby’s partitioning of continuous behavior into active sensing and behavior control routines [10], except that motor control is considered separate from negative feedback control.

## 2.1 Perceptual Nodes

The perceptual nodes rely on simulated sensors to perform the perceptual part of a behavior. The sensors access the environment database, evaluate and output the distance and angle to the target or targets. A sampling of different sensors currently used in our system is described below. The sensors differ only in the types of things they are capable of detecting.

**Object:** An object sensor detects a single object. This detection is global; there are no restrictions such as visibility limitations. As a result, care must be taken when using this sensor: for example, the pedestrian may walk through walls or other objects without the proper avoidances, and apparent realism may be compromised by an attraction to an object which is not visible. It should be noted that an object sensor always senses the object’s current location, even if the object moves. Therefore, following or pursuing behaviors are possible.

**Location:** A location sensor is almost identical to an object sensor. The difference is that the location is a unchangeable point in space which need not correspond to any object.

**Proximity:** A proximity sensor detects objects of a specific type. This detection is local: the sensor can detect only objects which intersect a sector-shaped region roughly corresponding to the field-of-view of the pedestrian.

**Line:** A line sensor detects a specific line segment.

**Terrain:** A terrain sensor, described in [17], senses the navigability of the local terrain. For example, the pedestrian can distinguish undesirable terrain such as street or puddles from terrain easier or more desirable to negotiate such as sidewalk.

**Field-of-View:** A field-of-view sensor, described in [17], determines whether a human agent is visible to any of a set of agents. The sensor output is proportional to the number of agents’ fields-of-view it is in, and inversely proportional to the distances to these agents.

## 2.2 Control Nodes

Control nodes typically implement some form of negative feedback, generating outputs that will reduce perceived error in input relative to some desired value or limit. This is the center of the reactivity of the behavioral controller, and as suggested in [9], the use of negative feedback will effectively handle noise and uncertainty.

Two control nodes have been implemented as described in [4] and [5], *attract* and *avoid*. These loosely model various forms of *taxis* found in real animals [7, 11] and are analogous to proportional servos from control theory. Their output is in the form of a recommended new velocity in polar coordinates:

**Attract** An *attract* control node is linked to  $\theta$  and  $d$  values, typically derived from perceptual nodes, and has angular and distance thresholds,  $t_\theta$  and  $t_d$ . The *attract* behavior emits  $\Delta\theta$  and  $\Delta d$  values scaled by linear weights that suggest an update that would bring  $d$  and  $\theta$  closer to the threshold values. Given weights  $k_\theta$  and  $k_d$ :

$$\Delta\theta = \begin{cases} 0 & \text{if } -t_\theta \leq \theta \leq t_\theta \\ k_\theta(\theta - t_\theta) & \text{if } \theta > t_\theta \\ k_\theta(\theta + t_\theta) & \text{otherwise} \end{cases}$$

$$\Delta d = \begin{cases} 0 & \text{if } d \leq t_d \\ k_d(d - t_d) & \text{otherwise.} \end{cases}$$

**Avoid** The *avoid* node is not just the opposite of *attract*. Typically in *attract*, both  $\theta$  and  $d$  should be within the thresholds. With *avoid*, however, the intended behavior is usually to have  $d$  outside the threshold distance, using  $\theta$  only for steering away. The resulting avoid formulation has no angular threshold:

$$\Delta\theta = \begin{cases} 0 & \text{if } d > t_d \\ k_\theta(\pi - \theta) & \text{if } d \leq t_d \text{ and } \theta \geq 0 \\ k_\theta(-\pi - \theta) & \text{otherwise} \end{cases}$$

$$\Delta d = \begin{cases} 0 & \text{if } d > t_d \\ k_d(t_d - d) & \text{otherwise.} \end{cases}$$

