An Architecture for Concurrent Reactive Agents in Real-Time Animation

MÔNICA COSTA
BRUNO FEIJÓ

ICAD - Laboratório de CAD Inteligente
Departamento de Informática, PUC-Rio
Rua Marquês de São Vicente, 225
22453-900 Rio de Janeiro, RJ, Brasil
{monica,bruno}@icad.puc-rio.br

PUC-RioInf.MCC32/96 September, 1996

Abstract. This paper proposes an architecture for real-time behavioral animation based on parallel interactions between simple recursive reactive agents and allowing for integration with external articulated figure software. An animated sequence of a navigation scene where two actors play different roles is generated by a prototype.

Keywords: Behavioral Animation, Real-Time Animation, Reactive Agents, Parallel Programming

Resumo. Este artigo propõe uma arquitetura para animação comportamental em tempo real baseada em interações em paralelo entre agentes reativos recursivos simples que permite a integração com programas de figuras articuladas. Uma seqüência animada de uma cena de navegação onde dois atores atuam em diferentes papéis é gerada por um protótipo.

Palavras-chave: Animação Comportamental, Animação em Tempo Real, Agentes Reativos, Programação Paralela

1 Introduction

In behavioral animation, actors perform complex tasks and are able to react to external stimuli and events, according to its own beliefs, intentions, humor, fears and feelings. Behavioral animation can be used in a number of cases, such as virtual prototyping in engineering design; hazard simulation; internet virtual meetings; and entertainment industry applications. Agent technology has emerged as a promising approach to behavioral animation [Bates (1994)][Terzopoulos et al. (1994)] [Tosa (1993)]. Costa et al. (1995) propose a model of simple recursive reactive agents for behavioral animation systems.

The performance of behavioral agents should happen in a rich and complex interactive 3D virtual world and, above all, in real-time. Creatures interacting with their environment in real-time have been emerging as one of the new trends in computer graphics. Wilhelms and Skinner's (1990) system is an interesting example of interactive behavioral animation control. Granieri et al. (1995) present implementation issues for real-time visual simulation of multiple animated synthetic human figures. Badler et al. (1993) and Badler (1995) present the state-of-the-art in simulating humans and explore several issues in real-time animation. High performance multiprocessing toolkits are becoming popular for real-time 3D graphics, such as IRIS Performer [Rohlf and Helman (1994)]. Furthermore, general-purpose parallel programming systems have been used in computer graphics, such as PVM [Geist et al. (1994)] adopted in this work. The present paper follows those trends and proposes an architecture for real-time behavioral animation based on parallel interactions between reactive agents and allowing for integration with external articulated figure software.

2 Actors as Reactive Agents

In a previous work [Costa et al. (1995)], the authors presented actors in behavioral animation as reactive agents with the structure shown in Fig.1. Following that proposal, agents are driven by motors which are themselves agents with the same structure. The elements of this structure are drawn on general principles of cognition [Stillings et al. (1987)].

The Sensory Centre has two kinds of basic functions: (1) functions to send and receive messages; (2) sensory perception functions. An agent is activated by a message sent by its parent-agent. Most of the time, this message is passed to the agents' motors by the cognition centre, in order to distribute tasks. A motor always reports success or failure to the agent that called it. The other kind of basic function, i.e. the sensory perception function, detects events in the virtual environment associated to vision, hearing and touch. As far as vision is concerned, stereoscopic capacity is not required because the distances between objects can be calculated straightforwardly. Also no pattern recognition is required because the list of the objects is available to the character. These two latter assumptions highlight the differences between behavioral animation and other simulation areas such as artificial life. In behavioral animation there is no extra bonus in simulating cognitive processes of vision and, consequently, visual information is structured in advance.

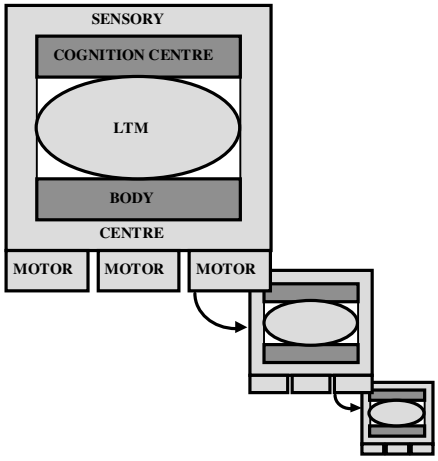

Fig.1 The agent structure

In accordance to the cognitive science, the LTM (Large Term Memory) of an agent is a window to a vast declarative memory area with facts specified by the animator and facts perceived by the character during its existence in the virtual environment. Sometimes there are facts that are common to more than one agent. Only an especial agent called the Universal Agent has the consciousness of the entire factual data base.

