An Interactive Approach to Behavioral Control

Jane Wilhelms
Robert Skinner

Computer and Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064

ABSTRACT

A method of interactively specifying behaviorally-controlled animation is presented. The method builds upon the behavioral animation approach described by Reynolds[14] using a connectionist approach inspired by Braitenberg.[3] Objects are given sensors and effectors, and the user interactively provides a mapping between them. The force from the motor effectors produces motion using a simple form of dynamic simulation. In this way, the user has general control over object behavior but the specific motion is automatically generated according to the state of the objects and their environment.

Key Words and Phrases: computer animation, behavioral animation, stimulus/response, interactive motion control.

1. Introduction

Recently a number of interesting advances have been made involving control and semi-automatic generation of motion for computer animation. This includes inverse kinematics[1,6,8,11] automatic constraint satisfaction[2,16,19,20] dynamics,[4,10,17] and collision detection and response.[9,13] These techniques can be considered rather *low-level*, in that control is applied in terms of linear and angular motion of objects and by defining forces, torques, or constraints applied to these objects. Less has been published in the area of *higher-level, general* motion control. Two prominent examples are the work of Zeltzer[21] in treating human walking as a multi-tier, parameterized hierarchy of motion commands, and the work of Reynolds on behavioral control.[14] This paper can be thought of as a way of interactively implementing behavioral control.

Reynold's behavioral model involved assigning objects procedurally defined behaviors which could be used to automatically generate their motion, and was used successfully to generate the motion of bird flocks and fish schools. Bird's (or *boids*) motion was determined by their desire to avoid collisions, to imitate the motion of neighboring birds, and to stay near the center of the flock. Their senses detected how much these desires were being satisfied, and this information was weighted, according to the importance of these desires, to produce new motion.

It appeared that an important problem in pursuing this promising and elegant idea was the means of quickly specifying and experimenting with different behaviors. At this point, another area entirely, i.e., neurobiology, proved inspiring. Most stimulating was the book *Vehicles* by Valentino Braitenberg,[3] which advances the idea that relatively simple mappings between sensors and effectors can produce motion that, while difficult to predict exactly, effectively produces an impression of emotions and thoughtful behavior on the part of the objects.

This paper deals with a method for interactively creating automatic complex motion using behavioral control. The method establishes connections between *sensors* and *effectors* associated with animated objects. Sensors are procedurally defined; obvious examples are light and recognition of the proximity of other objects. Effectors are most commonly motors producing motion but could also alter object color or other qualities. Using an interactive interface, sensors and effectors can be connected together with *nodes* to form a network. Connections pass signals of variable strength from sensors to effectors; nodes control the mapping of signals. Nodes may be simple *transfer functions* that invert, emphasize, or apply thresholds to the signal, or may involve more sophisticated procedural operations which alter that mapping in more complex ways, sometimes using previous experience or stochastic methods.

Initial results suggest that this method can be used to produce interesting and automatic motion whose general characteristics are under the control of the user. The approach may also be of interest beyond the field of computer animation. The connection networks might be of interest in exploring neural nets for machine learning. While strictly speaking, the connections are not neural nets at present, they could be extended in that direction to explore machine learning. Biologists might also find the system of use in simulating animal behavior.

2. Animated Objects

Animated objects for behavioral animation must include certain characteristics beyond the geometric and visual properties (such as position, orientation, topology, geometry, and color) necessary for ordinary kinematic animation. The objects need *sensors* that detect the stimuli of their environ-

ment. The objects also need *effectors*, e.g., motors, to propel them about. There must be *connections* established between sensors and effectors to produce a response. The nature of the response is also determined by *nodes* that alter the signals passing along the connections. Objects can be given *qualities* that allow other objects to recognize and react to them individually.

2.1. The Sensors

Sensors detect specific characteristics of objects in the environment. They each have a location and orientation on the animated object which affects how strongly they are stimulated. They also have a number of parameters which control how the stimulus is interpreted and passed on through the output connections. These parameters include a maximum and minimum linear and a maximum angular range within which they are sensitive. They also have a maximum and minimum scale which specifies whether and how much their sensitivity to the stimulus is affected by distance. Finally, they have a mode, which makes them sensitive either to the nearest example of the exciting stimulus, or to an average of the stimuli over their whole range.