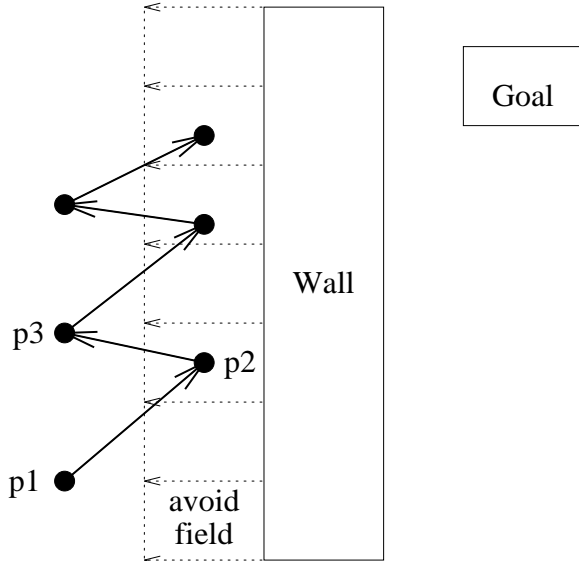


Figure 1: Sawtooth path due to potential field discontinuities

### 2.3 Motor Nodes

Motor nodes for controlling non-linked agents are implemented by interpreting the  $\Delta d$  and  $\Delta \theta$  values emitted from control behaviors as linear and angular adjustments, where the magnitude of the implied velocity vector gives some notion of the urgency of traveling in that direction. If this velocity vector is attached directly to a figure so that requested velocity is mapped directly to a change in the object's position, the resulting agent appears jet-powered and slides around with infinite damping as in Wilhelms and Skinner's environment [20].

#### 2.3.1 Walking by sampling potential fields

When controlling agents that walk, however, the motor node mapping the velocity vector implied by the outputs of the control behaviors to actual motion in the agent needs to be more sophisticated. In a walking agent the motor node of the behavior net *schedules* a step for an agent by indicating the position and orientation of the next footstep, where this decision about where to step next happens at the end of every step rather than continuously along with motion of the agent. The velocity vector resulting from the blended output of all control nodes could be used to determine the next footstep; however, doing so results in severe instability around threshold boundaries. This occurs because we allow thresholds in our sensor and control nodes and as a result the potential field space is not continuous. Taking a discrete step based on instantaneous information may step across a discontinuity in field space. Consider the situation in Fig. 1 where the agent is attracted to a goal on the opposite side of a wall and avoids the wall up to some threshold distance. If the first step is scheduled at position  $p_1$ , the agent will choose to step directly toward the goal and will end up at  $p_2$ . The agent is then well within the threshold distance for walls and will step away from the wall and end up at  $p_3$ , which is outside the threshold. This process then repeats until the wall

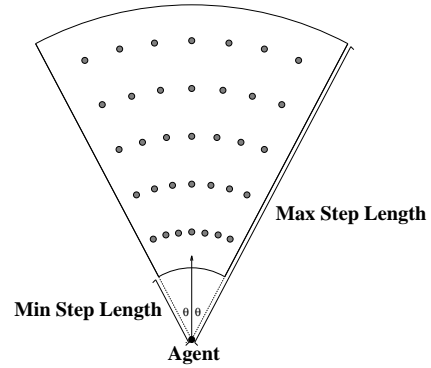


Figure 2: The fan of potential foot locations and orientations

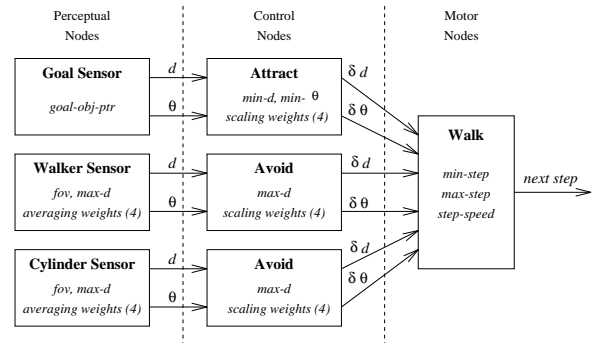


Figure 3: An example behavior net for walking

is cleared, producing an extremely unrealistic sawtooth path about the true gradient in the potential field.

To eliminate the sawtooth path effect, we sample the value of the potential field implied by the sensor and control nodes in the space in front of the agent and step on the location yielding the minimum sampled 'energy' value. We sample points that would be the agent's new location if the agent were to step on points in a number of arcs within a fan in front of the agent's forward foot. This fan, shown in Fig. 2, represents the geometrically valid foot locations for the next step position under our walking model. This sampled step space could be extended to allow side-stepping or turning around which the agent can do [3], though this is not currently accessed from the behavior system described in this paper. For each sampled step location, the potential field value is computed at the agent's new location, defined as the average location and orientation of the two feet.