The facts in the LTM are inert structures that should be operated by processes in the Cognition Centre. Processes are procedural knowledge of two types: controlled and automatic procedures (Fig.2). Controlled procedures require conscious attention like an interpreter. Automatic procedures are like compiled programs automatically triggered by events or goals. The Logical Procedures in Fig.2 are sentences in mathematical logic. They are used in situations where deductive thought is required in specific domains.

General path-planning with low degree of details is usually done by logical procedures.

Learned Procedures represent reactive plans encoded as compiled programs. These plans are continuously revised and, consequently, can adapt themselves to unexpected events that may occur in the environment. The name Learned Procedure comes from the fact that these procedures represent learned skills with no need for conscious attention.

Behavioral functions are primitive forms of automatic procedures defined by a single expression. They are used by agents that should react in a stimulus-response basis. Sometimes simple creatures are defined by a single agent and just one behavioral function, such as a very small insect flying around a lamp.

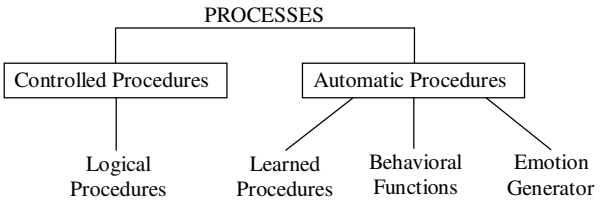

Fig.2 Processes in the Cognition Centre

The emotion generator operates on the LTM to generate emotional states from primitive emotional propositions.

The Body contains information about the physical structure of the character to which the agent belongs. Only the agents in the very low end of the hierarchy tree contain this sort of information.

Usually an agent has certain parts always empty. Decoration objects, for instance, usually have only bodies and no cognition or sensory centres. The agent controlling the human gait may have a sensory centre (to receive stimulus), a cognition centre with just one behavioral function (to perform the gait), a small LTM and a body.

In the proposed model, every visible object, from decoration artifacts to living characters, is an agent with the recursive structure shown in Fig.1. In this paper, the authors refer to actors as the topmost agents representing living characters which are directly under the Universal Agent's supervision.

3 A Concurrent Reactive Agent Architecture

The starting point for the proposed real-time architecture is to organize it in terms of two views connected by a generic interface layer, as shown in Fig.3.

The ***behavioral view*** defines the actors and is implemented by an ***agent tree*** where each node has the structure shown in Fig.1 and represents a concurrent process (Fig.4). The leaves of the tree in Fig.4 are the lowest level motors which should activate motion processes in the ***motion view***. The motion view in the proposed concurrent reactive agent architecture of Fig.4 is composed of a real-time articulated figure application. Each motor in the lowest level of the agent tree sends a set of basic parameters $\{p_i\}$ to the ***animation bus*** which should convert them into a synchronized sequence of basic manipulations for the articulated figure application. Therefore, the animation bus is an interface layer that controls the application in the motion view and sets up the motion clock used to synchronize detailed movements. As a collection of functions and interface toolkits, the animation bus is supposed to meet portability requirements.

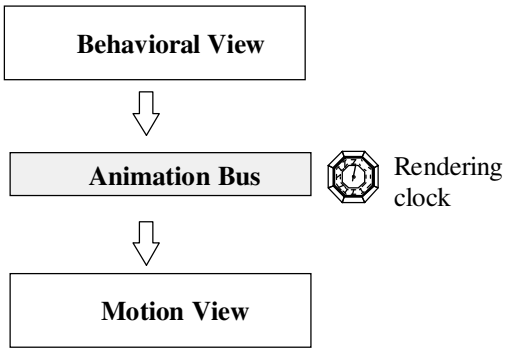

Fig.3 Views

Also, Fig.4 shows an example where two concurrent low-level agents send basic parameters to the animation bus concerning motion units for the right foot and the left arm. Suppose that the first motion starts at $t_b = 0.4\ s$ and the latter at $t_b = 0.8\ s$. These motion units cannot be visualized in the same moment the concurrent agents liberate them to the animation bus. Before being visualized, they must be converted into more complex detailed movements (e.g. raising the right foot causes leg movements and changes balance). Furthermore, the proper instructions to the articulated figure application must wait for the next clock tick in the animation bus. Suppose that the clock tick is $T = 1\ s$. In this case, time t_m is shifted by T in the motion view. Therefore, T is a natural delay in the response time of the proposed system.