Distance sensors (and *proximity* sensors, which are their inverse) detect the distance to other objects. They are particularly useful for collision avoidance maneuvers. *Quality* sensors detect characteristic qualities that are properties of other objects. Examples are the color of objects. Quality sensors have an added *filter* parameter that affects how sensitive they are to a particular quality.

Sensors pass a scalar output indicating how strongly they are stimulated and a vector output indicating the direction of the stimulus. The latter was added for simplicity in localizing stimuli. If a object in two dimensions is given three sensors, and an object in three dimensions given four, and the sensors are radially and symmetrically oriented, their scalar output alone can be used to find the unique stimulus direction. [7] Our objects typically had only two sensors in two dimensions and three in three, which makes it impossible to determine a unique stimulus location from scalar sensor values alone.

2.2. The Effectors

Effectors can produce some change in the object, in our case, a force that can cause motion. Potentially, this could also be a change of color, shape, etc. An effector acting as a force pushes on the object (or pulls on it if the value is negative). The effector's position and orientation on the body defines the point of application of the force and the force direction. The current value of the effector represents the magnitude of the force being applied, which is the sum of the scalar values input to the effector scaled by a parameter. Effectors also have a parameter that limits the total force it can apply.

In the real world, a force applied to a rigid body, such as our simulated objects, produces a linear acceleration; it may also produce an angular acceleration if applied away from the body's center of mass, because it also acts as a torque. Physically correct dynamics,[18] in which the acceleration is proportional to force and objects will continue to move at constant speed once started unless slowed by friction or other forces, is not used here.

Rather, the motion dynamics assume that the force is directly proportional to velocity, which inherently produces friction and causes objects to stop when not acted upon by other forces. (This does retain the ability to produce angular motion due to propulsive forces acting upon different parts of the body, which would be hard to imitate merely using kinematics.)

The equations used to generate motion are

$$\frac{\mathbf{F}}{m} = \mathbf{v}$$

$$\mathbf{J}^{-1} \cdot \tau = \omega$$

$$\tau = \mathbf{p} \times \mathbf{F}$$

where

$\mathbf{F}$ =	3D force vector in the local frame
τ =	3D torque vector ""
$\mathbf{p}$ =	3D point of application of force ""
m =	mass
$\mathbf{J}$ =	3x3 inertial tensor matrix relative to the local frame (moments of inertia on diagonal and products of inertia on off-diagonals)

More information on simple ways to calculate these quantities can be found in Wilhelms[18] or in any introductory physics or mechanics text.

2.3. Connections and Nodes

A connection can be thought of as a wire taking the value from the output of a sensor or a node and sending it to the input of one or more nodes or effectors. The connections implemented are slightly more complex, in that they can scale the signal and add to it as it passes. Sensors can have zero or more outputs but have no input connections. Effectors can have zero or more inputs but have no output connections.

Inclusion of nodes between sensors and effectors allows more complex mappings. Nodes can have both multiple input connections and multiple output connections. Each node type is associated with a procedure that calculates the output signal based upon its input. Parameters for nodes allow the user to more carefully control how the mapping takes place.

The simplest, standard node type is a *transfer function*, which takes the sum of the input values and produces a single output value based upon a user-defined function. The user defines the function as points on a cartesian coordinate system (see Figure 1). Transfer functions can be monotonic (steadily increasing) or non-monotonic functions, involving complex changes in tangent and thresholds. Transfer functions that produce a constant value independent of input, when attached to an effector, will keep the motor running continuously. Transfer functions mostly closely resemble the low-level nodes described by Braitenberg,[3] but are not very appropriate for controlling animation. For example, consider a transfer function node to cause objects to avoid one another based upon their distance. The sensor nearest the object will send a stronger signal to the effector on that side of the body than the

farther sensor will send to the opposite effector. This will cause the objects to veer away, but how quickly they turn will depend upon the differential push on the motors which will vary with distance. Furthermore, forward speed will also be affected by distance in ways that may not conform to the users wishes. For these reasons, more sophisticated nodes were developed.