### 2.4 An example behavior net

The example behavior net in Fig. 3 specifies an overall behavior for walking agents that head toward a particular goal object while avoiding obstacles (cylinders in this case) and each other. The entire graph is the *behavior net*, and each path from perception to motor output is considered a *behavior*. In this example there are three behaviors: one connecting a goal sensor to an attraction controller and then to the walk node (a goal-attraction behavior), another connecting a sensor detecting proximity of other walking agents to an avoidance controller

and then to the walk node (a walker-avoidance behavior), and a final behavior connecting a cylinder proximity sensor to an avoidance behavior and then to the walk node (a cylinder-avoidance behavior).

Each node has a number of parameters that determine its behavior. For example, the walker sensor and the cylinder sensor nodes have parameters that indicate how they will average all perceived objects within their field of view and sensing distance into a single abstract object. The Attract and Avoid nodes have scaling weights that determine how much output to generate as a function of current input and the desired target values.

The walk motor behavior manages the sampling of the potential field by running data through the perceptual and control nodes with the agent pretending to be in each of the sampled step locations. The walk node then schedules the next step by passing the step location and orientation to the execution module.

Note that this example has no feedback, cross-talk, or inhibition within the controller, though the behavioral controller specification supports these features [5]. Although this example controller itself is a feed-forward network, it operates as a closed-loop controller when attached to the agent because the walk node’s scheduling of steps affects the input to the perceptual nodes.

Our use of *attract* and *avoid* behaviors to control groups of walking agents may appear on the surface like Ridsdale’s use of *hot* and *cold* tendencies to control agents in his *Director’s Apprentice* system [18]. However, his system was not reactive and on-line as our behavioral controller is, it did not limit perception of agents, it had no structured facilities for tuning behavior parameters, and it did not take advantage of developments in reactive control and behavioral simulation. His system focused on the use of an expert system to schedule human activity conforming to stage principles and used hot and cold tendencies to manage complex human behavior and interaction. We limit the use of behaviors to reactive navigation and path-planning, using parallel transition networks rather than one large expert system to schedule events, and we look to symbolic planning systems based on results in cognitive science, such as [3, 8, 16], to automate high-level human behavior and complex human interactions.

### 3 Parallel Automata

Parallel Transition Networks (PaT-Nets) are transition networks that run in parallel with the behavior net, monitor it, and edit it over time [8]. They are a mechanism for scheduling arbitrary actions and introducing decision-making into the agent architecture. They monitor the behavior net (which may be thought of as modeling low level instinctive or reflexive behavior) and make decisions in special circumstances. For example, the agent may get caught in a dead-end or other local minimum. PaT-Nets recognize situations such as these, override the “instinctive” behavior simulation by reconfiguring connectivity and modifying weights in the behavior net, and then return to a monitoring state.

In our pedestrian example we combine object and location sensors (in perceptual nodes) with *attract* control nodes, and proximity and line sensors (in perceptual nodes) with *avoid* control nodes. Pedestrians are attracted to street corners and doors, and they avoid each other, light poles, buildings, and the street except at crosswalks.