The behavioral view has an important property called ***granularity*** which determines the degree of detail for the interactions between agents. Fig.5 illustrates a coarse granularity for the behavioral view, where actors are represented spatially by their bounding boxes and the only motion parameters sent to the animation bus are {*time interval, translation vector, rotation vector*}. These parameters are sent to the motion view, where detailed anthropometry-based movements and a realistic rendering are calculated and exhibited. This coarse granularity might be adequate for an evacuation simulation in an off-shore oil platform in fire, where no precise interactions between body segments are required for a realistic animation. .

paper depends on the sense functions in the sensory centre of the agents and on the decision-making algorithms that provide proper reactions to the external stimuli. The code for vision has a permanent loop where position information is asked for all actors other than the actor who owns the vision process. For each one of those actors, it is necessary to let each agent in the sense group to decide whether or not the visual stimulus will be used and to react properly if needed. The present paper has not investigated on the possibility of having a mathematical expression for the reactive capacity.

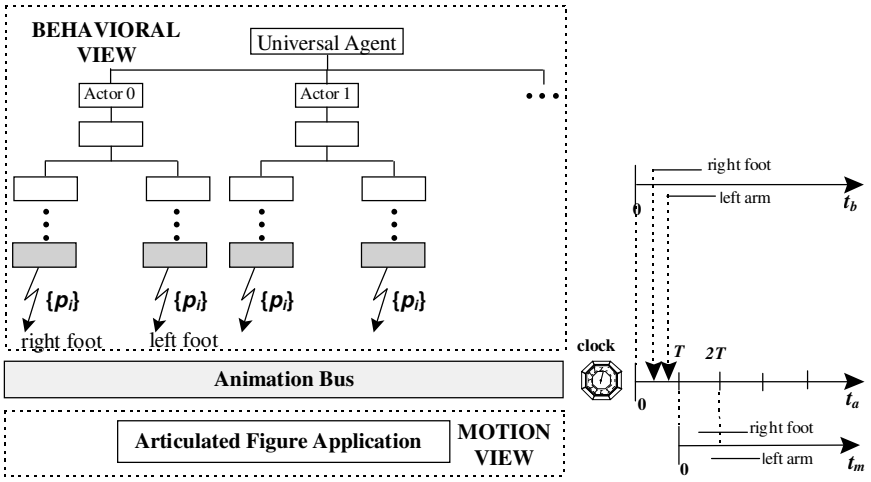

Fig.4 The Concurrent Reactive Agent Architecture and the time axis for different views

According to the adopted model for reactive agents, any agent in the agent tree can have any reactive sense in its sensory centre. In the implemented prototype, agents with vision in their sensory centre are organized in terms of ***sense groups***. Because more than one agent within the same actor can have vision capacity, the authors follow the strategy of notifying the topmost agents firstly. Fig.6 illustrates a virtual environment where two actors and two vision groups respond constantly to visual stimuli. In this case, when an agent decides to use the visual stimulus it asks for additional information (e.g. size, color, …).

The ***reactive capacity*** is the minimum time step that allows all the actors to be aware of the virtual environment in which they are immersed. A slow machine running a very complex behavioral view with very fast objects can produce large gaps in the continuity of movements and, worse than that, the actors can miss events entirely. The reactive capacity of a system based on the architecture proposed in this

4 PVM

In order to implement the concurrent nature of the proposed architecture, the authors use the Parallel Virtual Machine (PVM) system [Geist et al. (1994)]. PVM is a portable message-passing programming system which allows programmers to exploit distributed computing across a wide variety of computer types, including workstations, multiprocessors and PCs.

A *task* in PVM is defined as a unit of computation analogous to a UNIX process. The ability of PVM functions to automatically start up new tasks on the virtual machine (see *pvm_spawn* in the example below), provides a natural means for initiating new actors when required and building the agent tree where each agent is a different task.

PVM also supplies functions that allow the tasks to communicate and synchronize with each other. This feature is of fundamental relevance in the implementation of the concurrent reactive agent

architecture, provided that the intelligence exhibited by the agents is mainly a result from their interactions within the virtual environment. Examples of these functions are *pvm_send*, *pvm_recv*, *pvm_nrecv*, and some others depicted in Fig.8. In particular, the non-blocking character of *pvm_nrecv* lets the agents with sensory perception functions to continuously test for the arrival of messages from these functions without interrupting execution and, therefore, granting for reactivity.