Two very useful nodes are *love* and *hate*, which cause movement toward or away from incoming stimuli. For these nodes, inputs and outputs are ordered and one-to-one, so that each input (from a particular sensor) will pass to an associated effector. For these nodes, a different value can be sent to each output connection, depending upon the input from all known sensors. This gives the user much greater control over the response.

Parameters for these nodes include: *range* of stimuli to which they are sensitive; *forward scale factor* to control how much to push on all motors to produce forward motion; *revolute scale factor* to control how much more to push on one motor to cause turning; and *angular threshold* which specifies how many degrees the objects orientation diverges from the goal direction. If the threshold is set to zero, even the slightest divergence causes a corrective turn and motion may appear jittery. On the other hand, if the threshold is set high, the object will follow a zig-zag path toward (or away from) the goal.

Another very useful node is *avoid*, which acts much like hate nodes but passes output values that go up exponentially as objects approach. The sensory input for avoid nodes is assumed to be distance.

A final important node is *arbitrate*. It was soon found that the importance of signals must be weighted for reasonable motion. For example, if an object is strongly attracted to a goal some distance ahead, but also trying to avoid an obstacle in the way, the signal to go forward to the goal should be turned off to prevent a collision from occurring. Parameters for the arbitrate node indicate how signals should be passed through. Typically, the presence of an avoid signal takes precedence over all other signals.

Miscellaneous other nodes have been implemented. Some are simple, such as *and*, *or*, and *not* nodes. Network values are real, so these nodes respond as if a zero input is false, and any non-zero input is true. A *random* node is available that occasionally produces a positive signal and can make motion less smooth as well as break up cycles which sometimes happen when stable patterns are found. A *history* node alters its mapping according to the kind of signals it has already received. If it scales the signal up, it mimics a kind of habit formation, where the node reacts quicker to familiar signals; if it scales down, it mimics accommodation, where the node becomes less sensitive to its input.

2.4. Collision Response

Avoid nodes attempt to avoid a collision with the nearest object within range. When many objects are close, this strategy may fail and collisions can occur. A rather simplistic form of collision response has been implemented. When it is noted that forward motion will cause interpenetration, the object's velocity in the direction of the collision is removed (for inelastic collisions) or scaled and reversed (for elastic collisions). More realistic methods for collision response have been discussed elsewhere.[13]

3. Establishing Behaviors: the Notion Software

Notion is an interactive, multiple-window, behavioral animation system that runs on Silicon Graphics IRISes. *Notion* provides three permanent windows, as well as several temporary ones. The three permanent windows are: 1) the *graphics* window, which shows the objects in the scene (see Figure 2); 2) the *animation control* window, which accepts user input concerning the animation state, e.g., starting and stopping, running under behavioral or keyframe control, size of time steps and present time, and display characteristics (see Figure 3); and 3) the *active object* window, which shows characteristics and values of the present active object picked by the user; some of the active object fields can also be interactively edited by the user (see Figure 4). Behaviorally generated animation can be stored as keyframes for replay. Motion may be three-dimensional, or clamped to a two-dimensional plane.

In the graphics window, objects are drawn according to their geometric descriptions, and icons representing sensors and effectors appear automatically. Lines extending from the sensor/effector icons indicate graphically the intensity of stimulus and effect. It is possible to leave a trail along the object's prior path to more clearly represent their motion.

Multiple input choices, including keyboard, sliders, and (for orientation) a virtual sphere,[5] are available in temporary windows and can be used to alter the values in the window fields and the positions of graphical objects.

A temporary window can be called upon to show the current network for the active object (see Figure 5). By default, sensor icons appear along a leftward column of the network window, effector icons similarly along the right. The user can move these icons about and add nodes and connections between them.

Picking a transfer function node calls up another temporary window (Figure 1) which indicates the current transfer function for the node. Transfer functions are defined by adding, deleting, and moving points about on a cartesian grid. The network window and the transfer function windows zoom, pan, and scroll to show all sections clearly.