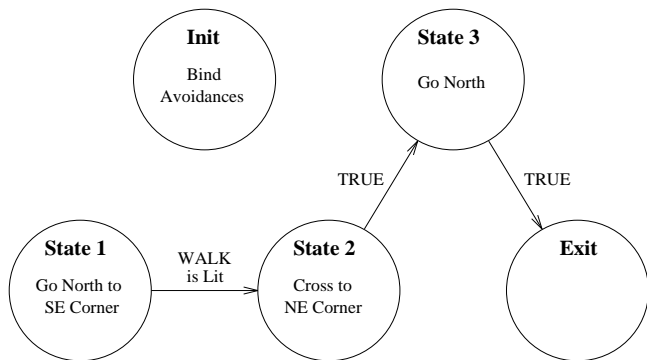


Figure 4: **North-net**: A sample **ped-net** shown graphically

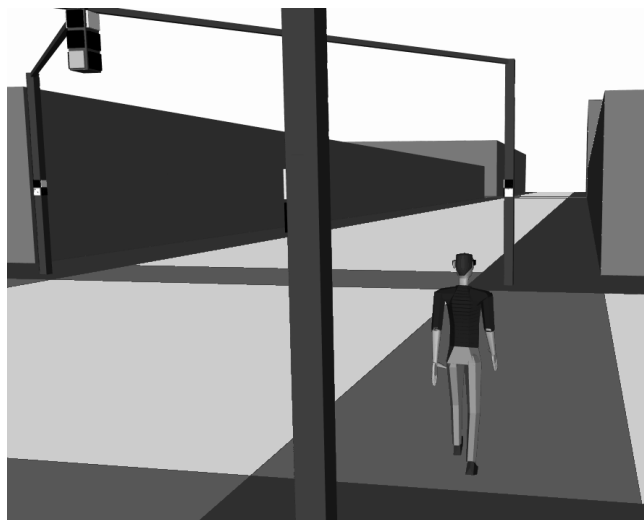


Figure 5: A pedestrian crossing the street

We use PaT-Nets in several different ways. **Light-nets** control traffic lights and **ped-nets** control pedestrians. **Light-nets** cycle through the states of the traffic light and the *walk* and *don’t walk* signs.

Fig. 4 is a simple **ped-net**, a **north-net**, which moves a pedestrian north along the eastern sidewalk through the intersection. Initially, avoidances are bound to the pedestrian so that it will not walk into walls, the street, poles, or other pedestrians. The avoidances are always active even as other behaviors are bound and unbound. In State 1 an attraction to the southeast corner of the intersection is bound to the pedestrian. The pedestrian immediately begins to walk toward the corner avoiding obstacles along the way. When it arrives the attraction is unbound, the action for State 1 is complete. Nothing further happens until the appropriate walk light is lit. When it is lit, the transition to State 2 is made and action *Cross to NE Corner* is executed. The agent crosses the street. Finally, the agent heads north.

Fig. 5 shows a pedestrian controlled by a **north-net**. The transition to State 2 was just made so the pedestrian is crossing the street at the crosswalk.

## 4 Real-Time Simulation Environment

The run-time simulation system is implemented as a group of related processes, which communicate through shared memory. The system is broken into a minimum of 5 processes, as shown in Fig. 6. The system relies on IRIS Performer [19] for the general multiprocessing framework. Synchronization of all processes, via spin locks and video clock routines, is performed in the CONTROL process. It is also the only process which performs the edits and updates to the run-time visual database. The CULL and DRAW processes form a software rendering pipeline, as described in [19]. The pipeline improves overall rendering throughput while increasing latency, although the two frame latency between CONTROL and DRAW is not significant for our application. Our CONTROL process is equivalent to the APP process in the Performer framework. We have used this framework to animate multiple real-time human figures [12].

### 4.1 CONTROL Process

The CONTROL process runs the main simulation loop for each agent. This process runs the PaT-Nets, and underlying behavior net for each agent. While each agent has only one behavior net, they may have several PaT-Nets running, which sequence the parameters and connectivity of the nodes in the behavior net over time (as shown in Fig. 6).

By far the costliest computation in the CONTROL process, for the behaviors modeled in this example application, is the evaluation of the **Walk** motor node in the behavior net, and specifically the selection of the next foot position. Since this computation is done only once for every footfall, it usually runs only every 15 frames or so (the average step time being about 1/2 second, and average frame rate 30Hz). If the CONTROL process starts running over its allotted frame time, the **Walk** nodes will start reducing the number of points sampled for the next foot position, thereby reducing computation time. The only danger here is described in Section 2.3.1, the potential for a sawtooth path. If many agents are walking at similar velocities, they can all end up computing their next-step locations at the same frame-time, creating a large computation spike which causes the whole simulation to hiccup. (It is visually manifested by the feet landing in one frame, then the swing foot suddenly appearing in mid-stride on the next frame.) We attempt to even out the computational load for the **Walk** motor node evaluation by staggering the start times for each agent, and thereby distributing the computation over about 1/2 second for all agents.