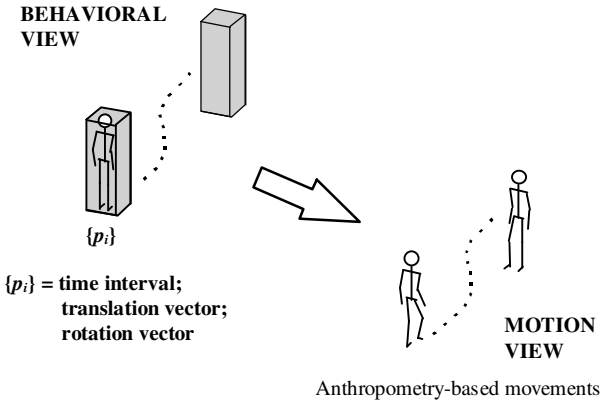

Fig.5 Example of a coarse granularity in the behavioral view

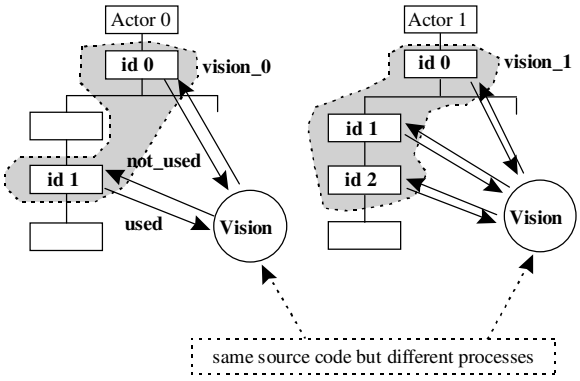

Fig.6 Sense Groups

Another important set of PVM functions is formed by the dynamic process group functions (e.g. *pvm_joingroup*, *pvm_lvgroup*, *pvm_gettid*, *pvm_gsize*, ...). Any PVM task can join or leave any group at any time. Groups are identified by a string of characters and each task in a group has a unique instance number which runs from 0 to the number of members in the group minus 1. The ability of dealing with dynamic process groups yields to the implementation of sense groups. In the case of vision, each actor has its own vision process and an associated vision group. Agents with vision within the same actor share the same vision process and are members of the vision group associated with that actor (see Fig.6). Each time the vision process detects any other agent in the field of view of the actor, the agents in the associated vision group are notified by a message until an agent decides to use the visual stimulus.

5 Example of Reactive Navigation

A reactive navigation example, where two actors play different roles in the scene, can be implemented with the agent hierarchy defined in Fig.7. The agents in the agent tree are essentially the same presented in a previous work by the authors [Costa et al. (1995)]. The two actors are capable of moving themselves in a building. Having in "mind" a destination room, they can generate a high-level navigation plan (with no details) based on mathematical logic (agent **go_to**). The agent **follow_path** follows a planned path, changing it in order to avoid obstacles. The agent **face** points the character to a new direction in order to avoid a collision. The agent **move** takes the character to a certain position along a straight line. This agent has a vision function in its sensory centre in order to detect obstacles. Once it detects an obstacle in the environment, it informs its parent agent (follow_path) who decides whether or not it will continue in the same direction . The agents **turn** and **step** perform small rotation and translation movements. Learned procedures determine the size of these basic movements. In the example implemented by the authors, the size of the basic movements is constant and arbitrary.

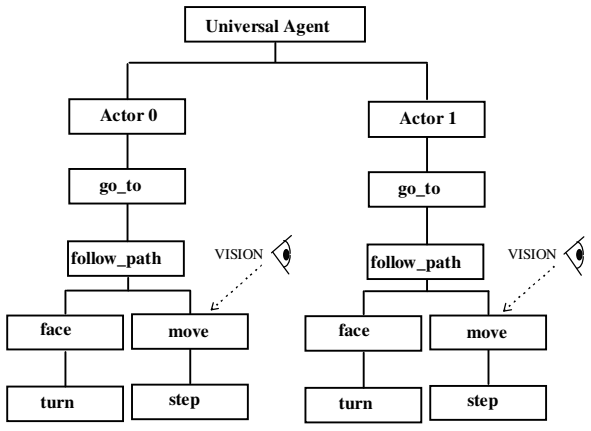

Fig.7 An agent hierarchy for two actors playing in a navigation scene

In the prototype developed by the authors, the actors are identified by a vector named **actors[n]** which contains all the data related to them. Actually, this is the LTM of the Universal Agent and, whenever an agent in the tree needs to read or modify some information in its own LTM, the process that represents that agent sends the appropriate message to the Universal Agent. In this manner, all the data are maintained centralized and consistent.