* * *

Behavioral animation is achieved by setting up the appropriate network and transfer functions, placing objects of interest under behavioral control, and setting animation control to *go*. Networks are synchronized, in the sense that there is an internal clock which, each time step, calculates new values. The sensors are set once at the beginning, then the nodes are pulsed until their data stabilizes (up to a maximum number of pulses), and then the effector values are calculated. Thus, a network consisting of a sensor, two nodes in sequence, and an effector will require two time steps for the stimulus of the sensor to reach the effector and cause a response. The user can set the actual motion (and display) time step to be any integral multiple of the network time step, as it may be desirable to give the network time to settle before producing new motion values for display.

4. Sample Behavior Patterns

4.1. Path Finding in Two Dimensions

Figure 5 shows the networks for an object that is attracted to blue cubes but desires to avoid collisions. Figure 2 shows the motion produced by this network. Considerable variation in the actual paths taken can be produced by changing parameters associated with sensors, nodes, and effectors. For example, changing the range over which the avoidance reaction occurs will prevent the objects from moving between the eight obstacles and force them to take the long way around to the goal white cube at the upper right.

4.2. Attraction and Avoidance in Three Dimensions

Figure 6 shows a similar attraction/avoidance behavior in three dimensions. The tall blocks are obstacles that do not move. The two cubes are attractors that cycle the cubes due to constant forces being applied to their effectors. The tetrahedrons are objects with senses. The tetrahedron that cycles in the center of the obstacles is attracted to everything except other tetrahedrons. The other tetrahedrons are attracted only to the moving cubes.

5. Discussion and Conclusions

The purpose of this system is rather to devise a useful tool for animating and exploring behavior, than to exactly simulate biological circuits. Very early it became clear that senses and nodes should be as high-level and sophisticated as possible, in order to simplify the user's task of building networks. This involves a conceptual move away from low-level Braitenberg vehicles and similar connectionist systems such as McCullough-Pitts neurons,[12] logic circuits, and neural nets.[15]

Behavioral animation is not expected to be the answer to all problems of specifying animation, any more than are any of the many other techniques such as key-positioning, hierarchical control, inverse kinematics, or dynamics. The ideal animation system should contain all of these tools. Behavioral control might be best used for generating group behaviors, or finding paths through an environment. Once a possible pattern is devised, other techniques could be brought in to refine the motion. Behavioral control is one tool among many to aid the animator in bringing his or her own vision to life.

References

1. Norman I. Badler, Kamran Manoochehri, and Graham Walters, "Articulated Figure Positioning by Multiple Constraints," *IEEE Computer Graphics and Applications*, vol. 7, no. 6, pp. 28-38, June, 1987.
2. Ronen Barzel and Alan H. Barr, "A Modeling System Based on Dynamic Constraints," *SIGGRAPH '88 Conference Proceedings*, vol. 22, no. 4, pp. 179-188, August, 1988.
3. Valentino Braitenberg, *Vehicles, Experiments in Synthetic Psychology*, The MIT Press, Cambridge, Massachusetts, 1984.
4. Lynne Shapiro Brotman and Arun N. Netravali, "Motion Interpolation by Optimal Control," *SIGGRAPH '88 Conference Proceedings*, vol. 22, no. 4, pp. 309-315, August, 1988.
5. Michael Chen, S. Joy Mountford, and Abigail Sellen, "A Study in Interactive 3-D Rotation Using 2-D Control Devices," *SIGGRAPH '88 Conference Proceedings*, vol. 22, no. 4, pp. 121-130, August, 1988.
6. David Forsey and Jane Wilhelms, "Manikin: Dynamic Analysis for Articulated Body Manipulation," *Graphics Interface '88*, June, 1988.
7. Allen Van Gelder, 1988. Personal Communication.
8. Michael Girard and Antony A. Maciejewski, "Computational Modeling for the Computer Animation of Legged Figures," *SIGGRAPH '85 Conference Proceedings*, vol. 19, pp. 263-270, July, 1985.
9. James K. Hahn, "Realistic Animation of Rigid Bodies," *SIGGRAPH '88 Conference Proceedings*, vol. 22, no. 4, pp. 299-208, August, 1988.
10. Paul M. Isaacs and Michael F. Cohen, "Controlling Dynamic Simulation with Kinematic Constraints," *SIGGRAPH '87 Conference Proceedings*, July, 1987.
11. James U. Korein and Norman I. Badler, "Techniques for Generating the Goal-Directed Motion of Articulated Structures," *IEEE Computer Graphics and Applications*, vol. 2, no. 9, pp. 71-81, November, 1982.
12. W. S. McCulloch and W. H. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-133, 1943.
13. Matthew Moore and Jane Wilhelms, "Collision Detection and Response for Computer Animation," *SIGGRAPH '88 Conference Proceedings*, vol. 22, no. 4, pp. 289-298, August, 1988.
14. Craig W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model," *SIGGRAPH '87 Conference Proceedings*, vol. 21, pp. 25-34, Association for Computing Machinery, July, 1987.
15. David Rumelhart, James McClelland, and the PDP Research Group, *Parallel Distributed Processing, Volumes 1 & 2*, The MIT Press, Cambridge, MA, 1986.
16. Demetri Terzopoulos, John Platt, Alan H. Barr, and Kurt Fleischer, "Elastically Deformable Models," *SIGGRAPH '87 Conference Proceedings*, July, 1987.
17. Jane Wilhelms, "Using Dynamic Analysis for Animation of Articulated Bodies," *IEEE Computer Graphics and Applications*, vol. 7, no. 6, June, 1987.
18. Jane Wilhelms, "Dynamics for Computer Graphics: A Tutorial," *Computing Systems*, pp. 63-93, USENIX Association, Winter, 1988. also UCSC Computer and Info. Sci., Tech. Report UCSC-CRL-87-5
19. Andrew Witkin, Kurt Fleischer, and Alan H. Barr, "Energy Constraints on Parameterized Models," *SIGGRAPH '87 Conference Proceedings*, (Anaheim, CA, July, 1987).
20. Andrew Witkin and Michael Kass, "Spacetime Constraints," *SIGGRAPH '88 Conference Proceedings*, vol. 22, no. 4, pp. 159-168, (Atlanta, GA, August, 1988).