Another computational load in the CONTROL process comes from the evaluation of the conditional expressions in the Pat-Nets, which may occur on every frame of the simulation. They are currently implemented via LISP expressions, so evaluating a condition involves parse and eval steps. In practice, this is fairly fast as we pre-compile the LISP, but as the PaT-Nets increase in complexity it will be necessary to replace LISP with a higher performance language (i.e. compiled C code). This may remove some of the generality and expressive power enjoyed with LISP.

Another technique employed to improve performance, when evaluating a large number of Pat-Nets and behavior nets, is to have the CONTROL process spawn copies of itself, with each copy running the behavior of a subset of the agents. This works as long as updates to the visual database are exclusive to each CONTROL

process. (In practice this is the case, since the current behavior net for one agent will not edit any parameters for another agent in the visual database.) Of course, the assumption in spawning more processes is that there are available CPUs to run them.

The CONTROL process also provides the outputs of the motor nodes in the behavior net to the MOTION process. These outputs, in the case of the walking behavior, are the position and orientation of the agent's next foot fall. It also evaluates the motion data (joint angles) coming from the MOTION process, and performs the necessary updates to the articulation matrices of the human agent in the visual database.

### 4.2 SENSE Process

The SENSE process controls and evaluates the simulated sensors modeled in the perceptual nodes of the behavior net. It provides the outputs of the perceptual nodes to the CONTROL process, which uses them for the inputs to the control nodes of the behavior net. The main computational mechanism the sensors employ are intersections of simple geometric shapes (a set of points, lines, frustums or cones) with the visual database, as well as distance computations. This process corresponds to an ISECT process in the Performer framework.

The major performance parameters of this process are the total number of sensors as well as the complexity and organization of the visual database. Since it needs read-only access to the visual database, several SENSE processes may be spawned to balance the load between the number of sensors being computed, and the time needed to evaluate them. (These extra processes are represented by the dotted SENSE process in Fig. 6.) There is a one frame latency between the outputs of the perceptual nodes and the inputs to the control nodes in the behavior net (which are run in the CONTROL process), but this is not a significant problem for our application.

### 4.3 MOTION Process

Once the agent has sensed its environment and decided on an appropriate action to take, its motion is rendered via real-time motion generators, using a motion system that mixes pre-recorded playback and fast motion generation techniques.

We use an off-line motion authoring tool [2, 13] to create and record motions for our human figures. The off-line system organizes motion sequences into *posture graphs* (directed, cyclic graphs). Real-time motion playback is simply a traversal of the graph in time. This makes the run-time motion generation free from frame-rate variations. The off-line system also records motions for several levels-of-detail (LOD) models of the human figure. (Both the bounding geometry of the figure, as well as the articulation hierarchy (joints) are represented at several levels of detail.) The three levels-of-detail we are using for the human figure are:

1. A 73 joint, 130 DOF, 2000 polygon model, which has articulated fingers and flexible torso, for use in close-up rendering, and fine motor tasks (*Jack*<sup>®</sup>),
2. A 17 joint, 50 DOF, 500 polygon model, used for the bulk of rendering; it has no fingers, and the flexible torso has been replaced by two joints,