The actor data are very simple, such as **name**, **radius** of a bounding cylinder, position *from* and *to* (**fpos** and **tpos**), direction *from* and *to* (**fdir** and **tdir**), number of descendant agents with vision (**nvision**) amongst others. The types of messages are classified as DATA, QUERY, UPDATE, SENSE_USED, SENSE_NOT_USED and others. Each message corresponds to an integer number.

Each concurrent reactive agent in the hierarchy of Fig.7 has a similar structure in terms of implementation. Fig.8 illustrates the concurrent mechanism for the agent **face**.

```
void   main(void)
{
 my_tid = pvm_mytid()                  /*get process id of the agent*/
 parent_id = pvm_parent();             /*get process id of its parent*/
 pvm_recv(parent_tid,DATA); /*receive data from its parent*/

 pvm_upkint(&actor_id,1,1);            /*unpack data*/
 pvm_upkint(&ua_tid,1,);

 /*face starts its motor: agent turn*/
 if ((ntask = pvm_spawn("turn", …,&turn_tid)) <= 0)
 {
  warn parent about the failure
  pvm_exit();
  exit(1);
 }
 pvm_initsend(…);                      /*open for sending*/

 pvm_pkint(&actor_id,1,1);             /*pack data*/
 pvm_pkint(&ua_tid,1,1);

 pvm_send(turn_tid,DATA);              /*send data*/

 /*loop while receiving messages from its parent*/
 while(buf_id = pvm_recv(parent_tid,-1))
 {
  pvm_bufinfo(buf_id,NULL,&msg_tag,NULL);
  if(msg_tag == ACT )
  {
   if(Face() == 1)                     /*Face is a learned procedure*/
   {
    pvm_initsend(…);                   /*open for sending*/
    pvm_send(parent_tid, SUCCESS);
   } else;    /* Face is always expected to succeed*/
  }
  else if (msg_tag == TERMINATE)
  {
   stop motor turn and exit
  }
 }
}

/*Learned Procedures*/

int     Face(void)
{
 get data fdir and tdir
 calculate the rotation step (turn-ang) to be sent to the agent turn
 while(turns are needed)
 {
  pvm_initsend(…);
  pvm_send(turn_tid,ACT);              /*ask turn to act*/
  buf_id = pvm_recv(turn_tid,-1);      /*receive message from turn*/
  pvm_bufinfo(buf_id,NULL,&msg_tag,NULL);
  if(msg_tag == SUCCESS)
  {
   calculate new fdir
   pack fdir
   pvm_send(ua_tid,UPDATE);           /*tell the UA to update fdir*/
  }
  else;        /*turn is always expected to succeed*/
 }
 return(1);
}
```

Fig.8 The Concurrent Agent **face**

The articulated figure application used in the prototype is the software Jack [Badler et al. (1993)]. In the example, the basic information of translation and rotation movements are converted into very simple manipulations of the end effectors of the Jack human figure. Those manipulations are a couple of feet translations and pelvis rotations for each step. A typical command generated by the animation bus is as follows:

create_foot_motion (right,t1,t2,distance,height,...)

where the right foot is asked to move by a certain distance starting in t1 and finishing at t2, reaching a given maximum height. The animation bus also schedules functions to stop and start Jack's motion system according to the clock.

Fig.9, Fig.10, and Fig.11 show two Jack human figures avoiding one another and walking to their respective destination rooms in the house.

The prototype is implemented in C in an Indigo2 machine.

6 Conclusions

This paper presents a concurrent reactive agent architecture for real-time animation based on emergent computation - i.e. behavior resulting from communication of independent components. A prototype is built and a number of positive conclusions emerges.

A parallel system can pose problems if there is no central control. However, this is not the case, because the Universal Agent plays the role of a monitor and helps controlling the system. The organization of the architecture in terms of the behavioral view and the motion view, connected by an animation bus, proved to be flexible and adequate. Indeed, sometimes a simple behavioral view (with a coarse granularity) can cope with a variety of situations and the burden of generating secondary detailed movements can be transferred to an external application.

PVM revealed itself as a robust, rich and adaptive programming paradigm. The concurrent mechanism is easily implemented, specially because manipulation of dynamic groups is a straightforward task.