21. **David Zeltzer,** "Motor Control Techniques for Figure **Animation,"** *IEEE Computer Graphics and Applications,* **vol. 2, no. 9, pp. 53-60, November, 1982.**

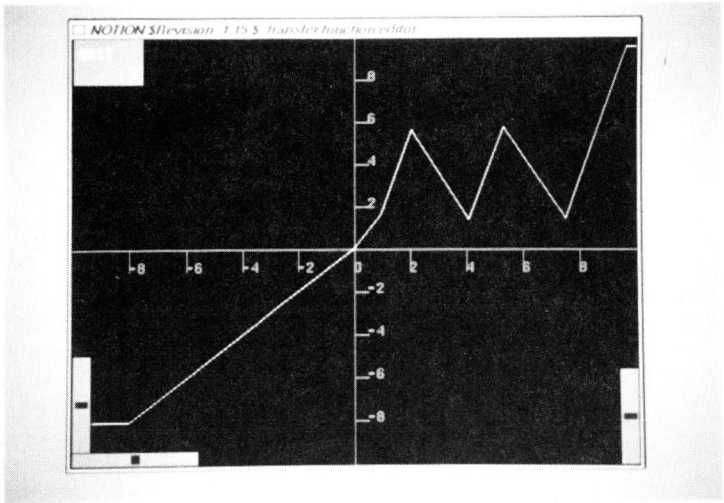