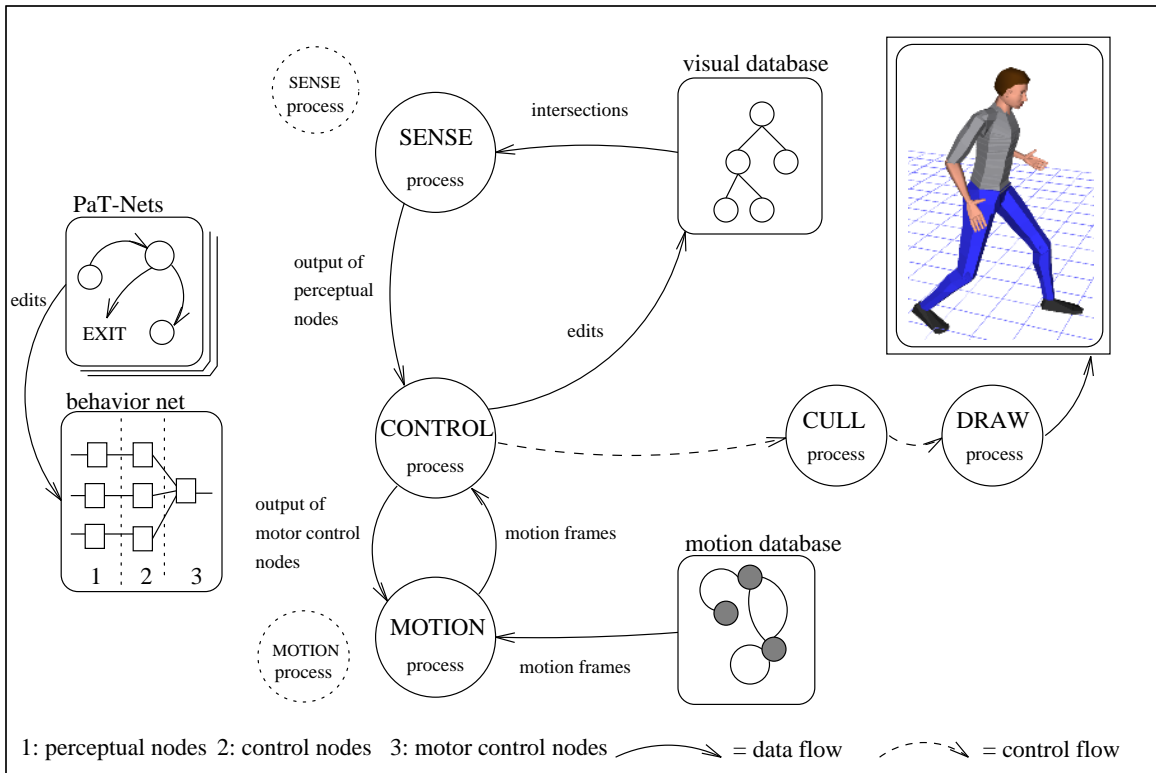


Figure 6: The multiprocessing framework for the real-time behavior execution environment

3. An 11 joint, 21 DOF, 120 polygon model used when the human agent is at a large distance from the camera.

This process produces a frame of motion for each agent, then sleeps until the next frame boundary (the earliest any new motion could be needed). It provides the correct motion frame for the currently active LOD model in the visual database. For certain types of sensors modeled in the perceptual nodes, this process will also be requested to provide a full (highest LOD) update to the visual database, in the case where a lower LOD is currently being used, but a sensor needs to interact with the highest LOD model.

The motion database consists of one copy of the posture graphs and associated motion between nodes of the posture graph. Each transition is stored at a rate of 60HZ, on each LOD model of the human agent. This database is shared by all agents. Only a small amount of private state information is maintained for each agent.

The MOTION process can effectively handle about 10-12 agents at update rates of 30Hz (on a 100MHz MIPS R4000 processor). Since the process only has read-only access to the motion database, we can spawn more MOTION processes if needed for more agents.

#### 4.4 Walking as an example

A MOTION process animates the behaviors specified by an agent's motor nodes by playing back what are essentially pre-recorded chunks of motion. As a time-space tradeoff, this technique provides faster and less variable run-time execution at the cost of additional storage requirements and reduced generality. The interesting issues arise in how we choose a mapping from

motor node outputs to this discrete representation; it plays a significant role in determining how realistic the animated agents will be.

The primary motor behavior to be executed is walking. Our full walking algorithm combines kinematics with dynamic balance control and is capable of generating arbitrary curved-path locomotion [15]. In order to reduce computational costs, however, we have not incorporated the algorithm directly into our run-time system. Instead, as implied by the preceding discussion, we record canonical "left" and "right" steps generated by the algorithm (which is a component of our off-line motion authoring system) and then play them back in an alternating fashion to produce a continuous walking motion.

The input to the appropriate MOTION process's walking subsystem consists of the specification of the desired next foot position and orientation (for the swing foot). This input is itself already discretized, as the motor node responsible (the **Walk** motor node) for evaluating how desirable it is for the agent to be at particular positions only computes the desirability criteria at a set number of points (in Fig. 2). However, even given that there are only  $n$  possibilities for the placement of the swing foot on the next step, this would still require us to record order  $n^2$  possible steps, since the planted foot could be in any one of the  $n$  different positions at the start of the step (determined by the **last** step taken) and any one of the  $n$  at the end.