There is a number of open issues. The architecture requires an agent language or a formalism to help establishing properties, creating reuse objects and calculating reactive capacity. Some theoretical approaches to behavioral animation systems are very promising, such as the Parallel Transition Networks by Granieri et al. (1995). However, most of these systems are not parallel in all levels. Finally, the animation bus is only tested as an ad-hoc module and a lot of efforts should be put into the implementation of this concept.

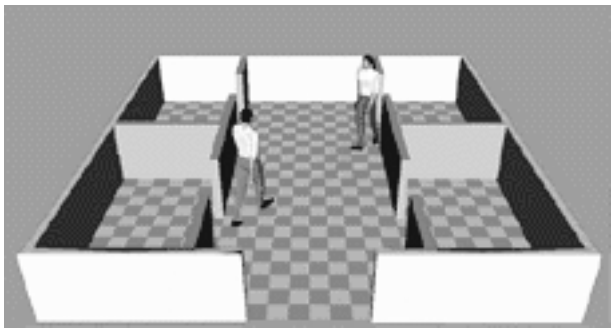

Fig.9

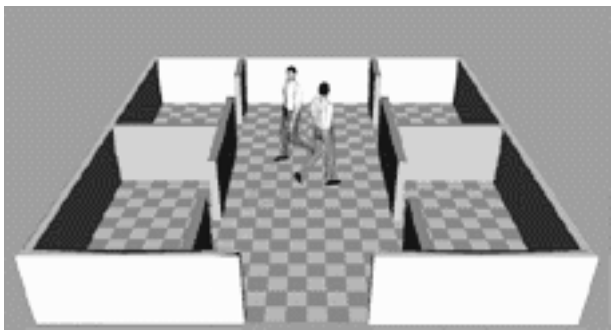

Fig.10

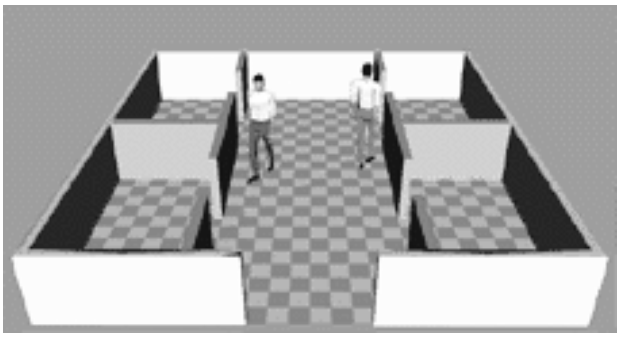

Fig.11

Acknowledgments

The authors would like to thank the CNPq for the financial support and Hélio Magalhães for the technical support in the generation of the images. Also thanks are due to the reviewers for their valuable comments.

References

N. I. Badler et al., *Simulating humans: computer graphics, animation, and control*, Oxford University Press (1993).

N. Badler, "Interactive humans: from behaviors to agents", *Dynamic Behaviors for Real-Time Synthetic Humans, Course 11 Notes of SIGGRAPH'95* (1995).

J. Bates, "The role of emotion in believable characters", *Communications of the ACM* 37(7) (1994).

M. Costa et al., "Reactive agents in behavioral animation", *Proceedings of SIBGRAPI'95* (1995), 159--165.

A. Geist et al., *PVM 3 Users's Guide and Reference Manual*, ORNL/TM-12187 (1994).

J. P. Granieri et al., "Behavioral control for real-time simulated human agents", *Proceedings of 1995 Symposium on Interactive 3D Graphics* (1995), 173--180.

J. Rohlf and J. Helman, "IRIS Performer: a high performance multiprocessing toolkit for real-time 3D graphics", *Proceedings of SIGGRAPH'94* (1994), 381--395.

N. A. Stillings et al., *Cognitive Science: an Introduction*, MIT Press (1987).

D. Terzopoulos et al., "Artificial fishes with autonomous locomotion, perception, behavior and learning, in a physical world", *Proceedings of the Artificial Life IV Workshop*, P. Maes and R. Brooks (eds.), MIT Press (1994).

N. Tosa, "Neurobaby", *Visual Proceedings of SIGGRAPH'93*, Tomorrow's Realities, ACM SIGGRAPH (1993), 212--213.

J. Wilhelms and R. Skinner, "A "Notion" for interactive behavioral animation control", *IEEE Computer Graphics and Applications* (1990), 14--22.