Figure 1. Transfer Functions

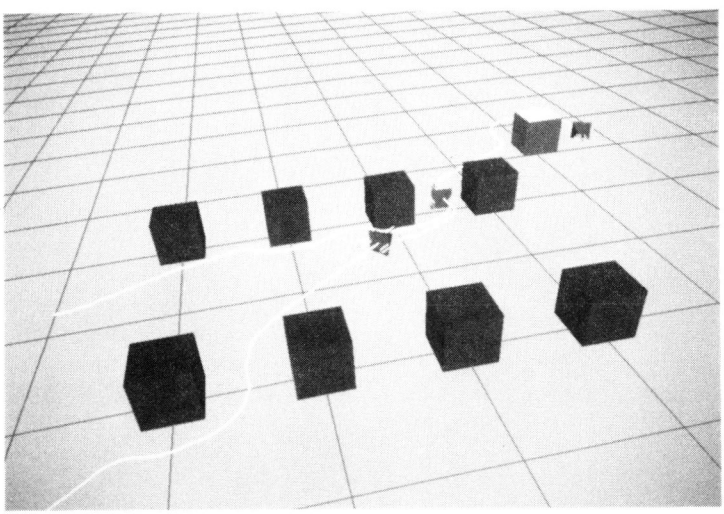

Figure 2. The Graphics Window

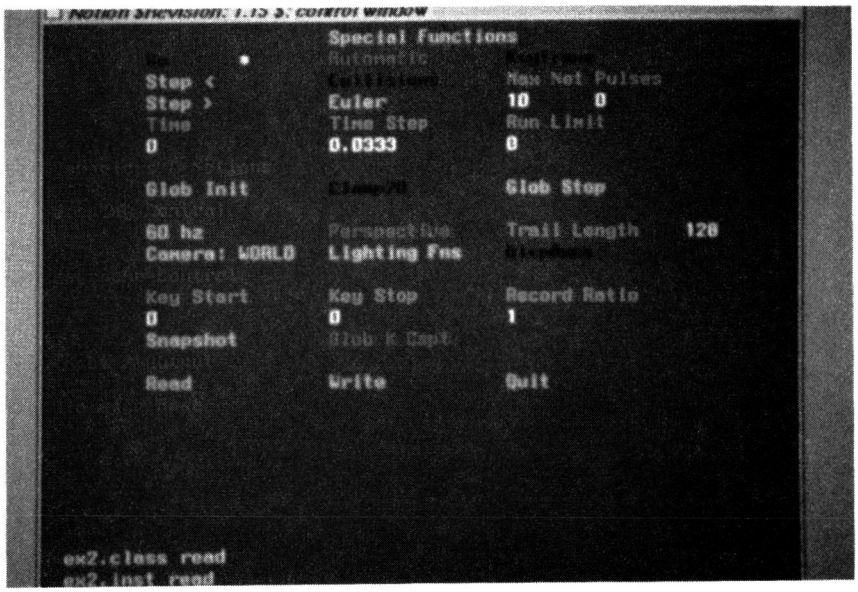

Figure 3. The Animation Control Window

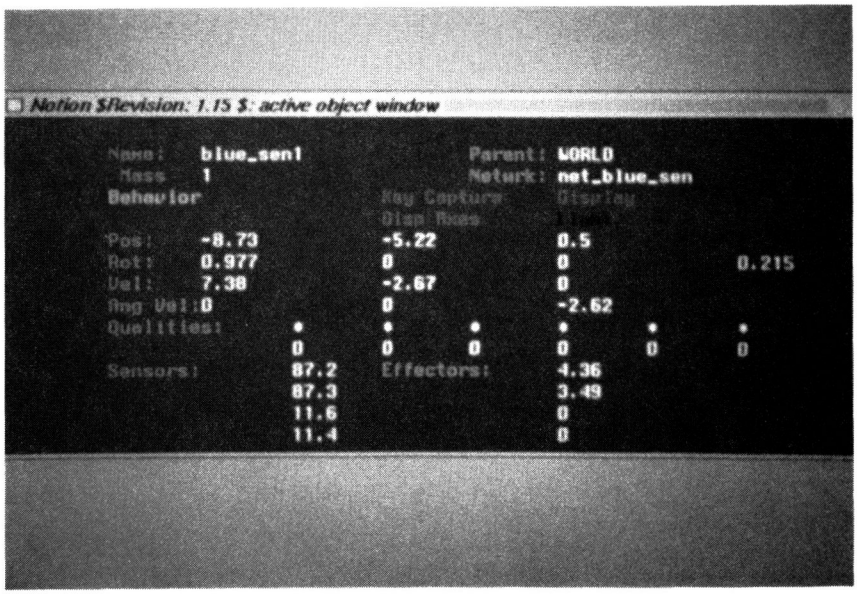