Without recording all  $n^2$  distinct steps it is necessary to choose the best match among those that we do record. One of the most important criteria in obtaining realistic results is to minimize foot slippage relative to

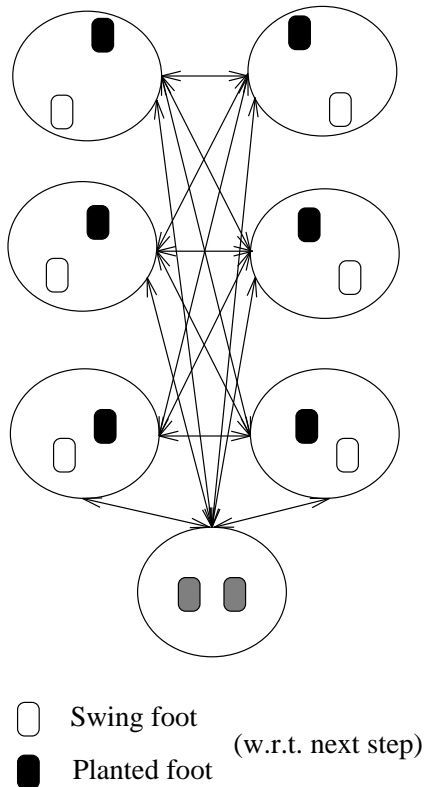


Figure 7: Posture graph for variable step length walking (3 step sizes)

the ground; foot slippage occurs when the pre-recorded movement (in particular its amount and direction) does not match that specified by the **walk** motor node at run time. On the basis that translational foot slippage is far more evident than rotational slippage (at least from our informal observations), we currently adopt an approach in which we record three types of step: short, medium, and long. Turning is accomplished by rotating the agent around his planted foot smoothly throughout the step. Having three step sizes significantly increases the chances of being able to find a close match to the desired step size, and, in fact, the **walk** motor node can be constrained to **only** consider the three arcs of the next foot location fan (see Fig. 2) that correspond exactly to our recorded step sizes. Doing so eliminates translational slippage, but has the sawtooth hazard.

The posture graph for all possible step-to-step transitions is shown in Fig. 4.4. Notice that even with only three kinds of straight-line walking there are many possible transitions, and hence numerous motion segments to be recorded. However, allowing for variable step length is very important. For instance, an attract control node can be set to drive the agent to move within a certain distance of a goal location; were there only a single step size, the agent might be unable to get sufficiently close to the goal without overshooting it each time, resulting in degenerate behavior (and possible virtual injury).

One thing worthy of mention with respect to the number of different walking steps required to reproduce arbitrary curved-path locomotion is that while there are theoretically order  $n^2$  of them, the similarities are sig-

nificant. It is thus possible that it will prove feasible to store a single full set of steps along with a little more information to represent how those steps can be modified slightly to realistically turn the agent left or right, and make it sufficiently fast for our real-time applications.

## 5 Conclusions and Future Work

We have designed a multiprocessing system for the real-time execution of behaviors and motions for simulated human-like agents. We have used only toy examples to date, and are eager to push the limits of the system to model more complex environments and interactions amongst the agents.

Although our agents currently have limited abilities (locomotion and simple posture changes), we will be developing the skills for interactive agents to perform maintenance tasks, handle a variety of tools, negotiate terrain, and perform tasks in cramped spaces. Our goal is a system which does not provide for all possible behaviors of a human agent, but allows for new behaviors and control techniques to be added and blended with the behaviors and skills the agent already possesses.

We have used a coarse grain parallelism to achieve interactive frame rates. The behavior net lends itself to finer grain parallelism, as one could achieve using a threaded approach. Our system now is manually tuned and balanced (between the number of agents, the number of sensors per agent, and the complexity of the visual database). A fruitful area of research is in the automatic load balancing of the MOTION and SENSE processes, spawning and killing copies of these processes, and doling out agents and sensors, as agents come and go in the virtual environment. Results in real-time system scheduling and approximation algorithms will be applicable here.