Figure 4. The Active Object Window

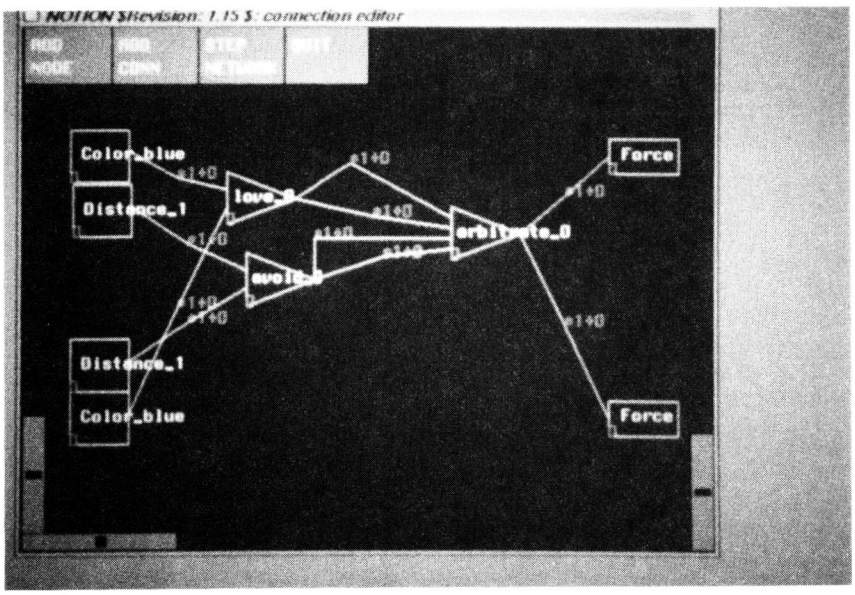

Figure 5. The Network Window

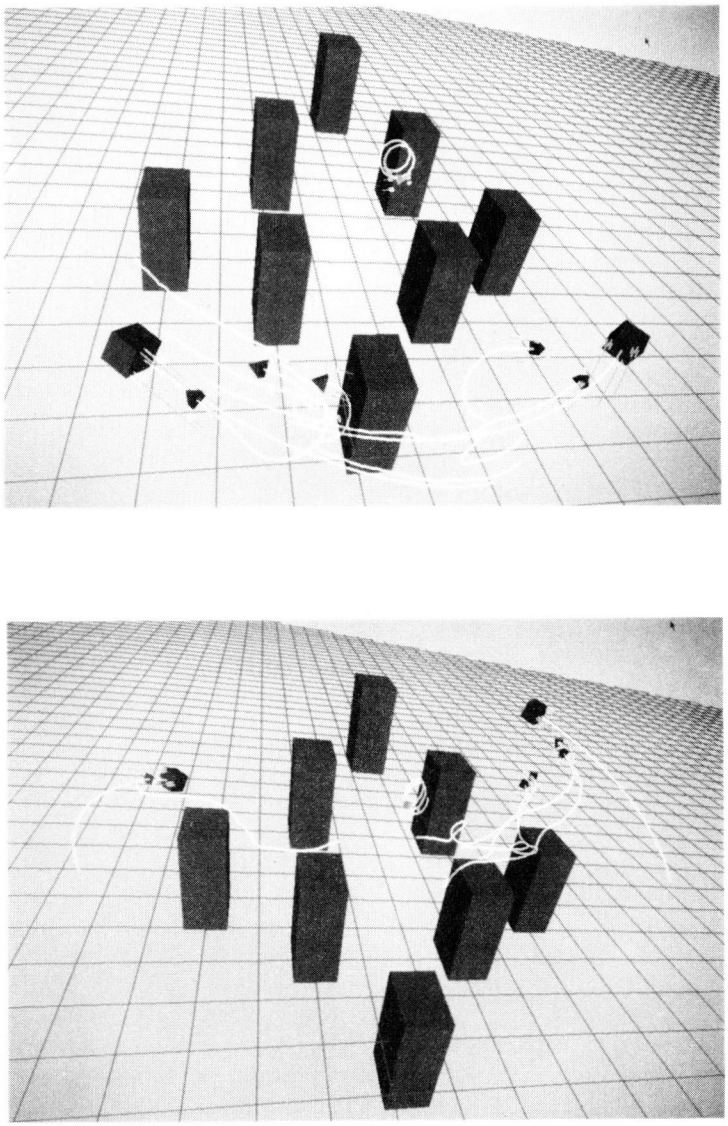

Figure 6. Three Dimensional Attraction/Avoidance Behavior