## 6 Acknowledgments

This research is partially supported by ARO DAAL03-89-C-0031 including U.S. Army Research Laboratory; Naval Training Systems Center N61339-93-M-0843; Sandia Labs AG-6076; ARPA AASERT DAAH04-94-G-0362; DMSO DAAH04-94-G-0402; ARPA DAMD17-94-J-4486; U.S. Air Force DEPTH through Hughes Missile Systems F33615-91-C-0001; DMSO through the University of Iowa; and NSF CISE CDA88-22719.

## References

- [1] Ronald C. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 105–122. MIT Press, 1990.
- [2] Norman I. Badler, Rama Bindiganavale, John Granieri, Susanna Wei, and Xinmin Zhao. Posture interpolation with collision avoidance. In *Proceedings of Computer Animation '94*, Geneva, Switzerland, May 1994. IEEE Computer Society Press.
- [3] Norman I. Badler, Cary B. Phillips, and Bonnie L. Webber. *Simulating Humans: Computer Graphics, Animation, and Control*. Oxford University Press, June 1993.
- [4] Welton Becket. *Simulating Humans: Computer Graphics, Animation, and Control*, chapter Controlling forward simulation with societies of behaviors.

- [5] Welton Becket and Norman I. Badler. Integrated behavioral agent architecture. In *The Third Conference on Computer Generated Forces and Behavior Representation*, Orlando, Florida, March 1993.
- [6] Welton M. Becket. *Optimization and Policy Learning for Behavioral Control of Simulated Autonomous Agents*. PhD thesis, University of Pennsylvania, 1995. In preparation.
- [7] Valentino Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. The MIT Press, 1984.
- [8] J. Cassell, C. Pelachaud, N. Badler, M. Steedman, B. Achorn, W. Becket, B. Douville, S. Prevost, and M. Stone. Animated conversation: rule-based generation of facial expression, gesture and spoken intonation for multiple conversational agents. In *Proceedings of SIGGRAPH '94. In Computer Graphics*, pages 413–420, 1994.
- [9] Thomas L. Dean and Michael P. Wellman. *Planning and Control*. Morgan Kaufmann Publishers, Inc., 1991.
- [10] R. James Firby. Building symbolic primitives with continuous control routines. In *Artificial Intelligence Planning Systems*, 1992.
- [11] C. R. Gallistel. *The Organization of Action: A New Synthesis*. Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, 1980. Distributed by the Halsted Press division of John Wiley & Sons.
- [12] John P. Granieri and Norman I. Badler. In Ray Earnshaw, John Vince, and Huw Jones, editors, *Applications of Virtual Reality*, chapter Simulating Humans in VR. Academic Press, 1995. To appear.
- [13] John P. Granieri, Johnathan Crabtree, and Norman I. Badler. Off-line production and real-time playback of human figure motion for 3d virtual environments. In *IEEE Virtual Reality Annual International Symposium*, Research Triangle Park, NC, March 1995. To appear.
- [14] David R. Haumann and Richard E. Parent. The behavioral test-bed: obtaining complex behavior from simple rules. *The Visual Computer*, 4:332–337, 1988.
- [15] Hyeongseok Ko. *Kinematic and Dynamic Techniques for Analyzing, Predicting, and Animating Human Locomotion*. PhD thesis, University of Pennsylvania, 1994.
- [16] Micheal B. Moore, Christopher W. Geib, and Barry D. Reich. Planning and terrain reasoning. In *Working Notes - 1995 AAAI Spring Symposium on Integrated Planning Applications.*, 1995. to appear.
- [17] Barry D. Reich, Hyeongseok Ko, Welton Becket, and Norman I. Badler. Terrain reasoning for human locomotion. In *Proceedings of Computer Animation '94*, Geneva, Switzerland, May 1994. IEEE Computer Society Press.
- [18] Gary Ridsdale. *The Director's Apprentice: Animating Figures in a Constrained Environment*. PhD thesis, Simon Fraser University, School of Computing Science, 1987.
- [19] John Rohlf and James Helman. IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. *Computer Graphics*, pages 381–394, 1994.
- [20] Jane Wilhelms and Robert Skinner. A 'notion' for interactive behavioral animation control. *IEEE Computer Graphics and Applications*, 10(3):14–22, May 1990.